Journal of Computers Vol. 31 No. 4, 2020, pp. 28-41
doi:10.3966/199115992020083104003

A Novel Hyper-Heuristic with Decomposition and Mathematical Programming

Zongzheng Chi, Xiaochen Lai, Shuwei Zhang, He Guo, Zhilei Ren*

School of Software, Dalian University of Technology, Dalian 116621, China
czz.dut@163.com, zren@dlut.edu.cn

Received 6 December 2018; Revised 28 August 2019; Accepted 11 October 2019

Abstract. Hyper-heuristics aim to automate the adaptation of heuristics to solve hard computational search problems, which solves problems from different domains by manipulating the domain-specific Low Level Heuristics (LLHs). However, despite the promising results obtained by various hyper-heuristics, when faced with new problem domains, the design and implementation of the LLHs require significant expense. Moreover, the intrinsic complexity of large scale problem instances also poses great challenge for hyper-heuristics, which makes the solving process time-consuming. In this paper, we propose a novel Decomposition and Mathematical Programming based Hyper-heuristic (**DEMPH**), which features the combination of the decomposition based problem solving and the generalization ability of mathematical programming solvers. We develop strategies to decompose the large scale problem instances to sub-problems, and apply the mathematical programming solver to solve the sub-problems. An adaptive high level strategy is employed to schedule the sub-problem selection strategies. Experiments on the generalized assignment problem and the three-index assignment problem indicate that **DEMPH** is able to tackle large scale problems efficiently, and generalize well on different type of instances.

Keywords: AP3, decomposition, GAP, hyper-heuristic, mathematical programming

1 Introduction

Recent years have witnessed the increasing popularity of hyper-heuristic, an emerging technique that aims to tackle complex problems, meanwhile remain easy and fast to implement. By separating the domain-independent decision making High Level Strategy (HLS) and the domain-specific Low Level Heuristics (LLHs), hyper-heuristics have the potential of cross-domain problem solving. Meanwhile, hyper-heuristics provide a unified framework to combine the strength of different LLHs, and countervail the weakness of single heuristic. With the promising achievements accomplished in the past decade, hyper-heuristics have been applied to various problem domains, such as the Satisfiability problem [1], the Traveling Salesman Problem [2-3], the p-Median problem [4], and various scheduling problems [5-6].

However, despite the great success, there are still limitations in solving complex problems with hyper-heurists, among which the following two challenges deserve in-depth study. (1) In most existing hyper-heuristics in the literatures, the LLHs are mostly crafted by both domain experts and algorithm developers. Consequently, great efforts have to be taken to design and implement the domain-specific LLHs. (2) When solving large-scale problem instances, most existing hyper-heuristics directly solve the whole problem instance, which might be ineffective, and easily trapped by local optima.

To tackle the challenges, we propose a novel hyper-heuristic algorithm, which features the combination of two main mechanisms, i.e., the Mathematical Programming (MP) based LLH and the decomposition based large scale instance solving. On the one hand, to address the challenge of design and implementation efforts for cross-discipling solving, we incorporate state-of-the-art optimization solvers in the hyper-heuristic framework. Thanks to the advancement of operations research, modern

* Corresponding Author

Mathematical Programming (MP) solvers have enabled the solving of problems from various domains. With the richly expressive application programming interface provided by MP solvers, it is straightforward to realize cross-discipline problem solving. On the other hand, to face the challenge of large-scale solving, we combine the MP solver with a set of decomposition based sub-problem generation LLHs. Motivated by the concept of very large scale neighborhood (VLSN) [7], we propose a set of LLHs, which work by partially fixing the decision variables, and solve the derived sub-problem instance to optimality by the MP solver. Furthermore, to mitigate the limitation of static LLH parameter configuration, we propose a Variable Sub-problem Scale Adaptation (VSSA) mechanism. VSSA adaptively adjusts the scale of the sub-problems for the LLHs, which helps promote generalization ability of the proposed framework.

By integrating the MP solver, the decomposition based LLHs, and the VSSA mechanism, we propose the **DE**composition and **M**athematical **P**rogramming based **H**yper-heuristic (**DEMPH**). The unique feature of **DEMPH** lies in the ability of solving large scale instances from different problem domains, and its adoption of the MP solver as LLH. To evaluate the proposed algorithm, we consider two well-known combinatorial optimization problems, i.e., the three index assignment problem (AP3) and the generalized assignment problem (GAP). Extensive numerical experiments demonstrate that, **DEMPH** is able to achieve promising results over large scale instances, for the two different problem domains.

The contributions of the paper could be summarized as follows:

(1) First, we propose a novel hyper-heuristic algorithm **DEMPH**, which combines the decomposition mechanism and mathematical programming solver. With the generalization power of mathematical programming and the decomposition based sub-problem generation mechanism, **DEMPH** is able to effectively tackle large-scale problem instances

(2) Second, we develop a variable sub-problem scale adaptation mechanism, to mitigate the limitation of static configuration of problem decomposition.

(3) Third, to evaluate the proposed algorithm, extensive experiments are conducted over large scale problem instances of two problem domains. Numerical results demonstrate that, the performance of **DEMPH** is comparable to the state-of-the-art algorithms.

The rest of the paper is structured as follows. In Section 2, we give the background of hyper-heuristic. In Section 3, we present the details of the proposed **DEMPH** algorithm. In Section 4, experimental results are presented and discussed. Finally, in Section 5, conclusion and future directions are addressed.

2 Related Work

2.1 Hyper-Heuristic

Hyper-heuristic is defined as “an automated methodology for selecting or generating heuristics to solve hard computational search problem” [8]. Typical hyper-heuristics consist of two layers, i.e. the High Level Strategy (HLS) and the LLHs. For the two layers, the HLS models the problem solving process as a scheduling problem, which aims to choose the most suitable LLH according to the state of the search process. Meanwhile, with respect to each problem domain for which the hyper-heuristic is designed, a set of LLHs are proposed and implemented, which are to be invoked by the HLS. By combining the HLS and the LLHs, hyper-heuristics have the potentials to realize cross-discipline solving effectively. Since the 1990s, hyper-heuristic has rapidly gained much research interest, and has been applied to solve many problems, such as the Timetabling problem [9], the Bin Packing problem [10], the Knapsack problems [11], etc. According to the comprehensive survey [8], the existing hyper-heuristics could be classified into the constructive LLH based approaches, and the perturbative LLH based approaches. For the former category, each LLH works by assigning values for the decision variables, to gradually construct the final solution. For the latter category, both the input and the output of the LLHs are solutions to the problem instances. Typical examples of perturbative LLHs include local search, crossover, mutation, etc. In particular, the perturbative LLH based hyper-heuristics are closely related to the adaptive operator selection mechanisms in memetic algorithms, and iterated local search algorithms, in that operators are manipulated in a similar paradigm in these algorithms [8]. In this study, we would focus on the latter category, due to its promising effectiveness and the generalization capability.

Compared with the existing hyper-heuristics, the unique feature of **DEMPH** lies in the combination of decomposition mechanism and mathematical solver as LLHs. To the best of our knowledge, this is the

first attempt in the hyper-heuristic community. The most relevant study is proposed in [12]. However, the Math-hyper-heuristic is a two phase approach. In the first stage, the instance is decomposed into several sub-problems, and the column generation algorithm is applied to pre-solve the sub-problems. Then, in the second stage, the solutions to the sub-problems are combined, and used as the initial solution of the hyper-heuristic. In particular, the decomposition mechanism in Math-hyper-heuristic is specific to the problem domain (the Vehicle Routing Problem), which is not straightforward to be generalized to other problem domains. In contrast, in this study we consider the fix-and-dive mechanism [13] (see Section 3.2 for detail), which has been widely used in mathematical programming based heuristics, and could be generalized to various problem domains.

2.2 GAP

The GAP is first proposed by Ross and Soland, and has various applications, from data storage and retrieval in disks, to caching in distributed systems [14]. The problem can be viewed as an assignment problem that assigns n different jobs to m agents. Each job is assigned to a single agent, subject to the agent's capacity. The GAP can be defined as an integer programming model:

$$\min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}, \tag{1}$$

subjects to

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

$$\sum_{j \in J} r_{ij} x_{ij} \leq b_i, \forall i \in I, \tag{3}$$

$$x_{ij} \in \{0,1\}, \forall i \in I, \forall j \in J, \tag{4}$$

where $I = \{1, 2, 3, 4, \ldots, m\}$, $J = \{1, 2, 3, 4, \ldots, n\}$. I can be viewed as the set of agents, while the J can be viewed as the set of jobs. b_i is the capacity of agent i, while r_{ij} is the resource for i assigned with job j. Without loss of generality, the solution of GAP can also be represented by a vector p, i.e., the solution of this instance is represented as a list of tuples: $\{(p_1, 1), (p_2, 2), \ldots, (p_n, n)\}$, where each tuple indicates an assignment of a job. To solve the GAP, extensive methods have been proposed in the past decades, including branch-and-bound [14], genetic algorithm [15], Tabu search [16], fuzzy programming [17], branch-and-price [18] etc. Among these approaches, the parallel genetic algorithm [19] achieves the state-of-the-art results, which features the ability to leverage multi-core/many-core computing environment.

2.3 AP3

The AP3 is first introduced by Pierskalla [20], which can be defined as an integer programming model:

$$\min \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} x_{ijk} = 1, \forall i \in I, \tag{5}$$

subjects to

$$\sum_{j \in J} \sum_{k \in K} x_{ijk} = 1, \forall i \in I, \tag{6}$$

$$\sum_{i \in I} \sum_{k \in K} x_{ijk} = 1, \forall j \in J, \tag{7}$$

$$\sum_{i\in I}\sum_{j\in J} x_{ijk} = 1, \forall k \in K, \tag{8}$$

$$x_{ij} \in \{0,1\}, \forall i \in I, \forall j \in J, \forall k \in K, \tag{9}$$

where $I = J = K = \{1, 2, 3, 4, \ldots, n\}$. Similar with the GAP, the solution of AP3 can also be represented with permutations: each solution comprises two permutations p and q, indicated by $\{(1, p_1, q_1), (2, p_2, q_2), \ldots, (n, p_n, q_n)\}$. Each triplet illustrates an assignment. Since AP3 is computational intractable but with wide applications, various algorithms have been proposed to tackle the problem, including Lagrangian relaxation [21], two-phase approximate algorithm [22], sub-gradient optimization [23], hybrid genetic algorithm [24], approximate backtrack [25], beam search [26], etc. Among the existing approaches, the approximate backtrack and the beam search are tree based search procedures, which may not be scalable to large-scale problem instances. In contrast, the local search based algorithms have the potential to solve large-scale instances. In particular, in the hybrid genetic algorithm [24], an efficient projection based local search is proposed, which is able to achieve promising performance.

As a brief summary, in this section, we introduce the related work of this study, including the hyper-heuristics, and the two problem domains, i.e., the GAP and the AP3. Unlike the existing hyper-heuristics, the algorithm in this study features a novel type of LLHs, i.e., the MP based LLH. Meanwhile, for both the GAP and the AP3 problem domains, hyper-heuristics have not been applied. Furthermore, this study is unique in that the scale of the problem instances for the two problem domains is much larger than the existing studies.

3 Decomposition-Mathematical Programming Based Hyper-heuristic

The **DEMPH** framework follows the general paradigm of hyper-heuristics. As shown in Fig. 1, the algorithm comprises two layers, i.e., the LLHs and the HLS. The two layers communicate with each other with predefined interface. On the one hand, the LLHs implement the domain-specific heuristics that operate on the problem instances. On the other hand, the HLS could be considered a scheduling module, which selects the most suitable LLHs to tackle the problem instances. In particular, in this study, **DEMPH** features the leveraging of MP solver as LLHs, which is able to solve sub-problems to optimality. Also, besides the MP based LLHs, **DEMPH** is also equipped with LLHs that realize the decomposition operations over the origin problem instance, which generate the sub-problems that are fed to the MP solver.

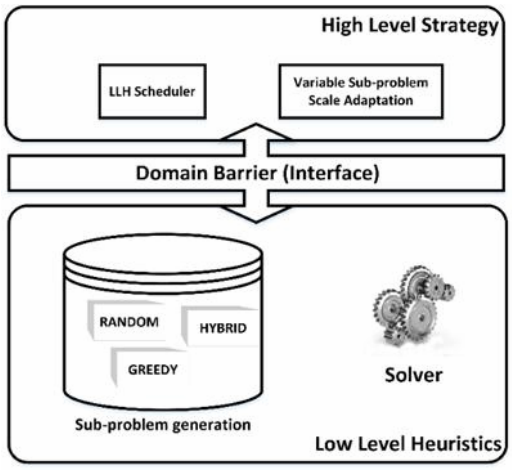

Fig. 1. The **DEMPH** framework

Similar as most hyper-heuristics, the algorithm iteratively improves the initial solutions. During each iteration of optimization, **DEMPH** follows the steps illustrated in Fig. 2, which is inspired by the fix-and-dive concept [13]. (1) Given a solution to the original problem instance, a subset of the decision variables is fixed and remains unchanged. (2) Then, the unfixed decision variables are employed to construct a

sub-problem instance (see Section 3.2 for detailed discussion). (3) After that, the sub-problem is fed to the MP based LLH, to obtain the optimal assignments for these unfixed variables. (4) Subsequently, the solution to the original problem instance is obtained by combining the fixed variable values and the ones returned by the MP based LLH. Such iteration continues, until the maximum iteration is reached. Upon termination, the best solution achieved is returned, as the final solution obtained by **DEMPH**. Among the search procedure, there are four major components that deserve more discussion, which are presented in the subsequent subsections.

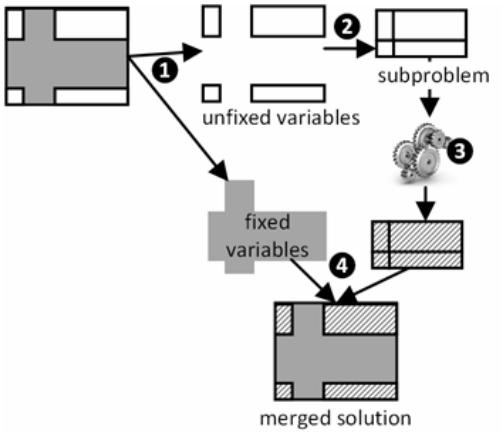

Fig. 2. Illustration of decomposition mechanism

3.1 Mathematical Programming Solver

In the literature, there are many MP solvers that have been successfully applied for problem solving, such as CPLEX [27], Gurobi [28], SCIP [29], etc. In this study, the mathematical programming solver we adopt in this study is Gurobi, which is among the state-of-the-art MP solvers. Due to the efficiency, Gurobi has been widely used in both industry and academic research, to tackle various optimization problems. Also, the support of modeling languages such as AMPL (A Mathematical Programming Language [30]) enables the declarative problem solving paradigm, and generalizes to other problem domains easily. However, as mentioned in Section 1, since GAP and AP3 are NP-hard problems, when the scale gets large, the search space scale grows exponentially accordingly. Consequently, Gurobi is not able to solve large scale GAP/AP3 instances within acceptable time. As a solution, in this algorithm, we employ Gurobi to solve the sub-problem of an instance, which is small enough to be handled efficiently.

3.2 Sub-problem Generation Low-level Heuristics

When generating a sub-problem based on a solution to a problem instance, a subset of the decision variables is obtained first. After that, the sub-problem of this instance is generated according to the subset. In this subsection, we illustrate how to generate a sub-problem based on a subset of the decision variables, for both GAP and AP3.

As mentioned in Section 2.2, a GAP instance consists of an agent set I of size n, a job set J of size m, as well as two auxiliary structures r and b. To generate a sub-problem with n agents and k jobs, a subset $J'=\{s_1, s_2,\ldots s_k\}$ is firstly selected from J. Meanwhile, all the agents remain the same as the original instance. Subsequently, for the resource structure r, only the columns with respect to J' are remained. Finally, for each agent i, the resource values of unselected jobs r_{ij} are subtracted from its capacity b_i.

Similarly, given an AP3 instance consisting of I, J, K of size n and a corresponding cost structure c, firstly subsets I', J', K' of size k are selected. Then, the cost structure c' is derived from c, according to the selected I', J', and K'.

For both the GAP and the AP3 problem, given a feasible solution to the origin instance, it is easy to fix a subset of variable values, and construct a sub-problem with respect to the unfixed variables. Then, solving the sub-problem to optimality could be considered as a special case of local search, which treats the search space of the sub-problem as the neighborhood. Consequently, merging the variable values of the optimal solution to the sub-problem with the fixed variable values could generate an improved

solution to the original instance.

We should note that, the selection strategy of the sub-problem construction plays an important role in the solving process. In this study, we model the sub-problem generation strategies as LLHs, i.e., during the search process, invoking to the LLHs will return sub-problems with respect to the strategies of the LLHs. The induced sub-problems will further be solved by Mathematical solvers. More specifically, we consider three types of LLHs, to generate sub-problems of size s:

Random. In this LLH, k sets of variables are randomly selected, and the sub-problem is constructed accordingly.

Greedy. In this LLH, the top-k sets of variables that contribute the most to the objective value are selected to construct the sub-problem.

Hybrid. In the hybrid approach, the top-(k/2) sets variables are selected with respect to **Greedy**, and the other k/2 sets of variables are randomly selected.

3.3 Variable Sub-problem Scale Adaptation

For all the three sub-problem generation LLHs, an important issue is that, there exists a parameter k, to control the scale of the induced sub-problem. During the search procedure, the parameter may have a great impact on the performance. On the one hand, if the parameter is set with a small value, the corresponding sub-problem might be too small, so that the search gets trapped in local optima with unsatisfying quality. On the other hand, if the parameter is set with large values, it may take too long for the employed mathematical solver to return the solution to the sub-problem, so that the accomplished iteration for the optimization is limited. To mitigate this challenge, we propose a Variable Sub-problem Scale Adaptation (VSSA) mechanism.

The idea is simple and straightforward, which is inspired by the Variable Neighborhood Search (VNS) heuristics [31]. In the VSSA mechanism, the search starts from a small sub-problem scale value. After solving the sub-problem with mathematical solver, and the repairing stage, we evaluate the obtained solution against the current best solution. If no better solution could be achieved, the sub-problem scale parameter is increased, so that the sub-problem generated for the subsequent iteration will be of larger scale. In contrast, if the achieved solution is better than the current best solution, the sub-problem scale parameter is reduced, to search more intensively around the best solution newly achieved.

3.4 High Level Strategy

The role of the LLH scheduling strategy in HLS is to choose the most suitable sub-problem selection strategy in this iteration. As mentioned, there are three basic strategies, i.e., the random strategy, the greedy strategy, and the hybrid strategy. Hence, using single selection strategy may not achieve robust results over different problem domains, or even different types of instances for single problem domain. As a solution, in this study, an adaptive selection strategy is proposed as follows. More specifically, we employ the improvement in terms of credit value to evaluate each sub-problem selection strategy. At the beginning, the credit value of each sub-problem selection strategy is assigned with a sufficient large positive value. Then at each iteration, the sub-problem selection strategy is selected proportional to the improvement achieved by the most recent LLH invoke. After the LLH is selected, the sub-problem is generated accordingly. MP solver is applied to solve the sub-problem, and the solution to the sub-problem is transformed back to the solution to the original problem. At the end of each iteration, the improvement of the solution objective value is recorded as the credit value of this sub-problem selection strategy. If two strategies have the same credit values, they will be selected randomly. With such adaptation, the proposed framework may automatically select the decomposition strategy according to the solution to be solved. The search procedure continues, until the stopping criterion is met.

As a brief summary, the **DEMPH** algorithm consists of two phases, i.e., the initialization phase and the main loop, which are described in Algorithm 1. During the initialization phase (lines 1-5), the incumbent solution and the best solution are initialized. Also, the auxiliary structures are also initialize, i.e., the credit values for the LLHs and the scale of the sub-problem (k). After that, at each iteration of the main loop (lines 6-17), the following steps are conducted. First, the LLH to generate the sub-problem is selected, according to the LLH scheduling module. Then, a sub-problem is generated with the **generateSubproblem** procedure accordingly. After that, the sub-problem is solved to optimality by the MP solver, and the obtained solution is used to update the incumbent solution. If the obtained solution is

better than the currently best known solution, the currently best solution is updated. Moreover, the k value is reset, so that the search is focused on the neighborhood of the newly obtained solution. Otherwise, if the best solution is not improved, k is increased, making the search procedure exploring larger area of the search space. The iteration continues, until certain stopping criterion is met.

Algorithm 1. DEMPH

Input: Problem instance π
Output: Best solution achieved by **DEMPH**

```
1.  //Initialization
2.  s ← randomSolution(π
3.  best ←s
4.  initializeLLHCredit()
5.  k ← resetScaleVSSA()
6.  //Main loop
7.  while stopping criterion not met
8.        LLH←selectLLH()
9.        subproblem←generateSubproblem(LLH, k, s, π)
10.       s' ←MPSolve(subproblem)
11.       s'' ←mergeSolution(s, s')
12.       updateLLHCredit(s'')
13.       if s'' better than best
14.          best ←s''
15.          k ← resetScaleVSSA()
16.       else
17.          k ← k× 2
18. return best
```

4 Experimental Results

In this section, we evaluate the performance of **DEMPH**, against its variants and other comparative algorithms, on both GAP and AP3 instances. In particular, we are interested in the following two Research Questions (RQs)

(1) **RQ1**: how does **DEMPH** performs, compared with the state-of-the-art algorithm and variants of **DEMPH**?

(2) **RQ2**: is **DEMPH** able to adaptively select the most suitable LLHs, to generalize to different problem instances?

To investigate the RQs, we consider the following three categories of algorithms for comparison. First, we consider the approaches in the existing literatures, for which we are able to have access to the source code. More specifically, for the GAP, the parallel genetic algorithm (**PGAP)** [19] is considered, and for AP3, the hybrid genetic algorithm (**HGA**) is considered. Second, the variants of **DEMPH** in which only single decomposition LLH are considered, which are indicated as **Random**, **Greedy**, and **Hybrid**. In each algorithm in this category, because there is only single decomposition LLH, the algorithm degrades into an iterated local search like approach. Also, in these three comparative approaches, the VSSA mechanism is not considered. Finally, for the third category, we consider **DEMPH** and its variant with static sub-problem scale parameter (indicated as **DEMPH-S**). With these variants, the mechanisms in **DEMPH** could be investigated. For example, by comparing **DEMPH-S** with **Random**, **Greedy**, and **Hybrid**, we intend to examine the usefulness of integrating multiple LLHs. Meanwhile, the effectiveness of VSSA could be demonstrated by comparing **DEMPH** and **DEMPH-S**. All of the algorithms are implemented in C++, and run under Windows on a PC with an Intel Core i5 3.2 GHz CPU and 4GB memory. Besides, the data set is available at http://oscar-lab.org/DEMPH.

4.1 Results on GAP

The instances of this experiment are generated with respect to the rules following the widely used OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html), and the rules from the Yagiura library (http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/gap/). The original instances are not considered, because these existing instances are relatively small, so that Gurobi is able to generate optimal solutions efficiently. There are 5 types of data sets according to the generation methods, i.e., Type A-E. For each type we generate a set of large scale instances, each of which has a size of 500×40000. For each type, 3 instances are generated.

Parameter tuning. For the each sub-problem generation LLH, there is a parameter to control the scale of the generated sub-problem. To configure the values for the LLHs for the comparative algorithms, we employ an open-source tool **irace** [32] (http://iridia.ulb.ac.be/irace/) to conduct the parameter tuning task. More specifically, the instances used are one instance from each type, 5 instances in all. The candidates for the sub-problem size are from {10, 20, 50, 100, 200, 500, 1000}. The running time is limited to 3 minutes. The parameter tuning result for each strategy is listed as follows. For the random strategy, the sub-problem size is set to be 20. For the greedy strategy, the sub-problem size is 10. Finally, for **DEMPH**, the sub-problem size is 20, the initial value of the sub-problem scale is assigned with 10. We should note that, with the VSSA mechanism we do not intend to propose a parameter-free algorithm. Instead, as demonstrated in related parameter control studies, the goal of parameter control is to alleviate the sensitivity to the parameters. Besides, for **PGAP**, the parameters are set with respect to the literature [19].

Investigation of RQ1. For each instance, each algorithm is executed for 10 independent runs, and the cut off time is set with 10 minutes. The experimental results of GAP are shown in Table 1. The table is organized as follows. The first column indicates the **id** of the instances. The second column presents the results obtained by **PGAP**. Then, for each algorithm, i.e., **Random**, **Greedy**, **Hybrid**, **DEMPH-S**, and **DEMPH**, we report the best solution achieved, and the average gap, which illustrates the gap between the average value found by each strategy and the best value found by all comparative algorithms:

$$average\_gap = \frac{average - best\ value}{best\ value}. \tag{10}$$

Table 1. The results of **DEMPH** over GAP instances

id	PGAP	Random		Greedy		Hybrid		DEMPH-S		DEMPH	
		Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap
A1	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
A2	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
A3	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
B1	**400001**	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001
B2	**400001**	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001
B3	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
C1	**400001**	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001	**400001**	<0.0001
C2	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
C3	**400000**	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001	**400000**	<0.0001
D1	2489241	2475527	0.0019	2533130	0.0252	2472671	0.0005	2477876	0.0027	**2471701**	0.0002
D2	2490224	2475102	0.0017	2532914	0.0253	2472963	0.0006	2478043	0.0028	**2471631**	<0.0001
D3	2488839	2474859	0.0017	2531736	0.0252	2472569	0.0007	2477284	0.0029	**2471218**	<0.0001
E1	9987232	**5010011**	0.0073	7727591	0.5429	5137684	0.0265	5048773	0.0080	5044976	0.0095
E2	9998769	5057727	0.0046	7732462	0.5642	5163101	0.0332	5057407	0.0063	**5039279**	0.0069
E3	10015158	5046403	0.0076	7942691	0.5913	5164273	0.0375	5014875	0.0148	**5044645**	0.014

From Table 1, we can the following interesting phenomena. First, the instances of Types A, B, C are relatively easy to solve, the minimum values of these types are 400000 in theory according to their generation rule. Second, over the Type D and Type E instances, we observe that **DEMPH** is able to obtain promising solutions, which outperforms the other comparative approaches, except for the instance E1. Interestingly, over Type E instances, the performance of **PGAP** is not satisfying. A possible reason is that, the experimental environment in this study is not a many-CPU server, so that **PGAP** could not leverage the benefit of the parallel mechanism. Third, when comparing **DEMPH-S** with **Random**,

Greedy, and **Hybrid**, we find that there is no single decomposition LLH that suits all the instances. For example, over the Type D instances, **Random** performs the best among the three variants (**Greedy**, and **Hybrid**). Meanwhile, **Hybrid** outperforms **Random** over the Type E instances. This observation to some extent confirms the necessity of combining multiple LLHs.

Answer to RQ1. From the numerical results, we observe that the performance of **DEMPH** is able to outperform the state-of-the-art algorithm over the large scale AP3 instances.

Investigation of RQ2. In this RQ, we are interested in whether **DEMPH** is able to select the most suitable LLH during the search procedure. To gain more insights into the dynamic perspectives of the proposed algorithm, we examine the anytime performance of **DEMPH** and its comparative approaches, i.e., the trend of the solution quality achieved by each algorithm against the elapsed time. In Fig. 3 to Fig. 4, we visually plot the solution quality of the algorithms in comparison against the run time. More specifically, Fig. 3 shows the anytime performance on a typical Type D instance, while Fig. 4 shows the value of solution on a Type E instance. From the figures, interesting phenomena could be observed. First, neither the random strategy nor the greedy strategy generalizes well when faced with different types of instances. For example, over the typical type D instance, **Hybrid** outperforms the other two variants (**Greedy** and **Random)**. Meanwhile, over the Type E instance, **Random** is able to obtain better results than **Greedy** and **Hybrid**. Interestingly, the runtime behavior of **DEMPH-S** is similar with the best variant in both cases. This observation to some extents demonstrates the generalization ability of the adaptive HLS, i.e., **DEMPH-S** is able to adaptively select the most appropriate strategy. Moreover, when equipped with the VSSA mechanism, the performance could be further improved. Over both instances, **DEMPH** outperforms all the other comparative approaches.

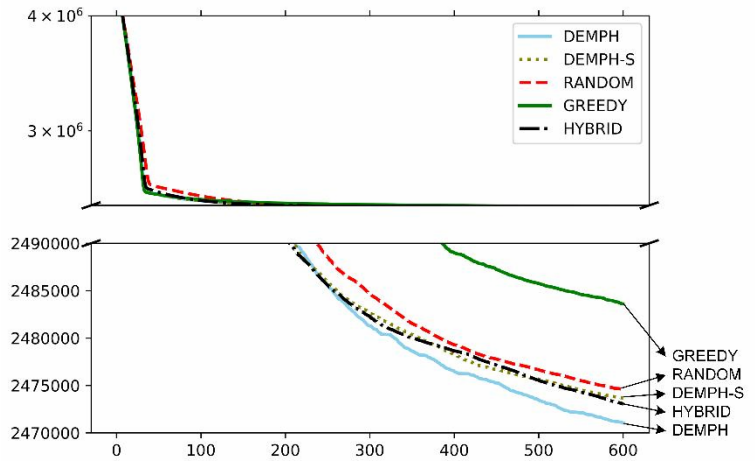

Fig. 3. Anytime performance comparison over GAP Type D instance

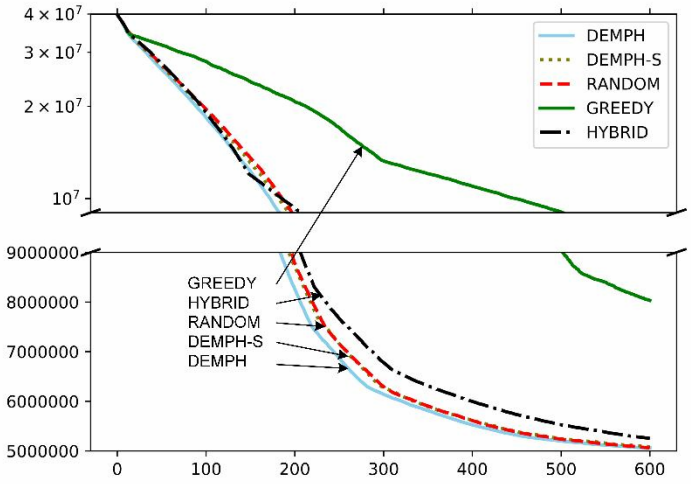

Fig. 4. Anytime performance comparison over GAP Type E instance

Answer to RQ2. With the analysis over typical AP3 instances, we confirm that **DEMPH-S** is able to select the most suitable LLHs. Moreover, with the VSSA mechanism, **DEMPH** is able to further improve the solution quality.

4.2 Results on AP3

The instances of AP3 are generated according to two widely used data sets, i.e., the Balas & Saltzman data set [21] and the Crama & Spieksma data set [22]. The instance size of these two datasets is relatively small, and Gurobi can solve them efficiently. Hence, we generate the new data sets with respect to the generation rule of the two existing data sets. Another issue worth noting is that, the cost of Balas & Saltzman data set is generated randomly from [0, 100]. As stated by Huang et al. [24], when the scale increases, the value of solution gets close to 0, and these instances tend to be easy to solve. Thus, when generating the instance following Balas & Saltzmans's rule, except that we change the cost into [0, 1000] and [0, 10000]. Meanwhile, for the Crama & Spieksma data set, there are 3 types of instance according to the generation ways. For each type, the sizes of instance are 100, 200, and 300. For each type and size, 3 instances are generated. Consequently, there are in total 45 instances.

Parameter Tuning. Similar as in the previous experiment, we use **irace** to conduct the parameter tuning. To tune the sub-problem scale parameters, for each type of instance, we randomly select an instance, which results in 5 instances. The candidate values for the sub-problem size are from {10, 20, 30}. Since the execution might be slow, we set the cut off time to be 1 minute. The parameter tuning result for each strategy is listed as follows. For the random strategy, the sub-problem scale is 30. For the greedy strategy, the sub-problem scale is 30. For **DEMPH-S**, the sub-problem size is 30. Besides, the parameters of **HGA** are set according to the literature [24].

Investigation of RQ1. Similar with the previous experiment setup, each comparative algorithm runs for 10 independent executions over all the instances, and the cut off time is set with 10 minutes. The results over the Balas & Saltzman like data set and the Crama & Spieksma like data set are presented in Table 2 and Table 3, respectively. The column Best shows the best solution found by each algorithm in all runs. The column Avg. gap shows the average value of the 10 runs. Unlike the experimental results for GAP, the solution quality achieved by the comparative algorithms is well distinguishable over most instances.

Table 2. The results of **DEMPH** over Balas & Saltzman like AP3 data set

Cost range	Size	id	**HGA**	**Random**		**Greedy**		**Hybrid**		**DEMPH-S**		**DEMPH**	
				Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap
1000	100	1	265	67	0.8452	54	0.2500	60	0.4405	48	0.1667	**42**	0.0595
1000	100	2	295	53	0.7742	41	0.4516	46	0.5161	50	0.6290	**31**	0.2903
1000	100	3	272	61	1.2097	49	0.6290	**31**	0.3226	47	0.5484	39	0.3710
1000	200	1	290	51	1.0366	51	0.6951	79	1.1707	53	0.8171	**41**	0.2927
1000	200	2	285	70	1.0429	52	0.2429	51	0.9429	**40**	0.2286	52	0.5571
1000	200	3	305	202	1.7538	317	4.1692	175	1.7692	208	2.3692	**65**	0.0385
1000	300	1	274	78	0.5500	96	1.0583	**60**	0.0167	70	0.6667	61	0.3167
1000	300	2	272	82	1.5510	66	0.3980	65	0.4490	**49**	0.0510	96	1.2245
1000	300	3	270	74	0.5948	74	1.1121	58	0.1552	**58**	0.6552	80	0.5776
10000	100	1	3307	1085	0.3993	846	0.0482	866	0.1687	**809**	0.0766	888	0.1001
10000	100	2	3311	1072	0.3429	865	0.1234	820	0.1047	839	0.0854	**802**	0.0025
10000	100	3	3309	1000	0.4090	**725**	0.1276	931	0.3372	775	0.1566	831	0.1959
10000	200	1	3998	1655	0.3132	1325	0.0065	1419	0.1177	1399	0.0718	**1317**	0.1090
10000	200	2	3740	1533	0.3385	1385	0.1415	1429	0.1890	**1254**	0.0311	1378	0.1236
10000	200	3	3612	1573	0.3380	1345	0.2132	1392	0.0630	1470	0.1248	**1318**	0.1017
10000	300	1	4062	2226	0.2141	2026	0.0101	2102	0.1024	**2022**	0.1489	2220	0.1555
10000	300	2	4127	2197	0.1752	**1963**	0.1174	1994	0.0670	2024	0.1006	2249	0.1577
10000	300	3	3797	2174	0.2785	1964	0.0076	2113	0.0998	2006	0.0481	2174	0.1197

From Table 2, we can observe that over Balas & Saltzman instances, **DEMPH** is able to achieve promising performance. When we consider the best solutions obtained by the three comparative algorithms, **DEMPH** outperforms the other variants over 7 instances. Meanwhile, when we consider the average solution quality as the performance measurement, GREEDY performs the best, while the

average solution quality of **DEMPH-S** is quite close to GREEDY. Besides, we observe that the comparative algorithm **HGA** does not perform well over these category of instances. A possible reason might be that, the values in the instances are randomly selected from a large range. Consequently, the local search based search procedure may not be effective enough. In contrast, with the MP solver, **DEMPH** and its variants could obtain much better results. Similarly, in Table 3, we could observe that for the best solution found, **DEMPH** performs the best over 11 instances. Also, for **Random**, **Greedy**, and **Hybrid**, there is no single variant that outperforms the other two approaches significantly.

Table 3. The he results of **DEMPH** over Crama & Spieksma like AP3 data set

Type	Size	id	**HGA**	**Random**		**Greedy**		**Hybrid**		**DEMPH-S**tatic		**DEMPH**	
				Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap	Best	Avg. gap
I	100	1	**3268**	**3268**	0.0187	3272	0.0285	**3268**	0.0116	**3268**	0.0132	**3268**	0.0003
I	100	2	**3724**	**3724**	0.0259	3736	0.0303	**3724**	0.0169	**3724**	0.0231	**3724**	0.0000
I	100	3	**3178**	**3178**	0.0181	3183	0.0286	**3178**	0.0135	**3178**	0.0175	**3178**	0.0003
I	200	1	**4303**	4624	0.1416	4813	0.1442	4500	0.0910	4500	0.1205	4778	0.1120
I	200	2	**4751**	5024	0.1340	5262	0.1441	4978	0.0908	5045	0.1168	5354	0.1285
I	200	3	**4980**	5242	0.1127	5455	0.1286	5111	0.0757	5141	0.0970	5513	0.1113
I	300	1	**5354**	6510	0.2736	6675	0.2769	6081	0.1844	6335	0.2425	6826	0.2850
I	300	2	**5369**	6394	0.2486	6571	0.2383	6134	0.1711	6319	0.2285	6598	0.2413
I	300	3	**4773**	5834	0.2834	5957	0.2777	5474	0.2127	5861	0.2772	6079	0.2761
II	100	1	16136	**16134**	0.0016	16219	0.0224	**16134**	0.0017	16135	0.0017	**16134**	0.0001
II	100	2	**15834**	**15834**	0.0023	15954	0.0260	**15834**	0.0020	15834	0.0021	**15834**	0.0001
II	100	3	**17038**	**17038**	0.0013	17328	0.0263	**17038**	0.0014	17038	0.0019	17039	0.0001
II	200	1	**32965**	33189	0.0153	34490	0.0572	33122	0.0104	33168	0.0159	33275	0.0104
II	200	2	**33058**	33438	0.0191	35041	0.0662	33213	0.0095	33345	0.0173	33364	0.0099
II	200	3	**32305**	32438	0.0126	33790	0.0546	32403	0.0090	32524	0.0138	32579	0.0090
II	300	1	**48707**	50550	0.0447	52199	0.0776	49577	0.0287	50268	0.0420	49959	0.0272
II	300	2	**50897**	52390	0.0355	54848	0.0807	51491	0.0209	52334	0.0370	52067	0.0243
II	300	3	55011	50366	0.0214	52959	0.0696	**49633**	0.0096	50180	0.0214	50206	0.0132
III	100	1	409	**408**	0.0098	**408**	0.0123	**408**	0.0061	**408**	0.0074	**408**	0.0000
III	100	2	**407**	**407**	0.0086	**407**	0.0111	**407**	0.0086	**407**	0.0098	**407**	0.0000
III	100	3	**404**	**404**	0.0062	**404**	0.0087	**404**	0.0025	**404**	0.0074	**404**	0.0000
III	200	1	**771**	773	0.0214	795	0.0350	786	0.0266	786	0.0298	787	0.0208
III	200	2	**765**	779	0.0340	790	0.0405	779	0.0301	781	0.0333	782	0.0235
III	200	3	750	774	0.0487	788	0.0547	774	0.0447	770	0.0440	778	0.0387
III	300	1	1770	1173	0.0090	1194	0.0183	1182	0.0098	1181	0.0128	**1180**	0.0081
III	300	2	1770	1176	0.0102	1187	0.0124	1173	0.0047	1182	0.0107	**1173**	0.0026
III	300	3	1767	1169	0.0757	1182	0.0807	1162	0.0693	1165	0.0720	**1097**	0.0342

Answer to RQ1. From the numerical results, we observe that over the AP3 instances, the performance of **DEMPH** is comparable to the state-of-the-art algorithm, and could achieve the best results of 18 instances.

Investigation of RQ2. Similar with the investigation to RQ2 for GAP, in this experiment, we are interested in the runtime behavior of **DEMPH** and its variants. Fig. 5 depicts the runtime behavior of the comparative algorithms along the search process on a typical Balas & Saltzman like data set instance, while Fig. 6 is associated with a typical Crama & Spieksma like data set instance. From the two figures, the following phenomena could be observed. First, similar with the investigation of RQ2 for GAP, we confirm that the behavior of **DEMPH-S** is able to adaptively select the most suitable LLH. For example, in Fig. 5, the anytime performance curve of **DEMPH-S** is very close to that of **Hybrid**, which performs better than **Greedy** and **Random**. Second, we observe that on different instances, all the comparative approaches other than **DEMPH** suffer from the pre-mature issue. In contrast, **DEMPH** is able to improve solutions after 400 iterations, and obtain the best solutions over both instances.

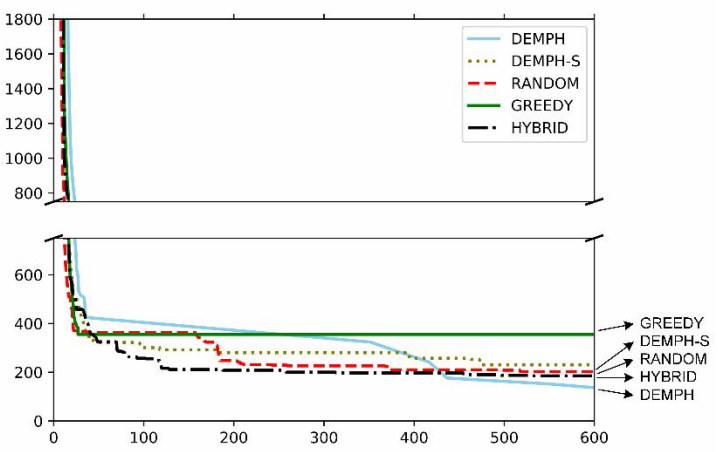

Fig. 5. Anytime performance comparison over Balas & Saltzman like AP3 instance

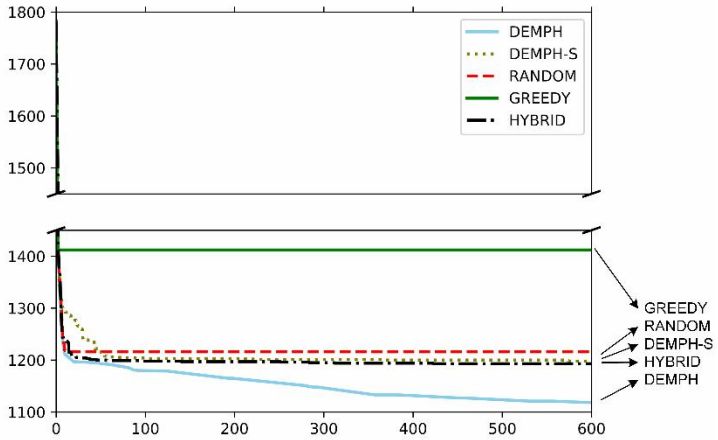

Fig. 6. Anytime performance comparison over Crama & Spieksma like AP3 instance

Answer to RQ2. In this RQ, we confirm the generalization ability of **DEMPH** by analyzing the runtime behavior of **DEMPH** and its variants. With the HLS, **DEMPH-S** performs similar with the best variant that considers single sub-problem generation LLH. Furthermore, **DEMPH** is able to outperforms **DEMPH-S**, which demonstrates the effectiveness of the VSSA mechanism.

5 Conclusion

In this paper, we propose an algorithm named **DEMPH** to solve large-scale problems. The contributions of the proposed algorithm are twofold. On the one hand, by using modern mathematical programming solver within a hyper-heuristic, the low level heuristics can be implemented much more easily. The **DEMPH** is easy to be implemented and generalized to new problem domains. On the other hand, with the decomposition based high level strategy, **DEMPH** is able to be applied to tackle very large scale problem instances, which are not possible for mathematical programming solvers. The experiment results demonstrate that with the high level strategy, **DEMPH** is able to solve problem very-large-scale instances, and exhibits a high adaptability when faced with different types of instances. The algorithm proposed in this paper is not only an effective method for solving GAP and AP3, it is also a promising framework. In the future work, we are interested in the following directions. First, in this study, both GAP and AP3 are modeled as integer programming problems, and the sub-problems are solved with the state-of-the-art mathematical programming solver. In many real-world problem domains, there may exist complex, non-linear constraints and objective functions. Hence, an interesting direction would be investigating the possibility of incorporating constraint solvers as LLHs. Also, in this study, the

optimization is achieved by mathematical programming solver. In the future, we will attempt to combine both the exact solvers and heuristic approaches such as local search.

References

[1] G. Kizilates, F. Nuriyeva, A parametric hybrid method for the traveling salesman problem, Mathematical and Computational Apllications 18(3)(2013) 459-466.

[2] A. Fukunaga, Automated discovery of local search heuristics for satisfiability testing, Evolutionary Computation 16(1)(2008) 31-61.

[3] G. Kızılates, F. Nuriyeva, A parametric hybrid method for the traveling salesman problem, Mathematical and Computational Applications 18(3)(2013) 459-466.

[4] Z. Ren, H. Jiang, J. Xuan, Z. Luo, Hyper-heuristics with low level parameter adaptation, Evolutionary Computation 20(2)(2012) 189-227.

[5] H. Ingimundardottir, T. Runarsson, Discovering dispatching rules from data using imitation learning: a case study for the job-shop problem, Journal of Scheduling 21(4)(2017) 1-16.

[6] R. Aron, I. Chana, A. Abraham, A hyper-heuristic approach for resource provisioning-based scheduling in grid environment, The Journal of Supercomputing 71(4)(2015) 1427-1450.

[7] S. Altner, K. Ahuja, O. Ergun, B. Orlin, Very large-scale neighborhood search. DOI: 10.1007/978-1-4614-6940-7_13.

[8] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. Woodward, A classification of hyper-heuristic approaches, in: M. Gendreau, J.Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, , Berlin, Heidelberg, 2010, pp. 449-468.

[9] N. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems, IEEE Transactions on Evolutionary Computation 19(3)(2015) 309-325.

[10] K. Sim, E. Hart, B. Paechter, A lifelong learning hyper-heuristic method for bin packing, Evolutionary Computation, 23(1)(2015) 37-67.

[11] J. Soria-Alcaraz, A. Espinal, M. Sotelo-Figueroa, Evolvability metric estimation by a parallel perceptron for on-line selection hyper-heuristics, IEEE Access 5(2017) 7055-7063.

[12] N. Sabar, X. Zhang, A. Song, A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows, in: Proc. 2015 IEEE Congress on Evolutionary Computation (CEC), 2015.

[13] D. Salvagnin, Detecting semantic groups in MIP models, in: Proc. International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2016.

[14] G. Ross, R. Soland, A branch and bound algorithm for the generalized assignment problem, Mathematical Programming, 8(1975) 91-103.

[15] P. Chu, J. Beasley, A genetic algorithm for the generalised assignment problem, Computers & Operations Research, 24(1997) 17-23.

[16] J. Diaz, E. Fernández, A tabu search heuristic for the generalized assignment problem, European Journal of Operational Research 132(2001) 22-38.

[17] S. Kar, K. Basu, S. Mukherjee, Solution of a class of generalized assignment problem, Journal of Intelligent & Fuzzy Systems 33(3)(2017) 1687-1697.

[18] A. Ghoniem, T. Flamand, M. Haouari, M. Exact solution methods for a generalized assignment problem with location/allocation considerations, INFORMS Journal on Computing 28(3)(2016) 589-602.

[19] Y. Liu, S. Wang, A scalable parallel genetic algorithm for the generalized assignment problem, Parallel Computing 46(2015) 98-119.

[20] W. Pierskalla, The tri-substitution method for the three-dimensional assignment problem, CORS Journal 5(1967) 71-81.

[21] E. Balas, M. Saltzman, An algorithm for the three-index assignment problem, Operations Research 39(1991) 150-161.

[22] Y. Crama, F. Spieksma, Approximation algorithms for three-dimensional assignment problems with triangle inequalities, European Journal of Operational Research 60(1992) 273-279.

[23] S. Maneechai, Variant of constants in subgradient optimization method over planar 3-index assignment problems, Mathematical and Computational Applications 21(1)(2016) 4.

[24] G. Huang, A. Lim, A hybrid genetic algorithm for the three-index assignment problem, European Journal of Operational Research 172(2006) 249-257.

[25] H. Jiang, J. Xuan, X. Zhang, An approximate muscle guided global optimization algorithm for the three-index assignment problem, in: Proc. IEEE Congress on Evolutionary Computation, 2008.

[26] H. Jiang, S. Zhang, Z. Ren, X. Lai, Y. Piao, Approximate muscle guided beam search for three-index assignment problem, in: Y. Tan, Y. Shi, C.A.C. Coello (Eds.), Advances in Swarm Intelligence, Springer, Cham, 2014, pp. 44-52.

[27] The IBM CPLEX Optimizer. <https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>.

[28] The Gurobi Solver. <http://www.gurobi.com/>.

[29] T. Achterberg, SCIP: solving constraint integer programs, Mathematical Programming Computation 1(1)(2009) 1-41.

[30] R. Fourer, D. Gay, B. Kernighan, AMPL: A Mathematical Programming Language, AT&T Bell Laboratories, Murray Hill, NJ, 1987.

[31] P. Hansen, N. Mladenovic, J. Brimberg, J.A.M. Pérez, Variable neighborhood search, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer International Publishing, 2016 (Chapter 3).

[32] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, M. Birattari, T. Stützle, The irace package, iterated race for automatic algorithm configuration, Operations Research Perspectives 3(26)(2011) 43-58.