Hyper-Heuristics with Low Level Parameter Adaptation

Zhilei Ren ren@mail.dlut.edu.cn
School of Mathematical Sciences, Dalian University of Technology, Dalian, 116621, China

He Jiang* jianghe@dlut.edu.cn
School of Software, Dalian University of Technology, Dalian, 116621, China

Jifeng Xuan xuan@mail.dlut.edu.cn
School of Software, Dalian University of Technology, Dalian, 116621, China

Zhongxuan Luo zxluo@dlut.edu.cn
School of Mathematical Sciences, Dalian University of Technology, Dalian, 116621, China

Abstract

Recent years have witnessed the great success of hyper-heuristics applying to numerous real-world applications. Hyper-heuristics raise the generality of search methodologies by manipulating a set of low level heuristics (LLHs) to solve problems, and aim to automate the algorithm design process. However, those LLHs are usually parameterized, which may contradict the domain independent motivation of hyper-heuristics. In this paper, we show how to automatically maintain low level parameters (LLPs) using a hyper-heuristic with LLP adaptation (AD-HH), and exemplify the feasibility of AD-HH by adaptively maintaining the LLPs for two hyper-heuristic models. Furthermore, aiming at tackling the search space expansion due to the LLP adaptation, we apply a heuristic space reduction (SAR) mechanism to improve the AD-HH framework. The integration of the LLP adaptation and the SAR mechanism is able to explore the heuristic space more effectively and efficiently. To evaluate the performance of the proposed algorithms, we choose the p-median problem as a case study. The empirical results show that with the adaptation of the LLPs and the SAR mechanism, the proposed algorithms are able to achieve competitive results over the three heterogeneous classes of benchmark instances.

Keywords

Hyper-heuristics, parameter control, heuristic space reduction, intensification, diversification, ant colony optimization.

1 Introduction

Informally, hyper-heuristics are those approaches of "using heuristics to choose heuristics" (Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward, 2010). The main objectives of hyper-heuristics are to (1) improve the flexibility of heuristic algorithms (Ross, 2005), (2) obtain "good enough" results without causing much implementation burden ("soon

*Corresponding Author.

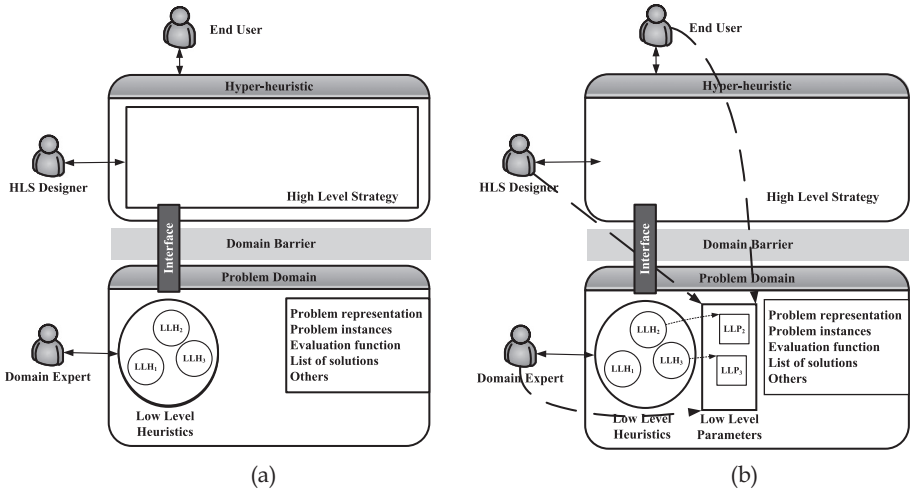


Figure 1: Hyper-heuristic hierarchy. (a) Hyper-heuristic. (b) Hyper-heuristic with static LLP configurations.

enough and cheap enough") (Burke, Kendall, Newall et al., 2003), and (3) automate the process of algorithm design (Burke, Hyde, Kendall, Ochoa, Özcan, and Qu, 2010). To achieve these goals, a hyper-heuristic is usually designed as a hierarchical framework. For example, Figure 1(a) illustrates the hierarchy of a typical hyper-heuristic. In the framework, the domain barrier (Cowling et al., 2001a) is introduced to separate the domain specific low level heuristics (LLHs) and the general high level strategies (HLSs). These two parts interact with each other through well defined interfaces. With this hierarchy, most of the hyper-heuristics' objectives can be achieved; for example with the domain barrier, a hyper-heuristic can be transplanted to different problem domains without modifying its HLS. The only requirement is to implement the LLHs of the specific domain. Besides, the domain barrier modularizes the hyper-heuristic framework, which involves less human interference.

As observed in Figure 1(a), in a hyper-heuristic framework, there are three main roles, including the HLS designer who proposes and implements the domain independent strategies, the domain expert who provides the domain specific LLHs and knowledge, and the end user. For generality and flexibility considerations, the HLS designer and the domain expert do not collaborate directly, but instead communicate in an indirect way. On one hand, the domain expert provides the information of the LLHs, such as the input, the output, and the functionality of each LLH. On the other hand, the HLS designer develops the HLS that explores the heuristic space derived by the LLHs, and manipulates the LLHs to conduct the search over the solution space. The indirect communication between the HLS designer and the domain expert is consistent with the motivation of the domain barrier. By separating these two roles, a hyper-heuristic would be easy to extend to new problem domains.

Since emergence, hyper-heuristics have been applied to various problem domains such as the bin packing problem (Cuesta-Cañada et al., 2005; Poli et al., 2007; Burke et al., 2010b), the job shop scheduling problem (Ho and Tay, 2005; Vázquez-Rodríguez and Petrovic, 2010), the timetabling problem (Ochoa et al., 2009; Pillay and Banzhaf, 2009;

Qu et al., 2009; Qu and Burke, 2009), and others. However, despite the great success of hyper-heuristics, there are still several difficulties in the process toward the automation of algorithm design. Among these difficulties, the parameterization of the LLHs poses great challenges to hyper-heuristics. The reason is that the LLHs are commonly parameterized. These domain specific low level parameters (LLPs) may lead to a series of problems. Figure 1(b) describes a scenario in which parameterized LLHs are employed in the hyper-heuristic framework, which illustrates several potential risks of incorporating LLPs. On one hand, if the LLPs are statically set by the domain experts, the performance of the framework may suffer from the generality issue, because static parameter configurations may not work well over different problem domains, or even different instances of the same domain (Serpell and Smith, 2010). Meanwhile, the tuning of these LLPs may also be time-consuming and error prone. On the other hand, leaving these LLP configurations to the HLS designer or the end user may not be appropriate as well, in that with these parameters, the HLS designer or the end user has to be aware of the details of the domain specific knowledge, which might break the domain barrier of the framework.

In order to alleviate this difficulty, we show how to adaptively maintain the LLPs with a search-based algorithm, and propose the hyper-heuristic framework with adaptive LLPs (AD-HH). In this framework, the HLS is decomposed into two modules, so as to manage the LLHs and the LLPs simultaneously. With the LLP adaptation, the proposed framework has the following unique features. First, in this approach, the HLS designer does not need to acquire much domain specific knowledge, in that the LLPs are optimized along the exploration of the search space. Second, with the LLPs adaptively maintained, the necessity of the time-consuming LLP tuning is eliminated, which is consistent with the "soon enough" objective of hyper-heuristics (Burke, Kendall, Newall et al., 2003). Besides, with the LLP adaptation, the interaction interface between the LLPs and the HLS is consistent with that between the LLHs and the HLS, thus the modularity of the hyper-heuristic framework can be preserved.

We do not claim, however, that we should eliminate all the parameters, or propose a parameter-free hyper-heuristic framework similar to Cowling et al. (2001b) or Kendall et al. (2002), among others. We note that AD-HH differs from these parameter-free approaches from both the LLH and the HLS perspectives. On one hand, these parameter-free approaches only employ parameter-free LLHs, while AD-HH is able to handle parameterized LLHs. On the other hand, in our study, the parameters in the HLS are retained in the AD-HH framework, which is based on the following reasons. First, the motivation of the framework is to reduce the intervention of the domain experts. The parameters remaining in the framework are maintained by the HLS designer. Thus, these parameters do not break the domain barrier, which means the framework can be easily generalized to other problem domains. Second, the HLS parameters may satisfy the diverse demand of the end users. For example, with a larger population of LLH sequences and/or a larger number of iterations, the algorithm may achieve better solutions, at the cost of more time elapsed. Instead, decreasing the values of these parameters may quickly lead to some solutions, yet the quality of the solutions may not be very competitive. Besides, the adaptation of parameters does not necessarily lead to algorithms with fewer parameters. Conversely, the introduction of the extra parameters are acceptable if their effect is positive (e.g., if the performance is less sensitive to these introduced parameters, as discussed in Eiben et al., 2007, or if these introduced parameters preserve the modularity of the algorithm, as in this study). For these reasons, we concentrate on the adaptation of the LLPs rather than the HLS parameters.

The LLP adaptation improves the modularity and the generality of hyper-heuristics. However, the scale of the search space increases accordingly, in that the LLPs are incorporated as optimization variables into the search space. As a solution, we propose the heuristic space reduction (SAR) mechanism, in order to improve the effectiveness and the efficiency of the search procedure. The SAR mechanism is based on the observation that there exists redundancy in existing LLH move acceptance criteria. In most existing hyper-heuristics, the LLHs are treated in an equivalent way, such that the LLHs of similar functionalities may be executed consecutively, which may lead to redundancy during the search process. The SAR mechanism explicitly partitions the LLHs into two subsets of LLHs that provide intensification and diversification functionalities. At each iteration of the search procedure, the SAR mechanism alternatively accepts the LLHs of different functionalities. In essence, the heuristic space is significantly reduced into a subspace by restricting LLHs to be selected from the Cartesian product of two subsets of LLHs. The motivation behind the reduction mechanism is inspired by the concept of metaheuristics, in which the process of optimization is interpreted as the combination of the intensification and the diversification strategies. With the LLP adaptation and the SAR mechanism, we can perform the search procedure effectively and efficiently but still retain the generality of hyper-heuristics.

To evaluate the performance of the proposed algorithms, we choose the p-median problem as a case study. Extensive experiments were carried out to test the robustness of our algorithms. For the benchmark set, we use three heterogeneous classes of instances, including 40 graph-based instances from ORLIB (Beasley, 1985), five random instances from RW (Resende and Werneck, 2003), and 10 Euclidean instances from TSPLIB (Reinelt, 1991). By comparing the proposed hyper-heuristics with the state of the art results, we demonstrate that the combination of the LLP adaptation and the SAR mechanism is able to achieve competitive results. Furthermore, through extensive statistical tests, we demonstrate that both the LLP adaptation and the SAR mechanism are effective and beneficial, while the combination of the two mechanisms contributes greatly to the competitive performance of the framework.

Our contributions can be summarized as follows. (1) To the best of our knowledge, this is the first study that considers the LLP adaptation with a search-based algorithm in the context of hyper-heuristics. (2) By incorporating two hyper-heuristic models into the AD-HH framework, we demonstrate the feasibility and the flexibility of the LLP adaptation mechanism. (3) In order to prevent the search space from drastic expansion, we propose the SAR mechanism, which is able to significantly reduce the scale of the heuristic space, meanwhile keeping the search effective and efficient. (4) The proposed framework is tested on the p-median problem, which is a novel domain for hyper-heuristics. Extensive experiments demonstrate that the combination of the LLP adaptation and the SAR mechanism is able to achieve promising results.

The paper is organized as follows. In Section 2 we introduce the related work of both hyper-heuristics and the parameter setting methodologies. In Section 3, we propose the AD-HH framework, in which the LLPs are adaptively maintained. In Section 4, we instantiate AD-HH by considering two hyper-heuristic models, that is, an ant-based hyper-heuristic and a genetic algorithm (GA) based hyper-heuristic, so as to demonstrate the flexibility of the framework. Since the LLP adaptation may expand the search space significantly, in Section 5, we propose the SAR mechanism, so as to restrict the search space from drastic growth. The empirical results and discussions are given in Section 6. Finally, the conclusion and the future work are presented in Section 7.

2 Related Work

In this section, we introduce the background of hyper-heuristics, as well as the existing parameter setting approaches.

2.1 Hyper-Heuristics

(Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward, 2010, p. 452) define hyper-heuristic as "an automated methodology for selecting or generating heuristics to solve hard computational search problems." In the same paper, hyper-heuristics were classified into two broad categories: the heuristic selection approach and the heuristic generation approach. The classification is based on the nature of the heuristic space, that is, these two approaches conduct different HLSs to explore the heuristic space. In heuristic selection approaches, existing LLHs are selected by the HLS to produce new heuristic algorithms. For example, Burke, Kendall, and Soubeiga (2003) proposed a Tabu search (TS) based hyper-heuristic for the timetabling problem and the rostering problem. Cuesta-Cañada et al. (2005) applied an ant-based algorithm to guide the LLH selection, in the domain of the 2D bin packing problem, and Dowsland et al. (2007) developed a simulated annealing (SA) based hyper-heuristic to select the LLHs for the shipper size decision problem.

On the other hand, unlike the heuristic selection approaches in which there are preexisting LLHs, in heuristic generation approaches, new heuristic algorithms are generated from basic components. Most of the existing heuristic generation approaches are based on genetic programming (GP; Burke, Hyde et al., 2009) and have been applied to various problem domains, such as the satisfiability problem (Fukunaga, 2008), the knapsack problem, and the bin packing problem (Burke et al., 2012), among others.

Apart from the above classification criterion that is based on the nature of the HLS, hyper-heuristics can also be classified according to the nature of the LLHs, that is, whether the LLHs used (either selected or generated) by the HLS are constructive or perturbative. In this paper, we concentrate on the perturbation-based LLH selection, due to its promising generality. As stated in Burke, Hyde, Kendall, Ochoa, Özcan, and Qu (2010), perturbative LLH selection approaches can be considered closely relevant to adaptive operator selection (AOS) methodologies and adaptive memetic algorithms (AMA; Ong et al., 2006) from the evolutionary computation community. Thus, many mechanisms of these approaches may be potentially applicable to perturbative LLH selection, such as the probability matching rule (Thierens, 2005), the dynamic multi-armed bandit (DMAB) based operator pursuit (DaCosta et al., 2008), and the subproblem decomposition (Ong et al., 2006). Furthermore, Aarts and Lenstra (1997, p. 4) claim that, "in many combinatorial optimization problems, solutions can be represented as sequences or partitions. These solution representations enable the use of k-exchange neighborhoods." Thus, the LLHs (such as k-exchange-based local search, shake, etc.) of one problem are likely to be adapted to other problems that have similar solution representations. As a result, perturbative hyper-heuristics are relatively easy to be transplanted to new problem domains. A similar idea was also mentioned in Burke, Hyde, Kendall, Ochoa, Özcan, and Woodward (2010).

However, despite the promising generality of the perturbative heuristic selection approaches, there is still room for improvement. For example, the LLHs employed in perturbative heuristic selection approaches are usually parameterized. This may contradict the goal of hyper-heuristics. First, if the LLPs are manually tuned, it would violate

the objective of algorithm design automation. Second, if the values of the LLPs are assigned with the values reported in the literature from which the LLHs are extracted, hyper-heuristics may not perform well. The reason is that given a problem domain, the distribution of the instances may vary greatly. Thus, a set of LLP configurations may perform well over one class of instances, but perform poorly over another class of instances. As an alternative, the adaptive maintenance of the LLPs seems an intuitive and promising direction. However, the adaptive maintenance of the LLPs has not been well investigated. As far as we know, there is only one framework, Hyflex (Burke, Curtois, Hyde et al., 2009; Burke, Curtois, Kendall et al., 2009), issuing the LLPs. In Hyflex, two parameters, intensity of change and depth of search, are introduced to control the behavior of certain LLHs. However, there are several limitations with this framework. For example, these two parameters are globally set, and the effects of these two parameters are problem and heuristic dependent (Burke, Curtois, Hyde et al., 2009), and thus may pose more challenges to the domain experts, for example, extra information about the LLPs (other than the ranges of the feasible values) has to be provided. As a result, more domain expert intervention is required for the algorithm.

2.2 Parameter Setting

The configuration of the parameters has a great influence on the performance of the algorithms. However, searching for the appropriate configuration of the parameters is itself a difficult problem. Currently there exist two main parameter setting techniques, offline parameter tuning and online parameter control (Eiben et al., 2007). Traditionally, the parameter tuning task is manually conducted in a trial and error fashion, which is usually time-consuming and error prone. Consequently, extensive interest has been focused on the automation of parameter tuning. For example, in order to automate the tuning procedure of the parameters, various methodologies have been proposed, such as machine learning (Birattari et al., 2010) and advanced local search (Hutter et al., 2009), among others. These approaches have been demonstrated to be very effective in achieving promising parameter configurations.

However, the automatic tuning techniques may suffer from several potential problems. First, in most offline tuning algorithms, the algorithm to be tuned has to be performed multiple times over the training instances, to collect the statistical evidence of whether one parameter configuration outperforms another. As a result, there is still computational overhead in these approaches. Second, most offline parameter tuning methodologies assume that the training instances accurately capture the distribution of the test instances (Hutter et al., 2009). However, this assumption may not always hold. For example, if the distributions of the instances are highly heterogeneous, or when the training instances and the test instances are extracted from different distributions, the offline tuning methodologies may not perform well.

Alternatively, the online control of the parameters has gained much attention. Based on the way the parameters are maintained, online parameter control methodologies can be classified into three categories (Eiben et al., 2007).

> **Deterministic**. In this approach, the values of the parameters are varied according to some prescheduled strategies. For example, Merkle et al. (2002) proposed an ant-based algorithm, in which the evaporation rate ρ starts from a small value, and gradually increases so as to achieve convergence.

Adaptive. In this approach, the values of the parameters are maintained based on the feedback information collected from the search procedure, which may involve the quality of the solution, the diversity of the population, and so on. Thus, this approach can also be considered similar to perturbative LLH selection-based hyper-heuristics, as well as DMAB-based operator selection (DaCosta et al., 2008), since all these approaches are based on the communication of the feedback information. For example, the classical 1/5 rule in evolution strategy (ES) is an adaptive parameter control approach (Rechenberg, 1973). Y. Li and W. Li (2007) developed an adaptive ant-based algorithm, in which the parameters α and β are adaptively adjusted based on entropy calculation.

Self-Adaptive. Instead of interacting with the search procedure according to some feedback information, in self-adaptive approaches, the parameters are encoded into the solution, and get optimized along with the exploration of the solution space. One example is the σ-self-adaptation in ES investigated by Hansen (2006). Serpell and Smith (2010) discussed the self-adaptation of the mutation operator for the permutation representations. See Meyer-Nieberg and Beyer (2007) for a survey of self-adaptive parameter control.

This concludes the introduction of both hyper-heuristics and parameter setting techniques. Interestingly, we observe that these two research directions have much in common. From the perspective of motivations, both directions aim at preventing the algorithms from being instance and/or problem dependent. From the perspective of methodologies, these two directions can also be considered relevant, especially when we compare perturbative LLH selection-based hyper-heuristics and adaptive parameter control approaches. Thus, in this paper, we intend to integrate perturbative LLH selection-based hyper-heuristics and adaptive parameter control into a unified framework, so as to improve the generality and the robustness of hyper-heuristics.

3 Hyper-Heuristics with Low Level Parameter Adaptation

As mentioned in Section 1, static LLP configurations may suffer from a series of problems, such as the training overhead, as well as the generality issue. As a solution, we intend to propose a general framework that incorporates the LLP adaptation.

The main idea of AD-HH is simple. In the framework, the HLS is further decomposed into two modules, the LLH management module (denoted as $\mathcal{M}_{\text{LLH}}$), and the LLP management module (denoted as $\mathcal{M}_{\text{LLP}}$). On one hand, $\mathcal{M}_{\text{LLH}}$ selects, applies, and evaluates the LLHs in a similar way as most of the existing hyper-heuristics. On the other hand, $\mathcal{M}_{\text{LLP}}$ is responsible for the selection and evaluation of LLPs. Figure 2(a) describes the framework diagram, in which the two modules communicate with each other so as to pass the values of LLPs, the feedback information about the quality of the applied LLPs, and so on. Note that since the maintenance of the LLPs is achieved through the information exchange between $\mathcal{M}_{\text{LLH}}$ and $\mathcal{M}_{\text{LLP}}$, the LLP adaptation in this study should be classified as an adaptive parameter control approach (see Section 2).

More specifically, our framework consists of the following components: $\mathcal{H}$, $\mathcal{Q}$, $\mathcal{S}$, $\mathcal{M}_{\text{LLH}}$, and $\mathcal{M}_{\text{LLP}}$. Among these components, $\mathcal{H} = \{LLH_1, LLH_2, \ldots, LLH_N\}$ indicates the set of LLHs, where N is the number of the LLHs, $\mathcal{Q} = \{q_1, q_2, \ldots, q_{num}\}$ is a population of num LLH sequences. In the population, each sequence is defined as $q_i = \langle q_i^1, q_i^2, \ldots, q_i^{len} \rangle$, where $q_i^j \in \mathcal{H}$ is the jth LLH of the ith sequence in $\mathcal{Q}$, and len

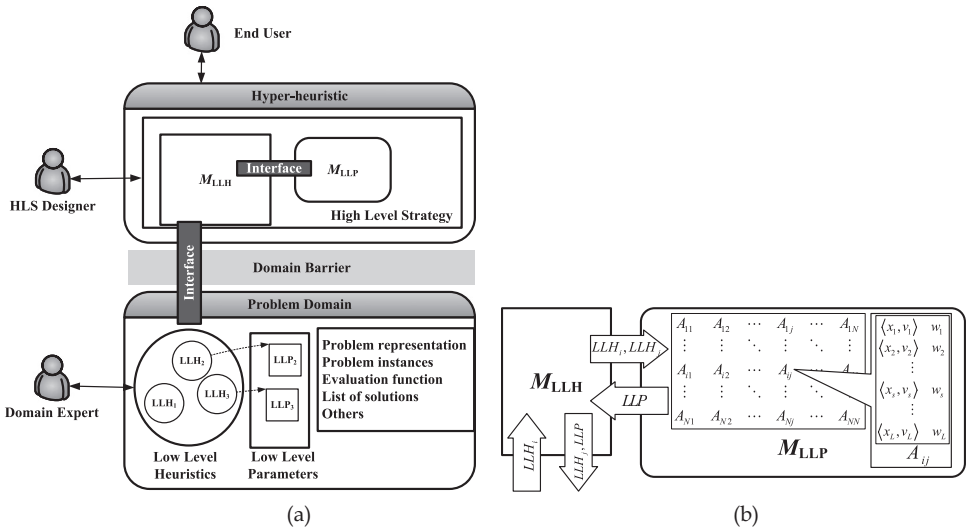


Figure 2: Hyper-heuristic with LLP adaptation. (a) AD-HH framework. (b) Illustration of the LLP generation.

indicates the length of each LLH sequence. Associated with each LLH sequence q_i, there exists a feasible solution s_i, over which the LLHs of q_i are applied, and the population of the solutions is defined as $\mathcal{S} = \{s_1, s_2, \ldots, s_{num}\}$.[1] Finally, $\mathcal{M}_{\text{LLH}}$ and $\mathcal{M}_{\text{LLP}}$ are the modules to manipulate the population of LLH sequences and LLPs, respectively.

With each component of the framework specified, we now present the pseudocode of AD-HH in Algorithm 1. Before the main loop, several components are initialized, including the initial LLH sequence population $\mathcal{Q}'$, the initial solution population $\mathcal{S}'$, $\mathcal{M}_{\text{LLH}}$, and $\mathcal{M}_{\text{LLP}}$ (lines 2–4). Then, at each generation of the main loop, a new LLH sequence population $\mathcal{Q}$ is first constructed with respect to the population $\mathcal{Q}'$ of the previous generation, and/or some other auxiliary information (line 6). After that, each LLH sequence $q_i \in \mathcal{Q}$ is applied over s_i for actual problem solving. Unlike most of the existing hyper-heuristics, in this study, the LLHs may be parameterized. As a solution, before invoking the parameterized LLH, the corresponding LLP is to be selected using $\mathcal{M}_{\text{LLP}}$ (lines 10–11). When a sequence of LLHs is executed (line 12), and returns the feedback information, the information is used to evaluate the LLHs and the corresponding LLPs (line 13). Once all the LLH sequences in the population have been applied, the LLH sequence population $\mathcal{Q}'$ for the next generation is to be selected (line 14). In this study, binary tournament selection is employed. Meanwhile, the selection of $\mathcal{S}'$ is implicitly conducted, since each LLH sequence corresponds to a feasible solution. Finally, if necessary, the structures of $\mathcal{M}_{\text{LLH}}$ and $\mathcal{M}_{\text{LLP}}$ are updated (line 16).

Note that AD-HH is a generic framework in which several interfaces (underlined in Algorithm 1) have to be implemented so that the framework can be instantiated for problem solving. For example, we have to provide an $\mathcal{M}_{\text{LLH}}$ specific method to construct new LLH sequences (line 6), an $\mathcal{M}_{\text{LLP}}$ specific method to assign LLP values

[1]By associating each LLH sequence with a solution, the quality of the LLH sequences can easily be measured using the objective values of their corresponding solutions.

Algorithm 1 AD-HH

Input: number of the LLH sequences num, length of each LLH sequence len, LLH management module $\mathcal{M}_{\text{LLH}}$, LLP management module $\mathcal{M}_{\text{LLP}}$
Output: the best solution achieved $best$

1 **begin**
2 Initialize $\mathcal{M}_{\text{LLH}}$ and $\mathcal{M}_{\text{LLP}}$
3 Randomly initialize a population of LLH sequences $\mathcal{Q}'$
4 **for** *each LLH sequence* ${q'}_i \in \mathcal{Q}'$ **do** Randomly initialize an associated solution ${s'}_i$
5 **while** *stopping criterion not met* **do**
   `// Construct Q using` $\mathcal{M}_{\text{LLH}}$, $\mathcal{Q}'$, `and/or other context information`
6 $\mathcal{Q} \leftarrow$ ConstructSequences ($\mathcal{M}_{\text{LLH}}$, $\mathcal{Q}'$)
7 **for** *each LLH sequence* $q_i \in \mathcal{Q}$ **do**
8 $s_i \leftarrow {s'}_i$; `// Assign the associated solution from its parent`
9 **for** $j = 1$ *to* len **do**
10 **if** q_i^j *is parameterized* **then**
11 Set the value for the parameter of q_i^j using $\mathcal{M}_{\text{LLP}}$
12 Apply q_i^j on s_i, along with the associated parameter
13 Evaluate the LLH sequence and the corresponding parameters with the objective value of s_i
14 $\mathcal{Q}' \leftarrow$ Select($\mathcal{Q} \cup \mathcal{Q}'$)
15 Record the currently best solution achieved so far as $best$
16 UpdateStructure ($\mathcal{M}_{\text{LLH}}$, $\mathcal{M}_{\text{LLP}}$) ; `// optional`
17 **return** $best$
18 **end**

(line 11), and (possibly) structure updating methods for $\mathcal{M}_{\text{LLH}}$ and $\mathcal{M}_{\text{LLP}}$ (line 16). In the following sections, we discuss the design and implementation of these modules. In particular, in Section 3.1, we propose an ant-based LLP adaptation mechanism. In Section 4, we discuss how to instantiate AD-HH in the context of two different $\mathcal{M}_{\text{LLH}}$ models, so as to demonstrate the flexibility of the framework.

3.1 Ant-Based Low Level Parameter Adaptation

In this section, we discuss the details of $\mathcal{M}_{\text{LLP}}$, which is inspired by ant-based algorithms. We choose the ant model to achieve the LLP adaptation based on the following considerations. First, most ant-based algorithms conduct the search in a restart paradigm (i.e., at each iteration, ant-based algorithms construct the solutions from scratch). This feature is suitable for LLP adaptation, since in hyper-heuristics, the LLH selection is usually highly dynamic. Second, ant-based algorithms provide an implicit learning mechanism based on the indirect communication between a colony of artificial ants, which is also suitable for intelligently selecting the LLP values. Finally, the pheromone structure of ant-based algorithms provides an intuitive but effective description of the distribution of the search space. Thus, in this study, we propose an ant-based LLP adaptation mechanism.

Recall that in Algorithm 1, the unique feature of AD-HH lies in the LLP adaptation. At each decision point LLH_i, the HLS selects the next LLH_j and the associated parameter in two phases. First, LLH_j is selected according to $\mathcal{M}_{\text{LLH}}$. Then, the value of the corresponding parameter is generated with respect to LLH_i and LLH_j. By maintaining the values of the LLPs for every possible LLH transition combination, we take the dependencies between the LLHs into account. This strategy is analogous to the ant model. In ant-based algorithms, the construction of the solution is conducted in an incremental way, and each variable is selected with respect to the previous variable, under the guidance of the pheromone matrix. As a result, we assume the value of each LLP should

be dynamically maintained for different LLH transitions, rather than globally assigned with the same value.

In the LLP adaptation mechanism, the LLPs are treated as variables, which get optimized along the search procedure. However, ant colony optimization (ACO) is traditionally designed to solve combinatorial optimization problems, and thus it is not directly applicable to LLP adaptation. The reason is that there are various types of parameters for the LLHs. For example, for the mutation operator of GA, the *mutation-rate* is usually a real-valued parameter; while for the shake operator of variable neighborhood search (VNS), the shake-strength is an integer parameter. For the rest of this section, we first outline the LLP adaptation mechanism, which is modeled as a mixed discrete-continuous problem. Then, the auxiliary data structures for maintaining the LLPs, as well as the pseudocode of the LLP adaptation, are presented in detail.

In order to deal with the continuous parameters, some modifications were introduced. Since the proposal of ACO, much study investigated problem domains other than the combinatorial optimization problems, such as continuous ACO (CACO; Bilchev and Parmee, 1995), $ACO_{\mathbb{R}}$ (Socha, 2004; Socha and Dorigo, 2008), and ACO_{MV} (Socha, 2004). Among these approaches, $ACO_{\mathbb{R}}$ and its extension ACO_{MV} provide a general model that is close to ACO for the discrete domain problems. In addition, ACO_{MV} has the advantage of being applicable to the mixed discrete-continuous optimization problems. Hence, in this study, our LLP adaptation mechanism is based on ACO_{MV}.

The main idea of extending ACO to the continuous domains is to replace the discrete probability distribution with the continuous probability density function (PDF). In ACO, a pheromone is employed to capture the characteristics of the variable distribution. In ACO_{MV}, instead of choosing the variables according to the pheromone matrix, a PDF is sampled to choose the value of each continuous variable. More specifically, a set of solutions is maintained in a collection called the solution archive to describe the variable distribution of the continuous solution space. Furthermore, in order to tackle the mixed discrete-continuous problems, ACO_{MV} first treats those discrete variables in the same way as the continuous ones. Then, before the objective function is applied to calculate the objective value of the solution, the continuous values are rounded to the nearest discrete values.

Analogous to the pheromone in ant-based algorithms, we employ a two-dimensional archive matrix $(A_{ij})_{N\times N}$ to achieve the LLP adaptation. Each element A_{ij} of the matrix represents an LLP archive. Associated with each LLH transition $\langle LLH_i, LLH_j\rangle$, if LLH_j is parameterized, we keep track of a number of LLP tuples in A_{ij}. The lth tuple of A_{ij} consists of the value x_l of the parameter[2] and its corresponding objective value v_l (supposing the problem to be solved is a minimization problem). An example of a parameter archive is presented in Figure 2(b).

For each A_{ij}, the distribution of the parameter x is described as a weighted sum of several Gaussian functions, in that such a PDF is easy to sample, and meanwhile provides sufficient flexibility. The number of the tuples stored in each archive is set to L, which also indicates the number of the Gaussian kernel functions as follows:

$$G_{ij}(x) = \sum_{l=1}^{L} w_l \frac{1}{\sigma_l\sqrt{2\pi}} e^{-\frac{(x-\mu_l)^2}{2\sigma_l^2}}, \tag{1}$$

[2]If there is more than one parameter for LLH_j, x_l should be replaced with a vector.

where w is the vector of the weights corresponding with each tuple, μ is the vector of the means, and σ represents the vector of the standard deviations for A_{ij}.

Note that the tuples in each A_{ij} are sorted in ascending order of the objective values, and the weight of the lth tuple is defined as:

$$w_l = \frac{1}{\xi L\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2\xi^2 L^2}}, \tag{2}$$

where ξ is a locality parameter (Socha and Dorigo, 2008). The smaller ξ is, the more the tuple with the best rank is preferred, while the larger ξ is, the more uniform the search behavior will be. From Equation (2) we observe that the better the rank is, the higher the weight of the corresponding tuple, which means that the adaptation mechanism prefers the best ranked tuples. More specifically, the probability of choosing the lth Gaussian function is defined as:

$$p_l = \frac{w_l}{\sum_{r=1}^{L} w_r}. \tag{3}$$

Suppose that the lth Gaussian function is selected as the kernel PDF, then the mean value and the standard deviation are given by:

$$\mu = x_l, \tag{4}$$

$$\sigma = \sum_{r=1}^{L} \frac{|x_r - x_l|}{L-1}. \tag{5}$$

With μ and σ, the parameter value can be generated using the Box-Muller method (Box and Muller, 1958). The process of parameter generation is described in Algorithm 2, and is illustrated in Figure 2(b) as well. After the parameter is generated from A_{ij}, and applied to LLH_j, the feedback information of the execution is used to update A_{ij}, as presented in Algorithm 3. Since the initialization of the archive matrix A is trivial, that is, each archive A_{ij} is initialized to be empty, the corresponding pseudocode is not presented. By invoking Algorithms 2 and 3 in Algorithm 1 (lines 11 and 16, respectively), we can incorporate the LLP adaptation mechanism in AD-HH.

As a final note, in the ant-based $\mathcal{M}_{\text{LLP}}$ proposed in this section, we adopt a two-dimensional archive matrix A to capture the distribution of LLPs. Alternately, if the LLPs are to be globally set, rather than being dependent on the LLH transition, we can simply set the dimension of A to be 1. In the experiments in Section 6, we briefly compare these two design strategies. We also examine whether the ant model is capable of learning effective LLP configurations.

4 Management of Low Level Heuristics

In this section, we instantiate the AD-HH framework in the context of two different $\mathcal{M}_{\text{LLH}}$ models, which are abstracted from an ant-based hyper-heuristic (denoted as AH) and a GA-based hyper-heuristic (denoted as GH), respectively. For each $\mathcal{M}_{\text{LLH}}$, we first introduce the background information of the model. Then, the main factors of each model are presented and discussed. After that, we present the modifications that have to be made to incorporate each model into the AD-HH framework. For each

Algorithm 2 GenerateParameter

Input: the index i and j of the two LLHs
Output: the parameter value p
begin
  if *The number of the tuples in* A_{ij} *is less than* L **then**
    Uniformly generate the value of p within the range of the parameter
    return p
  Select the lth Gaussian function with probability calculated by Equation (3)
  Set the mean value of the PDF as $\mu = x_l$
  Calculate the standard deviation σ by Equation (5)
  Generate the value of the parameter p with Box-Muller method, with μ and σ as input
  if *The parameter is required to be discrete* **then**
    Round p to its nearest discrete value
  return p
end

Algorithm 3 UpdateArchive

Input: the index i and j of the two LLHs, the value of the parameter p, the objective value v obtained from the LLH$_j$'s execution
begin
  if *There exists a tuple* $\langle x_l, v_l \rangle$ *such that* $x_l = p$ **then**
    $v_l \leftarrow v$
  else
    Construct a new tuple $\langle p, v \rangle$
    Insert the tuple into A_{ij}
  Sort A_{ij} in ascending order of the objective values of each tuple
  if *The number of tuples exceeds* L **then**
    Exclude the tuple with the largest objective value
end

model, we illustrate the implementation details that are required by AD-HH, including the LLH sequences constructing, and (possibly) the structure updating procedures (see Algorithm 1).

4.1 Ant-Based $\mathcal{M}_{\text{LLH}}$

AH is a class of perturbative heuristic selection approaches that has attracted much research interest. Burke et al. (2005) proposed an AH to solve the project presentation problem. Cuesta-Cañada et al. (2005) applied an AH with multiple pheromone matrices for the 2D bin packing problem. A recent AH algorithm was proposed by Chen et al. (2007) to solve the travelling tournament problem. Various applications have demonstrated the generality of AH. In this section, the modifications that have to be made to incorporate AH into AD-HH are discussed and described in pseudocode.

In existing AHs (Burke et al., 2005; Chen et al., 2007), LLH management is conducted as follows. First, a fully connected graph is constructed where each vertex represents an LLH, and the arcs between the vertices indicate the invocation sequence relationship between the LLHs. Each ant is associated with a solution, and the path of each ant corresponds to a sequence of LLHs. Then, at each iteration, each artificial ant k traverses the graph to construct the LLH sequence, which is denoted as q_k^1 through q_k^{len}, where len indicates the length of each sequence. During the construction phase, the selection of the LLHs is guided by the pheromone, with each entry representing the desirability of the transition between the LLHs. After each LLH sequence is constructed and applied

Algorithm 4 ConstructSequences-AH

```
   Input: A population of LLH sequences Q', A pheromone matrix τ
   Output: A new population of LLH sequence Q
1  begin
2      Q ← Q'
3      foreach LLH sequence q_i ∈ Q do
4          q_i^1 ← q_i^len
5          for j = 2 to len do Select the next LLH q_i^j with respect to Equation (6)
6      return Q
7  end
```

over the associated solution, the pheromone is then updated according to the quality of the solutions obtained.

As described, the pheromone matrix $(\tau_{ij})_{N\times N}$ is the most important structure in AH, where N indicates the size of the LLH set $\mathcal{H}$. Each element τ_{ij} represents the desirability of guiding the ants from LLH_i to LLH_j. With the pheromone τ, we can define the probability of the transition from LLH_i to LLH_j at each iteration:

$$P_{ij} = \frac{\tau_{ij}}{\sum_{LLH_l \in \mathcal{H}} \tau_{il}}. \tag{6}$$

Note that during the initialization stage, each element of τ_{ij} is uniformly assigned small random values, indicating that the LLH selection at the beginning is conducted randomly. Then, with the accumulation of the pheromone, the selection of the LLHs is gradually biased, favoring promising LLH transitions.

Algorithm 4 presents a more detailed pseudocode with the pheromone τ illustrated that fulfills the LLH sequence construction requirement of AD-HH. For each ant, the LLH sequence starts with the last LLH of its parent (line 4). Then, the ant traverses the fully connected graph to select the next LLH (line 5). Along the traversal, the transition probability between the LLHs is calculated based on Equation (6). After all the LLH sequences have been constructed, the population of the LLH sequences is returned for problem solving.

Since AH employs the pheromone τ to describe the LLH transition probability, at each iteration, after an LLH sequence is applied, we have to update τ with respect to its quality. (Recall that in this study, the quality of each LLH sequence is measured by the objective value of the solution corresponding to the LLH sequence.) Thus, after the LLHs are selected and applied on the solution associated with each ant k, the pheromone τ is updated using Equation (7):

$$\tau_{ij} = \begin{cases} \rho \cdot \tau_{ij} + \dfrac{v_{\text{best}}}{v_k} & LLH_i \text{ and } LLH_j \text{ are along the journey of ant } k, \\ \rho \cdot \tau_{ij} & \text{otherwise,} \end{cases} \tag{7}$$

where v_k and v_{best} (recall that we suppose the problem to be a minimization problem) represent the objective values of the solution associated with ant k and the currently best solution achieved by the search process, respectively, and ρ indicates the evaporation rate. The pseudocode for updating τ in AH is presented in Algorithm 5. By embedding Algorithms 4 and 5 in AD-HH (lines 6 and 16 of Algorithm 1, respectively), we obtain AD-AH.

Algorithm 5 UpdateStructure-AH

```
Input: A population of LLH sequences 𝒬, A pheromone matrix τ
begin
    foreach LLH sequence q_i ∈ 𝒬 do
        for j = 2 to len do
            Update τ corresponding to q_i^{j-1} and q_i^j using Equation (7)
end
```

Algorithm 6 ConstructSequences-GH

```
Input: A population of LLH sequences 𝒬'
Output: A Modified population of LLH sequence 𝒬
begin
    𝒬 ← ∅
    foreach LLH sequence q_i ∈ 𝒬' do
        Randomly select another sequence q_j such that q_j ∈ 𝒬', q_j ≠ q_i
        offspring ← crossover-hyper (q_i, q_j)
        offspring' ← mutate-hyper (offspring)
        𝒬 ← 𝒬 ∪ {offspring'}
    return 𝒬
end
Procedure crossover-hyper (LLH sequence q_i, LLH sequence q_j)
begin
    offspring^1 ← q_i^len
    c ← random(2, len)
    for k = 2 to c do offspring^k ← q_i^k
    for k = c + 1 to len do offspring^k ← q_j^k
    return offspring
end
Procedure mutate-hyper (LLH sequence offspring, mutation rate θ)
begin
    for i = 1 to len do Randomly change the LLH of offspring^i with probability θ
    return offspring
end
```

4.2 Genetic Algorithm-based $\mathcal{M}_{\text{LLH}}$

In this section, we focus on the GA-based $\mathcal{M}_{\text{LLH}}$, which is abstracted from GH. First, we briefly introduce the background of GH. Then, the modifications that have to be made for GH are presented and discussed.

GH was first addressed by Cowling et al. (2002), to solve the trainer scheduling problem. In the paper, the authors proposed Hyper-GA, which applies a GA model with a one-point crossover operator and a uniform mutation operator to manipulate a set of LLHs. Later in the same year, Han et al. (2002) improved Hyper-GA by adaptively maintaining the lengths of the LLH sequences, and proposed ALChyper-GA for the same problem. Since then, there have been various GH variants. For simplicity, the GH model in this section is similar to Hyper-GA.

In GH, each LLH sequence $q_i \in \mathcal{Q}$ represents a chromosome of the population. At each iteration, new chromosomes are constructed by applying genetic operators (e.g., crossover and mutation operators) over the chromosomes of the previous iteration. After the new chromosomes are constructed, the LLHs in each chromosome are applied for problem solving. Similar to AH, the fitness of each chromosome is evaluated using the solution quality returned by the LLHs. Then, the chromosomes with better fitness are selected as the population of the next generation. The process continues until the stopping criterion is met.

To instantiate the GA-based $\mathcal{M}_{\text{LLH}}$, the pseudocode for the GA-based LLH sequences constructing method is presented in Algorithm 6. In the algorithm, the LLH

sequences of the offspring generation are produced by conducting crossover and mutation operators over the LLH sequences of the previous generation. In particular, for each LLH sequence $q_i \in \mathcal{Q}'$ of the population, another LLH sequence $q_j \in \mathcal{Q}'$ is first selected randomly (line 4), as the other parent of the crossover procedure. Then, the one-point crossover operator is applied to construct the LLH sequence $offspring$, with respect to its parents (line 5). After that, the offspring undergoes the mutation procedure (line 6) to obtain $offspring'$, to achieve more diversity in the search within the heuristic space. Finally, $offspring'$ is inserted into $\mathcal{Q}$, which is returned as the LLH sequence population of the next iteration. In more detail, the implementation of the crossover and the mutation operators is also listed (lines 10–22). Unlike AH and its variants, in GH, no auxiliary structures such as the pheromone matrix have to be maintained. As a consequence, by embedding Algorithm 6 into AD-HH (line 6 of Algorithm 1), we obtain AD-GH.

In summary, in this section, we instantiate AD-HH in the context of an ant-based $\mathcal{M}_{\text{LLH}}$ and a GA-based $\mathcal{M}_{\text{LLH}}$, respectively. In the following section, we shall discuss the potential risk of the LLP adaptation, as well as a possible solution.

5 Heuristic Space Reduction by Low Level Heuristic Bipartition

The LLP adaptation alleviates the laborious tuning task of the LLPs. However, since the LLPs are introduced as variables to be optimized, the search space is expanded accordingly. In this section, we propose heuristic space reduction (SAR) mechanisms for AD-AH and AD-GH, and develop AD-AHSAR and AD-GHSAR, respectively, so as to prevent the search space from drastic expansion.

The motivation of the SAR mechanism is based on the observation that redundancies may exist in LLH-selection-based hyper-heuristics. As stated elsewhere (Burke et al., 2005; Özcan et al., 2008), there are two main LLH move acceptance criteria: the any moves (AM) hyper-heuristics that accept any LLH sequences, and the only improving (OI) hyper-heuristics that only accept LLH sequences that improve the solution quality. Burke et al. (2005) claim that AM outperforms OI, in that the OI criterion provides no diversification mechanism, and may easily get trapped in local optima. In essence, both AD-AH and AD-GH discussed in Section 4 employ the AM criterion.

However, in the AM criterion, all the LLHs are treated in an equivalent way, which may lead to redundancy during the search over the heuristic space. For example, if a random restart heuristic is invoked immediately after a greedy heuristic, the execution of this greedy heuristic is generally a waste of time. Similarly, if the same local search operator is consecutively executed, it is impossible that the search could escape from the local optima.

Based on the fact that the existing LLH move acceptance criteria are either too greedy or too generous, we intend to develop a more balanced move criterion. The SAR mechanisms are inspired by the definition of metaheuristics, which interprets the process of optimization as the combination of the intensification and the diversification strategies. Motivated by this definition, in the SAR mechanism, we explicitly accept the LLHs of different functionalities at each iteration. Our mechanism in essence reduces the heuristic space by restricting the LLHs to be selected from the Cartesian product of the two subsets of the LLHs. With this mechanism, the heuristic space can be significantly reduced, and the exploration could be more effective.

To present the SAR mechanism, we first briefly review the definition of metaheuristics, as well as the two main functionalities of the LLHs employed in metaheuristics.

Over the last few decades, great efforts have been focused on various metaheuristics, including GA (Holland, 1992), ACO (Dorigo et al., 1996), VNS (Hansen and Mladenović, 2001), greedy randomized adaptive search procedure (GRASP; Feo and Resende, 1995), and so on. Despite appearing to be far different from each other, these algorithms have much in common (Taillard et al., 2001). A metaheuristic is defined as "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space" (Osman and Laporte, 1996, pp. 513–514), or the combination of intensification and diversification (Blum and Roli, 2003). In the definition, exploitation (also called intensification) indicates those strategies that conduct an intensive search in order to improve solution quality; while exploration (also called diversification) refers to the strategies that lead the search to the diverse regions of the solution space. Metaheuristics intend to balance the intensification and the diversification strategies by combining those intensification and diversification LLHs. During the intensification process, those LLHs such as local search heuristics are usually applied to improve solution quality; during the diversification process, various perturbative heuristics such as crossover, mutation, and shake are employed to guide the search procedure to new regions of the solution space.

For the rest of this section, we discuss the SAR mechanism, in the context of both AD-AH and AD-GH, so as to prevent the search space from drastic expansion.

5.1 Heuristic Space Reduction for AD-AH

In this section, we describe the SAR mechanism for AD-AH. As mentioned in Section 4.1, the LLH sequences in AD-AH are selected by traversing the fully connected graph induced by all the LLHs, which implies that the AM criterion is employed in AD-AH. To achieve heuristic space reduction, we replace the fully connected graph induced by all the LLHs with a bipartite graph induced by two subsets of the LLHs, so as to reduce the redundancy in the existing acceptance criteria. Without loss of generality, given a minimization problem, let Π be the solution space, with objective function $f : \Pi \mapsto \mathbb{R}$, and a heuristic is defined as a function $h : \Pi \mapsto \Pi$. Let $\mathcal{H}$ be the set of LLHs. The intensification LLH set $\mathcal{I}$ and the diversification LLH set $\mathcal{D}$ are defined as:

$$\mathcal{I} = \{i | i \in \mathcal{H}, \forall \pi \in \Pi, f(i(\pi)) \leq f(\pi)\}, \tag{8}$$

$$\mathcal{D} = \{d | d \in \mathcal{H}, \exists \pi \in \Pi, f(d(\pi)) > f(\pi)\}. \tag{9}$$

Similar LLH division criteria have been issued in other studies as well. For example, Özcan et al. (2006) classified the LLHs into hill climbers and mutational heuristics; Meignan et al. (2010) partitioned the LLH set into an intensifier set and a diversifier set. Another division criterion was proposed in Burke, Curtois, Kendall et al. (2009), in which the diversification LLHs were further classified into mutational, ruin-recreate, crossover, and others. For simplicity, we only bipartition the LLH set. For those heuristics with more than one input solution, such as the crossover of GA, we only need to replace the objective function f with some other evaluation function. Instead of traversing the fully connected graph in search of LLHs, at each iteration of the search process, the path of each solution is constructed by traversing the bipartite graph, and the LLH transition

Algorithm 7 Procedure mutate-hyper-reduction (*LLH sequence q, mutation rate θ*)

```
   Output: mutated LLH sequence
1  begin
2      for i = 2 to len do
3          if (len − i)%2 = 0 then
4              if q^i ∉ I then Randomly select q^i ∈ I
5              else if random (0,1) < θ then Randomly select q^i ∈ I
6          else
7              if q^i ∉ D then Randomly select q^i ∈ D
8              else if random (0,1) < θ then Randomly select q^i ∈ D
9      return q
10 end
```

probability is modified as:

$$P'_{ij} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{LLH_l \in \mathcal{D}} \tau_{il}} & LLH_i \in \mathcal{I}, LLH_j \in \mathcal{D}, \\ \dfrac{\tau_{ij}}{\sum_{LLH_l \in \mathcal{I}} \tau_{il}} & LLH_i \in \mathcal{D}, LLH_j \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The only difference between AD-AH and AD-AHSAR lies in the definition of the LLH transition probability. With P_{ij} in Equation (6) replaced by P'_{ij} in Equation (10), the SAR mechanism can be easily integrated into the AD-AH framework. By dividing the LLH set into two subsets, and traversing the bipartite graph induced by the Cartesian product of these two subsets, the consecutive execution of the intensification and the diversification LLHs can be guaranteed.

5.2 Heuristic Space Reduction for AD-GH

Similar to AD-AH, in essence, the AM criterion is employed in AD-GH. Thus, in this section we discuss how to conduct the SAR mechanism for AD-GH. In order to achieve the reduction of the heuristic space, we modify the procedure mutate-hyper from Algorithm 6, so as to reduce the search space, meanwhile balancing the intensification and the diversification of the search process. In more detailed fashion, the modifications are presented in Algorithm 7.

In the procedure, the mutation of the LLH sequence is conducted under two circumstances. For those LLHs that violate the consecutive invocation constraint (note that we restrict that the last LLH of each sequence be an intensification LLH, so that the output solution produced is a local optimum), a random selection is conducted so that the constraint is satisfied (line 4 and line 7). On the other hand, for those LLH for which the constraint is not violated, the mutation is carried out with probability θ, which is similar to the one used in the procedure mutate-hyper.

In summary, in this section, we exemplify the feasibility of the heuristic space reduction in the contexts of both AD-AH and AD-GH. As a result, we can combine the LLP adaptation and the SAR mechanism into an integrated framework, and develop AD-AHSAR and AD-GHSAR. In the following section, we examine the effectiveness of each mechanism, as well as their combination through extensive experiments.

6 Empirical Study

In this section, we choose the p-median problem as a case study to test the performance of our algorithms. We first introduce the background of the p-median problem, as well as the LLHs that are employed in the hyper-heuristic framework. Then, extensive experiments are conducted so as to evaluate the effectiveness and the efficiency of the proposed algorithms.

6.1 Preliminaries

Before presenting the empirical results, we first introduce the background information of the p-median problem. The reasons we choose the p-median problem are as follows. First, it is a classic NP-hard problem (Kariv and Hakimi, 1979) from the location theory with wide applications ranging from industry to data mining. Second, for the p-median problem, there exist various metaheuristics from which the LLHs can be extracted, such as VNS (Hansen and Mladenović, 1997), ACO (Kochetov et al., 2005), and GA (Correa et al., 2001). In addition, there are several parameterized LLHs for the p-median problem, which makes it suitable as the test problem.

We take as given a set F of m facilities, a set U of n users, and an $n \times m$ matrix C with the cost traveled c_{ij} for satisfying the demand of the user located at i from the facility located at j, for all $j \in F$ and $i \in U$. The objective of the p-median problem is to select a subset $J \subseteq F, |J| = p$ from all the facilities of set F, so as to minimize the sum of these costs:

$$\min \sum_{i \in U} \min_{j \in J} c_{ij}. \tag{11}$$

Since each solution to a p-median instance consists of a subset of medians with a fixed number p, in this study we adopt fixed length encoding (Crawford et al., 1997). In this encoding, each solution consists of a list of p integers, and each element of the list represents the index of a facility that is selected as a median. All the LLHs are extracted from those existing metaheuristics mentioned above, and are listed as follows. Among these LLHs, the interchange and the LK(k) are intensification LLHs, while the rest of the LLHs are diversification LLHs.

6.1.1 Interchange

Interchange was first proposed in Teitz and Bart (1968), and widely used in the metaheuristics, such as VNS (Hansen and Mladenović, 1997) and GRASP (Resende and Werneck, 2004). This heuristic iteratively swaps facilities, aiming to reduce the objective value, until no move can be applied. Note that there are two popular implementations of this LLH, proposed by Whitaker (1983) and Resende and Werneck (2003). The two versions have the same complexity for traversing an interchange neighborhood. However, the Resende and Werneck (2003) implementation is much faster, which is based on a well designed space-time trade-off. In this paper, we employ the Resende and Werneck implementation.

6.1.2 LK(k)

LK(k) is extracted from ACO (Kochetov et al., 2005), in which $k \in \{1, 2, \ldots, \min\{p, m - p\}\}$ is a depth parameter. Traversing an LK(k) neighborhood involves k swaps, which

is k times that of interchange. From the implementation aspect, since the interchange neighborhood is a subset of an LK(k) neighborhood (Kochetov et al., 2005), LK(k) can benefit from the trade-off as in Resende and Werneck (2003). Thus, an implementation based on Resende and Werneck (2003) is adopted. In the previous version of this work (Ren et al., 2010), the value of k was sampled with three typical values. In this paper, k is treated as a parameter that is adaptively maintained instead.

6.1.3 Crossover and Mutation

The two heuristics crossover and mutation are extracted from GA (Correa et al., 2001). Note that crossover requires two input solutions, and generates one offspring. In particular, for mutation, each median is swapped with a random nonmedian facility with a probability indicated by mutation-rate.

6.1.4 Initialization with Pheromone (AntInit)

At each iteration of ACO (Kochetov et al., 2005), each solution is constructed with probability according to the pheromone trail.

6.1.5 Shake

Proposed in VNS (Hansen and Mladenović, 1997), shake can be viewed as a special case of mutation, with the input provided by the currently best solution. This operator has a parameter shake-strength that represents the distance (indicated by the number of different medians) between the input solution and the output solution of the shake operation.

6.1.6 Random

Random is a restart operator that can provide diversification functionality as well.

6.1.7 Random Plus Greedy (RPG)

First proposed in Resende and Werneck (2004), RPG is also a restart operator in which randomness is combined with the greedy operator. In this operator, the first half of the medians are randomly selected, while the rest of the medians are greedily selected.

All the experiments in this paper were performed on a Pentium IV 3.2 GHz PC with 4 GB memory, running GNU/Linux with kernel 3.0.0. All the codes were implemented in C++, compiled using g++ 4.5 with flag -O2. The run time is measured in seconds. To examine the generality of the algorithms, we consider three heterogeneous classes of benchmark instances, including 40 graph-based instances from the ORLIB class (Beasley, 1985), five random instances from RW (Resende and Werneck, 2003), and 10 Euclidean-based instances from the TSPLIB class (Reinelt, 1991). Each instance from ORLIB is represented as a graph with a corresponding p. Every node of the graph is considered as both a user and a facility. The distance between a user and a facility is the length of the shortest path between the two nodes in the graph. On the other hand, each RW instance is generated using random distance matrices. The distance from each user to each facility is an integer value, and is uniformly distributed within the interval $[1, n]$. The main feature of the RW instances is that the distance from user i to facility j does not always equal the distance from user j to facility i. In addition, the distance from facility i to user i does not equal 0. For the distance matrix, we chose rw1000, with p varying from 100 to 500 (note that the same cost matrix with different p would yield

Table 1: HLS parameter configurations.

Parameter	Value	Source
Length of each LLH sequence: len	5	—
Population size: num	10	(Ren et al., 2010)
Maximum iteration: $iter$	100	(Ren et al., 2010)
Evaporation rate in Algorithm 3: ρ	0.1	(Ren et al., 2010)
Mutation rate in Algorithms 6 and 7: θ	0.1	(Cowling et al., 2002)
Parameter archive size: L	50	(Socha and Dorigo, 2008)
Parameter weight locality: ξ	$1e-4$	(Socha and Dorigo, 2008)

different instances). Finally, the TSPLIB instances were first introduced in the context of the p-median problem by Hansen and Mladenović (1997), and were then widely used to test the performance of various algorithms (Hansen and Mladenović, 2001; Resende and Werneck, 2004; Pullan, 2008). For each instance, every point represents both a user and a potential facility, and the distance from a user to a facility is the Euclidean distance between the two points. In this study, the distance matrix is derived from fl1400, with varying values of p from 50 to 500. For the three chosen instance sets, the ORLIB instances are relatively easy to solve, and have been exactly solved (Beasley, 1985). The scale of the solution space of the ORLIB instances (measured by the number of feasible solutions) ranges from $\binom{100}{5}$ to $\binom{900}{90}$. On the other hand, the scale of the RW instances ranges from $\binom{1000}{100}$ to $\binom{1000}{500}$, and the TSPLIB instances are the most difficult to solve, in that they have the largest search space (ranging from $\binom{1400}{50}$ to $\binom{1400}{500}$).

In hyper-heuristics, there are parameters in both HLSs and LLHs. In this study, since we concentrate on the LLP adaptation, we do not conduct a tuning of the HLS parameters. On the contrary, all the HLS parameters are set with the same values, so as to compare the influence of the LLPs in a fair way. The parameters are presented in Table 1; they are set with the same values as in previous works (Han et al., 2002; Socha and Dorigo, 2008; Ren et al., 2010), except for the length of each LLH sequence len. The reason that the setting of the parameter len is changed is as follows. In this study, we set the length of the LLH sequences to be the same for both AD-AH, AD-GH, and their variants. On one hand, in the previous version of this work (Ren et al., 2010), this parameter is set to 2, which is too small for AD-GH and its variants. On the other hand, the larger the value of len is, the longer it will take our algorithms to achieve convergence. Thus, for efficiency, the parameter len is set to 5 in this study.

6.2 Numerical Results

In this section, we present the performance of our algorithms AD-AHSAR and AD-GHSAR. For reference, we compare the performance of AD-AHSAR and AD-GHSAR with the state of the art results, which are provided by probabalistic bean search (PBS Pullan, 2008). PBS achieves the best known upper bounds for most benchmark instances. However, since PBS is a distributed algorithm tested in a much higher performance environment than in this study, we only report its best solution objective values over the instances.

To report the performance of AD-AHSAR and AD-GHSAR, each algorithm is performed for 20 independent runs over all the instances, and the results are presented in Table 2. In the table, the results are organized as follows. Columns 1–2 indicate the benchmark instances. Column 3 gives the best known results achieved by the state of

Table 2: Numerical results over benchmark instances.

			AD-AHSAR				AD-GHSAR			
Instance	ID	OPT	Best	Avg $\pm SD$	$\%err$	Time (s)	Best	Avg $\pm SD$	$\%err$	Time (s)
ORLIB	pmed1	**5,819**	**5,819**	5,819.00 ± 0.00	0.0000	1.06	**5,819**	5,819.00 ± 0.00	0.0000	1.00
	pmed2	**4,093**	**4,093**	4,093.00 ± 0.00	0.0000	0.81	**4,093**	4,093.00 ± 0.00	0.0000	0.73
	pmed3	**4,250**	**4,250**	4,250.00 ± 0.00	0.0000	0.86	**4,250**	4,250.00 ± 0.00	0.0000	0.74
	pmed4	**3,034**	**3,034**	3,034.00 ± 0.00	0.0000	0.77	**3,034**	3,034.00 ± 0.00	0.0000	0.72
	pmed5	**1,355**	**1,355**	1,355.00 ± 0.00	0.0000	0.78	**1,355**	1,355.00 ± 0.00	0.0000	0.68
	pmed6	**7,824**	**7,824**	7,824.00 ± 0.00	0.0000	6.76	**7,824**	7,824.00 ± 0.00	0.0000	5.95
	pmed7	**5,631**	**5,631**	5,631.00 ± 0.00	0.0000	3.69	**5,631**	5,631.00 ± 0.00	0.0000	3.06
	pmed8	**4,445**	**4,445**	4,445.00 ± 0.00	0.0000	2.63	**4,445**	4,445.00 ± 0.00	0.0000	2.48
	pmed9	**2,734**	**2,734**	2,734.00 ± 0.00	0.0000	2.04	**2,734**	2,734.00 ± 0.00	0.0000	1.94
	pmed10	**1,255**	**1,255**	1,255.00 ± 0.00	0.0000	2.58	**1,255**	1,255.00 ± 0.00	0.0000	2.34
	pmed11	**7,696**	**7,696**	7,696.00 ± 0.00	0.0000	12.44	**7,696**	7,696.00 ± 0.00	0.0000	12.88
	pmed12	**6,634**	**6,634**	6,634.00 ± 0.00	0.0000	11.00	**6,634**	6,634.00 ± 0.00	0.0000	10.16
	pmed13	**4,374**	**4,374**	4,374.00 ± 0.00	0.0000	4.49	**4,374**	4,374.00 ± 0.00	0.0000	3.98
	pmed14	**2,968**	**2,968**	2,968.00 ± 0.00	0.0000	4.45	**2,968**	2,968.00 ± 0.00	0.0000	3.90
	pmed15	**1,729**	**1,729**	1,729.00 ± 0.00	0.0000	5.22	**1,729**	1,729.00 ± 0.00	0.0000	4.14
	pmed16	**8,162**	**8,162**	8,162.00 ± 0.00	0.0000	37.15	**8,162**	8,162.00 ± 0.00	0.0000	39.83
	pmed17	**6,999**	**6,999**	6,999.00 ± 0.00	0.0000	21.15	**6,999**	6,999.00 ± 0.00	0.0000	17.02
	pmed18	**4,809**	**4,809**	4,809.00 ± 0.00	0.0000	5.98	**4,809**	4,809.00 ± 0.00	0.0000	6.08
	pmed19	**2,845**	**2,845**	2,845.00 ± 0.00	0.0000	7.04	**2,845**	2,845.00 ± 0.00	0.0000	6.20
	pmed20	**1,789**	**1,789**	1,789.00 ± 0.00	0.0000	9.81	**1,789**	1,789.00 ± 0.00	0.0000	8.62
	pmed21	**9,138**	**9,138**	9,138.00 ± 0.00	0.0000	53.37	**9,138**	9,138.00 ± 0.00	0.0000	54.59
	pmed22	**8,579**	**8,579**	8,579.00 ± 0.00	0.0000	43.30	**8,579**	8,579.00 ± 0.00	0.0000	43.66
	pmed23	**4,619**	**4,619**	4,619.00 ± 0.00	0.0000	10.31	**4,619**	4,619.00 ± 0.00	0.0000	8.73
	pmed24	**2,961**	**2,961**	2,961.00 ± 0.00	0.0000	10.56	**2,961**	2,961.00 ± 0.00	0.0000	9.26
	pmed25	**1,828**	**1,828**	1,828.00 ± 0.00	0.0000	15.26	**1,828**	1,828.00 ± 0.00	0.0000	13.98
	pmed26	**9,917**	**9,917**	9,917.00 ± 0.00	0.0000	108.31	**9,917**	9,917.00 ± 0.00	0.0000	93.36
	pmed27	**8,307**	**8,307**	8,307.00 ± 0.00	0.0000	95.41	**8,307**	8,307.00 ± 0.00	0.0000	91.13
	pmed28	**4,498**	**4,498**	4,498.00 ± 0.00	0.0000	12.46	**4,498**	4,498.00 ± 0.00	0.0000	11.74
	pmed29	**3,033**	**3,033**	3,033.00 ± 0.00	0.0000	15.47	**3,033**	3,033.00 ± 0.00	0.0000	13.74

Table 2: (Continued)

			AD-AHSAR				AD-GHSAR			
Instance	ID	OPT	Best	Avg $\pm SD$	$\%err$	Time (s)	Best	Avg $\pm SD$	$\%err$	Time (s)
	pmed30	**1,989**	**1,989**	1,989.00 ± 0.00	0.0000	24.03	**1,989**	1,989.00 ± 0.00	0.0000	21.40
	pmed31	**10,086**	**10,086**	10,086.00 ± 0.00	0.0000	172.51	**10,086**	10,086.00 ± 0.00	0.0000	166.23
	pmed32	**9,297**	**9,297**	9,297.00 ± 0.00	0.0000	142.65	**9,297**	9,297.00 ± 0.00	0.0000	127.29
	pmed33	**4,700**	**4,700**	4,700.00 ± 0.00	0.0000	21.17	**4,700**	4,700.00 ± 0.00	0.0000	19.55
	pmed34	**3,013**	**3,013**	3,013.00 ± 0.00	0.0000	23.47	**3,013**	3,013.00 ± 0.00	0.0000	20.77
	pmed35	**10,400**	**10,400**	10,400.00 ± 0.00	0.0000	187.99	**10,400**	10,400.00 ± 0.00	0.0000	196.01
	pmed36	**9,934**	**9,934**	9,934.00 ± 0.00	0.0000	184.53	**9,934**	9,934.00 ± 0.00	0.0000	178.57
	pmed37	**5,057**	**5,057**	5,057.00 ± 0.00	0.0000	25.75	**5,057**	5,057.00 ± 0.00	0.0000	27.90
	pmed38	**11,060**	**11,060**	11,060.00 ± 0.00	0.0000	189.00	**11,060**	11,060.00 ± 0.00	0.0000	200.00
	pmed39	**9,423**	**9,423**	9,423.00 ± 0.00	0.0000	193.03	**9,423**	9,423.00 ± 0.00	0.0000	198.63
	pmed40	**5,128**	**5,128**	5,128.00 ± 0.00	0.0000	34.72	**5,128**	5,128.00 ± 0.00	0.0000	32.11
RW	$p = 100$	**5,210**	**5,210**	5,223.00 ± 6.89	0.2495	108.06	**5,210**	5,224.25 ± 9.25	0.2735	100.74
	$p = 200$	**2,704**	**2,704**	2,707.95 ± 3.38	0.1461	65.36	**2,704**	2,706.90 ± 3.21	0.1072	65.34
	$p = 300$	**2,018**	**2,018**	2,018.10 ± 0.30	0.0050	79.20	**2,018**	2,018.20 ± 0.40	0.0099	80.05
	$p = 400$	**1,734**	**1,734**	1,734.00 ± 0.00	0.0000	94.54	**1,734**	1,734.00 ± 0.00	0.0000	94.05
	$p = 500$	**1,614**	**1,614**	1,614.00 ± 0.00	0.0000	97.96	**1,614**	1,614.00 ± 0.00	0.0000	98.77
TSPLIB	$p = 50$	**29,089.71**	29,090.22	29,090.22 ± 0.00	0.0018	97.96	29,090.22	29,090.22 ± 0.00	0.0018	98.77
	$p = 100$	**16,552.35**	**16,552.35**	16,554.17 ± 3.00	0.0179	80.80	**16,552.35**	16,554.15 ± 2.99	0.0178	92.15
	$p = 150$	**12,026.41**	12,026.42	12,026.53 ± 0.39	0.0010	101.65	**12,026.41**	12,026.83 ± 0.93	0.0035	104.24
	$p = 200$	9,356.66	9,356.64	9,358.23 ± 1.11	0.0168	97.03	**9,356.62**	9,358.16 ± 1.18	0.0160	87.36
	$p = 250$	7,737.72	**7,737.20**	7,740.04 ± 1.59	0.0300	109.03	7,737.73	7,740.56 ± 1.33	0.0367	104.13
	$p = 300$	**6,617.27**	6,618.11	6,620.96 ± 1.46	0.0557	116.19	6,618.60	6,621.71 ± 1.81	0.0671	113.34
	$p = 350$	5,719.30	**5,719.03**	5,722.48 ± 1.84	0.0556	142.27	5,720.54	5,724.86 ± 3.07	0.0972	128.30
	$p = 400$	**5,006.75**	5,006.77	5,014.81 ± 3.48	0.1610	146.77	5,009.20	5,015.79 ± 3.57	0.1806	153.53
	$p = 450$	4,473.36	**4,471.57**	4,476.19 ± 3.18	0.0632	186.66	4,471.58	4,476.20 ± 2.92	0.0635	191.93
	$p = 500$	4,046.86	**4,046.39**	4,048.52 ± 1.10	0.0595	196.17	4,046.45	4,048.56 ± 0.99	0.0606	194.25

the art algorithm. Columns 4–7 and Columns 8–11 present the best solution objective value (denoted as best), the average solution objective value plus or minus the standard deviation (denoted as avg $\pm$ SD), the average percentage error rate (denoted as $\%err$), and the average time elapsed for AD-GHSAR and AD-AHSAR, respectively. Among these measurements, $\%err$ is defined following Hansen and Mladenović (1997):

$$\%err = \frac{v_{\text{avg}} - v_{\text{opt}}}{v_{\text{opt}}} \times 100, \tag{12}$$

where v_{avg} and v_{opt} indicate the average solution objective value and the best known upper bound, respectively.

From Table 2, the following observations can be drawn. Over the ORLIB instances, both AD-AHSAR and AD-GHSAR are always able to achieve optimal solutions ($\%err = 0$ for all the ORLIB instances). This observation demonstrates the effectiveness of our algorithms. Meanwhile, it confirms the fact that the ORLIB instances are relatively easy to solve (see Section 6.1). On the other hand, over larger scale instances, the proposed algorithms are also able to obtain competitive results. For example, over RW instances, AD-AHSAR and AD-GHSAR are also able to achieve the best known upper bounds for all the instances, but with the average percentage error ranging from 0 to 0.2735%. Over TSPLIB instances, which have the largest search spaces, our algorithms achieve five new best known upper bounds. However, due to the problem hardness, our algorithms are not able to perform equally well over all of these instances. Taking a more detailed look, AD-AHSAR outperforms PBS over five instances, obtains one currently best known upper bound, and gets outperformed by PBS over four instances. For AD-GHSAR, similar observations can be drawn. Over all the 10 TSPLIB instances, AD-GHSAR outperforms PBS over three instances, achieves two currently best known upper bounds, and there are five instances over which AD-GHSAR is outperformed by PBS. For all the TSPLIB instances, the average percentage errors of AD-AHSAR and AD-GHSAR are always less than 0.2%.

When we compare the performance of AD-AHSAR and AD-GHSAR, we observe that these two algorithms have similar performance over most of the benchmark instances. As for the best solutions obtained by the two algorithms, over all the instances, there are six instances over which AD-AHSAR outperforms AD-GHSAR, and two instances over which AD-GHSAR performs better. In addition, we observe that both algorithms are stable, and have small standard deviations.

As a brief summary, in this section, we have presented the numerical results obtained by AD-AHSAR and AD-GHSAR. The results demonstrate that both algorithms are able to achieve competitive solutions, which are comparable to the state of the art results. In the following section, we investigate the underlying reasons for the encouraging performance.

6.3 Effectiveness Evaluation

In this section, we examine the effectiveness of the proposed framework from various aspects, so as to investigate the reasons for the competitive performance. In conducting the experiments, we intend to answer the questions Q1, Q2, and Q3.

6.3.1 Q1: Is the Design of the LLP Adaptation Reasonable and Beneficial?

This question investigates the two hypotheses raised in Section 3.1, that is, the effectiveness results from the learning capability of the ant model rather than the random

selection of the LLP values; and the ant-based LLP adaptation is able to select effective LLP values with respect to different LLH transitions. To answer this question, two sets of algorithms are introduced for comparison. First, we look at the hyper-heuristics with randomly selected LLPs (denoted as R-AH and R-GH). These two algorithms are similar to AD-AH and AD-GH, except that at each decision point of the parameterized LLHs (see lines 10–11 of Algorithm 1), the corresponding LLPs are randomly selected within the range of the corresponding LLPs, rather than dynamically maintained by the adaptation algorithm. Second, we consider two variants of AD-AH and AD-GH, in which a one-dimensional archive matrix is employed (denoted as AD-AH-1D and AD-GH-1D; see Section 3.1).

6.3.2 Q2: Is Each Mechanism of the Framework Useful?

By this question, we intend to examine the effectiveness of the LLP adaptation mechanism and the SAR mechanism. More specifically, the comparisons are conducted between the hyper-heuristics with and without each mechanism. To answer this question, the hyper-heuristics with static LLP configurations (denoted as AHSAR, GHSAR, AH, and GH) are introduced for comparison. These algorithms can be viewed as special cases of their counterparts with adaptive LLPs. For example, if we assign the lower bound and the upper bound of each LLP to be the same in AD-AH, we obtain AH. In particular, we first compare AD-AH (AD-GH) with AH (GH), so as to evaluate the effects of the LLP adaptation. Then, we compare AHSAR (GHSAR) and AH (GH), in order to investigate the impact of the SAR mechanism.

6.3.3 Q3: Is the Combination of the Two Mechanisms Necessary and Effective?

By this question, we intend to investigate whether the combination of the two mechanisms is necessary. To answer this question, we compare AD-AHSAR (AD-GHSAR) with two baseline algorithms, that is, AHSAR (GHSAR) and AD-AH (AD-GH). Each baseline algorithm differs from AD-AHSAR (AD-GHSAR) in only one mechanism. The comparisons are conducted in such a way as to investigate the influence of each mechanism on the whole framework.

To investigate these questions, the rest of Section 6.3 is organized as follows. First, we give the background of the experiments. After that, a series of comparisons is conducted, and statistical tests are presented to justify the decisions we make. Based on the comparison results, detailed analysis and discussion are presented, in order to answer the questions above.

6.3.4 Experimental Setup

In this section we introduce the background of the experiments.

6.3.4.1 Performance Measurement and Statistical Tests

When comparing the performance of the algorithms in this section, we concentrate on the effectiveness of the framework. More precisely, for all the comparisons in this section, the performance of each algorithm over a given instance is measured by the average percentage error rate of 20 independent runs (see Equation (12) in Section 6.2). Furthermore, in order to compare the performance of two algorithms over a given set of instances, and draw confident conclusions, statistical tests are conducted to judge which algorithm outperforms the other. Moreover, as stated in García et al. (2009), to investigate the performance of optimization heuristics, nonparametric tests are preferable to

their parametric counterparts. The reason is that parametric tests are usually based on strong assumptions that may not hold for the results obtained by heuristic algorithms. Following Hutter et al. (2009), we employ the two-sided Wilcoxon signed rank test to detect the potential differences between algorithms. In the tests, the null hypothesis states that both algorithms in comparison have similar performance, and we consider the 95% confidence level (i.e., the p-values below .05 are treated to be statistically significant), unless otherwise stated.

6.3.4.2 Test Instances and LLP Tuning

To conduct the statistical tests, a set of benchmark instances has to be specified. In this section, the selection of the test instances falls into two categories. If neither of the algorithms for comparison involves static LLP configurations, the test is conducted over all 55 instances. However, if either of the algorithms for comparison uses static LLP configurations, these LLPs should first be tuned with an offline tuning methodology. As required by the tuning task, the benchmark instance set has to be separated into two disjoint sets, that is the training set and the test set. After the tuning of the LLPs over the training instances, the performance of the algorithms is tested over the test instances.

In order to investigate the generality and the quality of the LLP adaptation, the hyper-heuristics with static LLP configurations are employed as the baselines. The comparisons with these baseline algorithms are conducted in two scenarios, to simulate different situations in which the LLPs are tuned. More specifically, the first scenario simulates the situation in which not all the instance distributions are known a priori. This scenario is introduced to test the generality of the LLP adaptation mechanism. Since there are three heterogeneous classes of instances in this study, this scenario is further divided into three sub scenarios, depending on the training instances (1.1 for TSPLIB, 1.2 for OR, and 1.3 for RW). For example, in Scenario 1.1, the training instances are selected from the TSPLIB instances, and the rest of the instances comprise the test set. Scenario 1 seems to be unfair for the offline tuning methodologies, in that a strong bias is posed against them. However, in the context of cross domain problem solving as in Hyflex (Burke, Curtois, Hyde et al., 2009; Burke, Curtois, Kendall et al., 2009), or in unseen instance solving within a single domain as in this paper, chances are that this scenario may truly exist. On the other hand, Scenario 2 is introduced to examine the quality of the LLP adaptation. In this scenario, the training set covers all three instance sets, so as to simulate the situation in which the practitioners have knowledge about all the instance distributions.

In Scenarios 1 and 2, the whole benchmark instance set is separated into two disjoint sets: the training set and the test set. In this study, the number of training instances is set to be five, taking into account that there are five RW instances. For each scenario, the training instances are randomly selected.[3] As a result, the test set in each scenario consists of 50 instances.

For the offline tuning methodology, we employ the iterated F-Race (Birattari et al., 2010). In particular, we employ irace[4] (López-Ibáñez et al., 2011) with limited execution time for each algorithm, that is, we restrict each algorithm to be performed for no longer

[3]Note that in the same scenario, the training instances are the same for all algorithms to be tuned.

[4]irace is an R implementation of iterated F-Race, available at http://iridia.ulb.ac.be/irace/

Table 3: Pre-tuned LLP configurations.

	Parameter	AHSAR	GHSAR	AH	GH
Scenario 1.1 (TSPLIB)	shake-strength	0.5810	0.5038	0.1982	0.3637
	mutation-rate	0.4076	0.7033	0.4230	0.2861
	LK-rate	0.4397	0.5072	0.4615	0.4309
Scenario 1.2 (ORLIB)	shake-strength	0.3114	0.3033	0.2255	0.1573
	mutation-rate	0.4001	0.1001	0.2745	0.7204
	LK-rate	0.1125	0.1045	0.1987	0.1492
Scenario 1.3 (RW)	shake-strength	0.3121	0.3325	0.7113	0.2240
	mutation-rate	0.1998	0.2788	0.1884	0.3744
	LK-rate	0.2012	0.1472	0.2640	0.3265
Scenario 2 (ALL)	shake-strength	0.4008	0.5000	0.2611	0.5101
	mutation-rate	0.8129	0.3127	0.3665	0.3012
	LK-rate	0.6123	0.5009	0.5396	0.4989

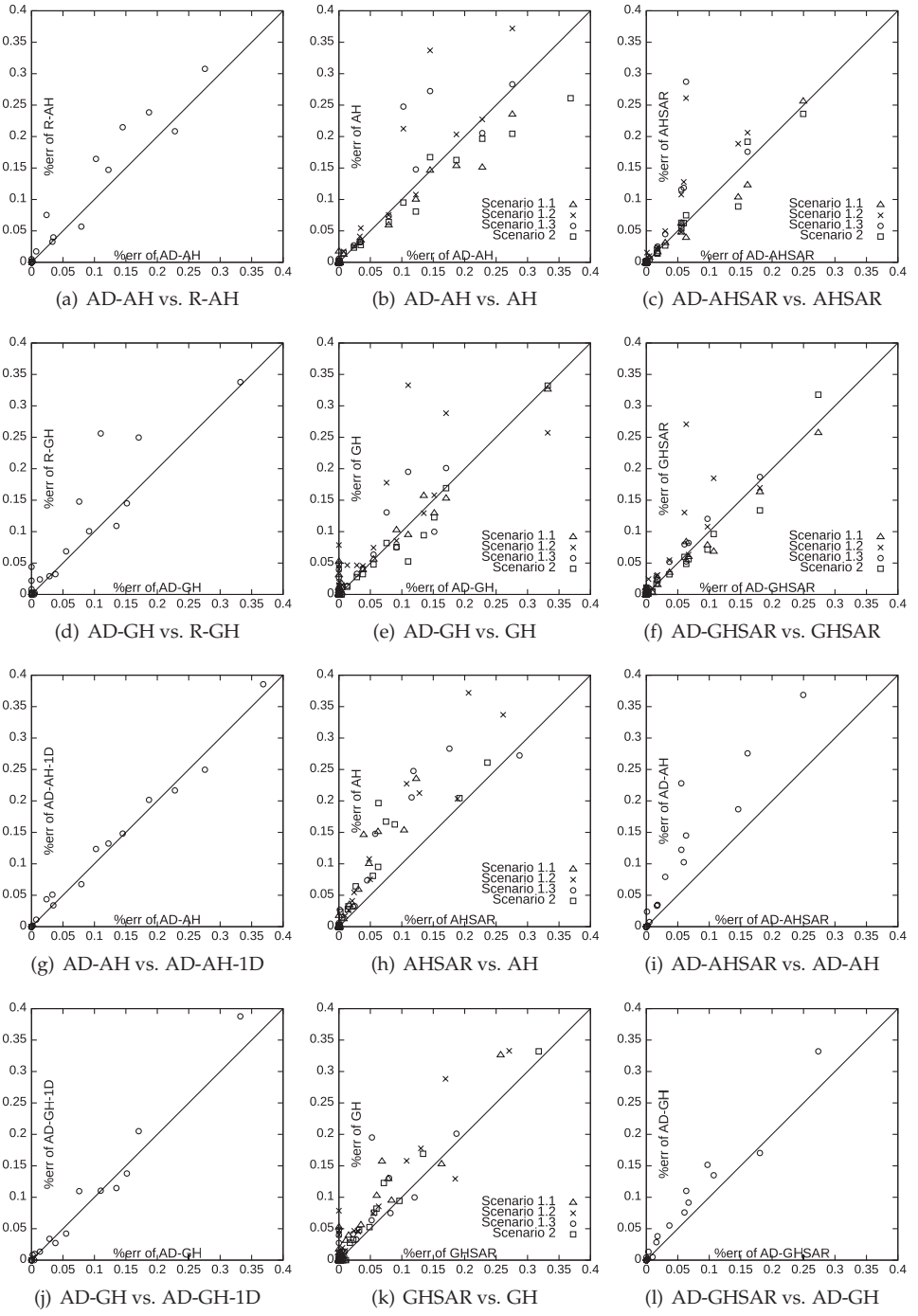
than 60 s, so that the training time is acceptable.[5] For the LLPs to be tuned, there are the mutation-rate from the mutation operator, the shake-strength from the shake operator, and the LK-depth from the LK(k) operator. Among these LLPs, the mutation-rate is a real-valued parameter within $[0.1, 0.9]$, and the other two LLPs are integer parameters within $[1, p]$. However, since the range of the latter two LLPs are dependent on the number of medians p, during the LLP tuning, we introduce two real-valued parameters, shake-rate and LK-rate, both of which range within $[0.1, 0.9]$. After the pretuning task, the actual LLPs used are $\lceil \text{shake-rate} \times p \rceil$ and $\lceil$ LK-rate $\times p \rceil$.

The tuning results are summarized in Table 3, which is organized as follows. Column 1 indicates the scenarios described above. Column 2 represents the LLP names. Then columns 3–6 present the LLP configurations for each algorithm. From Table 3, the following observations can be drawn. First, for the same algorithm, the LLP values vary greatly as the scenario changes. For example, in Scenario 1.2, the LK-rate of GHSAR is 0.1045, while in Scenario 2, the value for the same LLP is 0.5009. This observation implies that the tuning procedure might be dependent on the training instances. Second, in a single scenario, the values of the LLPs also vary greatly between different algorithms. For example, in Scenario 2, AHSAR prefers larger values of mutation-rate (0.8129), while GH prefers a much smaller mutation-rate (0.3012). This observation indicates that the static setup of LLPs across different algorithms (e.g., setting the LLP values from the literature) may not be appropriate, in that for the same LLP, different values may be preferred when applied in different algorithms.

6.3.5 Comparison Results and Discussion

After introducing the background information of the experiments, we now conduct comparisons between various algorithms, so as to evaluate the performance of the framework. In order to gain an intuitive understanding of the relative comparison between the algorithms, in Figure 3, we visually present the average performance of the algorithms in each comparison. The figure is organized as follows. Figures 3(a), 3(d),

[5] Although with such a restriction, the tuning is still very time-consuming. For example, it takes more than 20 hr for the tuning of AHSAR in Scenario 1.1.



(a) AD-AH vs. R-AH (b) AD-AH vs. AH (c) AD-AHSAR vs. AHSAR

(d) AD-GH vs. R-GH (e) AD-GH vs. GH (f) AD-GHSAR vs. GHSAR

(g) AD-AH vs. AD-AH-1D (h) AHSAR vs. AH (i) AD-AHSAR vs. AD-AH

(j) AD-GH vs. AD-GH-1D (k) GHSAR vs. GH (l) AD-GHSAR vs. AD-GH

Figure 3: Comparison of the %*err* of each algorithm in different LLP tuning scenarios.

3(g), and 3(j), on the left side of Figure 3, illustrate the results for the comparisons that correspond to Q1. The middle of Figure 3, Figures 3(b), 3(e), 3(h), and 3(k), illlustrate the results for Q2. The right side of Figure 3, Figures 3(c), 3(f), 3(i), and 3(l), illustrate the results for Q3. Taking Figure 3(b) as an example, we describe the content of each figure part. Figure 3(b) illustrates the comparison between AD-AH and AH. In Figure 3(b) the x axis and the y axis indicate the $\%err$ of AD-AH and AH over the test instances, respectively. More specifically, each point (x, y) in Figure 3(b) indicates that there are one or more instances over which AD-AH's $\%err$ and AH's $\%err$ are x and y, respectively. For clarity, we also plot the reference line $y = x$. Consequently, a point above the line implies that over the corresponding instance(s), AD-AH outperforms AH, in that AD-AH is able to obtain a smaller $\%err$. For those comparisons in which static LLP configurations are involved, the comparisons in both Scenario 1 and Scenario 2 are presented, with different point types. Note that most of the plots in Figure 3 are relatively sparse. The reason is that each point may represent multiple instances. For example, since the ORLIB instances are relatively easy, the origin (0, 0) represents the comparisons on multiple instances.

The companion table to Figure 3 is Table 4. Table 4 presents the results of the statistical tests, which are organized as follows. The first column indicates the three questions, Q1, Q2, and Q3, as described above. The second column presents all the comparisons. Then in columns 3–6, the results of the Wilcoxon tests are reported for each scenario. The comparison results consist of the p-value, as well as the name of the algorithm that performs better in the comparison (in Table 4, where the difference is not significant, we represent this by (—)).

After presenting and describing the comparison results, we now analyze these results. For each group of comparisons, we first describe the phenomenon observed from Figure 3. Then, the results of the relative comparisons are presented, along with the conclusion of the Wilcoxon test. Finally, the potential reasons for the observations are discussed.

6.3.5.1 Investigation of Q1

To answer Q1, we investigate the two hypotheses raised in Section 3.1, that is, the ant-based LLP adaptation is able to learn appropriate LLP values, rather than randomly selecting them; and the ant-based LLP adaptation is able to select effective LLP values with respect to different LLH transitions.

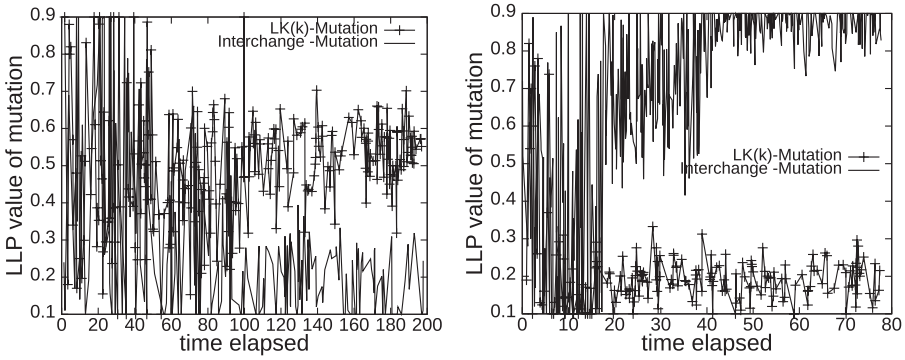
From Figure 3(a), we observe that most points lie above the reference line, which implies that AD-AH may perform better than R-AH. This observation is confirmed by the Wilcoxon test, which indicates that AD-AH outperforms R-AH with a 95% confidence level (p-value = .0171). This means that the effectiveness of the LLP adaptation does not result from the random selection of the LLP values. On the contrary the ant model is able to intelligently adapt the LLP configurations. This observation also partially confirms the assumption that the ant-based adaptation has a learning capability suitable for our framework. Next, we examine the second hypothesis. From Figure 3(g), we observe that most points lie above, yet close to the reference line, which implies that AD-AH performs better than AD-AH-1D, but the difference may not be significant. The result of the Wilcoxon test shows that the null hypothesis cannot be rejected (p-value = .2611), which means that the two algorithms perform similarly. Meanwhile, a similar observation can be drawn for AD-GH. For example, AD-GH outperforms R-GH (p-value = .0347), and performs similarly to AD-GH-1D (p-value = .3575).

Table 4: Performance comparison using Wilcoxon test (all values are p-values).

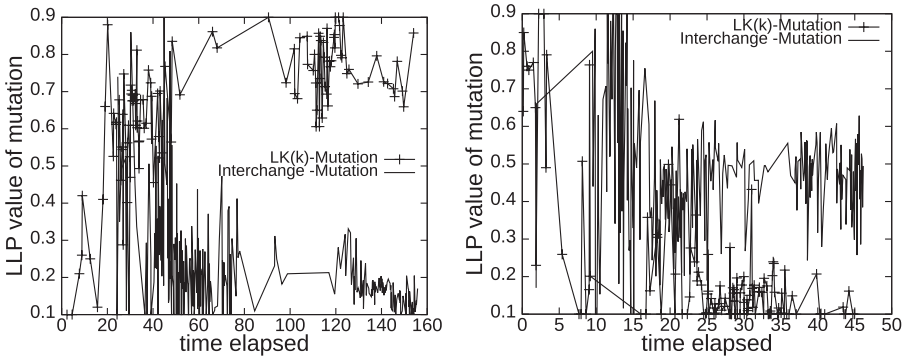
	Comparison	Scenario 1.1 (TSPLIB)	Scenario 1.2 (ORLIB)	Scenario 1.3 (RW)	Scenario 2 (ALL)
Q1	AD-AH vs. R-AH		.0171 (AD-AH)[a]		
	AD-GH vs. R-GH		.0347 (AD-GH)		
	AD-AH vs. AD-AH-1D		.2661 (—)[b]		
	AD-GH vs. AD-AH-1D		.3575 (—)		
Q2	AD-AH vs. AH	.5693 (—)	.0097 (AD-AH)	.1475 (—)	.0479 (AH)
	AD-GH vs. GH	.2775 (—)	.003 (AD-GH)	.0106 (AD-GH)	.0522 (GH)
	AHSAR vs. AH	.0005 (AHSAR)	.0001 (AHSAR)	.0098 (AHSAR)	.0002 (AHSAR)
	GHSAR vs. GH	.0038 (GHSAR)	.0015 (GHSAR)	.0057 (GHSAR)	.0179 (GHSAR)
Q3	AD-AHSAR vs. AHSAR	0.3008 (—)	.0017 (AD-AHSAR)	.0039 (AD-AHSAR)	.8311 (—)
	AD-GHSAR vs. GHSAR	0.4548 (—)	.0021 (AD-GHSAR)	.0137 (AD-GHSAR)	.4630 (—)
	AD-AHSAR vs. AD-AH		.0005 (AD-AHSAR)		
	AD-GHSAR vs. AD-GH		.0034 (AD-GHSAR)		

[a]Note that for each comparison, if neither of the algorithms in the comparison has a static LLP configuration (e.g., AD-AH vs. R-AH), the test is reported only once. Otherwise, the corresponding comparison is conducted in Scenarios 1–2, and in each scenario there are 50 test instances, as discussed in Section 6.3.1.

[b]Note that for comparisons where the difference is not significant, we denote this by (—). Where an algorithm performs significantly better, that algorithm is given in parentheses.



(a) AD-AH, fl1400 with $p = 500$ (b) AD-AH, rw1000 with $p = 200$



(c) AD-GH, fl1400 with $p = 500$ (d) AD-GH, rw1000 with $p = 200$

Figure 4: Examples of the *mutation-rate* in different LLH transitions.

One possible reason for this observation might be that there are relatively few parameterized LLPs in this study (mutation-rate, shake-strength, and LK-depth), thus the dependencies between the LLP values and the LLH transitions may not be strong enough to tell the difference between the two variants. We suppose that the difference between the two variants might be more significant if there were more parameterized LLPs, which deserves future work.

Although the statistical tests did not find significant differences between AD-AH (AD-GH) and AD-AH-1D (AD-GH-1D), during the experiments, we discover that AD-AH (AD-GH) is able to adapt different LLP values with respect to different LLH transitions. For instance, in Figure 4, we present the tendency for the mutation-rate value against time. The mutation-rate is adapted by AD-AH (AD-GH) over two benchmark instances (fl1400 with $p = 500$, and rw1000 with $p = 200$),[6] in a typical execution of the algorithm. In each part of Figure 4 the two lines represent the mutation-rate's value, when applied after LK(k) and interchange. We observe that AD-AH (AD-GH) prefers a different mutation-rate in different LLH transitions. This observation to some extent supports the hypothesis that the two-dimensional archive matrix design is reasonable, and more expressive. As a result, since AD-AH (AD-GH) is able to select different LLP

[6]The reason we do not choose the ORLIB instances is that these instances are relatively easy, thus the algorithms may converge too quickly.

configurations with respect to different LLH transitions, and the performance is better (though not significantly) than AD-AH-1D (AD-GH-1D), we adopt this design in our implementation.

6.3.5.2 Answer to Q1

By examining the hypotheses raised in Section 3.1, we confirm that the LLP adaptation is capable of learning effective LLP configurations, and partially validate that the implementation choice of the ant model is reasonable. Thus in the following tests, we focus on the framework, so as to investigate the functionalities of each mechanism, as well as the effectiveness of the combination of the two mechanisms.

6.3.5.3 Investigation of Q2

Now we examine the influence of each mechanism of the framework. We first compare AD-AH (AD-GH) and AH (GH) to evaluate the effectiveness of the LLP adaptation. Then, AHSAR (GHSAR) is compared with AH (GH) to investigate the impact of the SAR mechanism.

From Figure 3(b), we observe that the results are dependent on the scenario in which the comparisons are conducted. For example, in Scenario 2, most points (denoted as squares) lie below the reference line, which implies that AD-AH is outperformed by AH (p-value = .0479) in this scenario. Meanwhile, in Scenarios 1.1 and 1.3, the points (denoted as triangles and circles) lie around the reference line, which means that AD-AH and AH have similar performance in these scenarios (p-value = .5693 and .1475, respectively). In any case, in Scenario 1.2, most points (denoted as crosses) lie above the reference line, indicating that AD-AH outperforms AH in this scenario (p-value = .0097).

These observations may imply that if all the instance distributions are known a priori (as in Scenario 2), the tuning tool (irace in this study) is able to generate very promising LLP configurations. However, if the knowledge of the instance distribution is not provided, irace tends to be instance dependent. For example, in Scenario 1.2, the tuning is conducted over ORLIB instances, and the performance of AH is statistically worse than AD-AH. The reason may be that in this study, the hardness of the instances varies greatly among different instance sets. Of all the three instance sets, ORLIB instances are relatively easy, thus we cannot tell the difference between different LLP configurations. As a result, the LLPs provided by irace in this scenario may not perform well over the test instances. On the contrary, AD-AH does not require knowledge of the instance distribution, and tends to be more stable compared with AH. Similarly, when we compare AD-GH and GH, we observe that AD-GH is outperformed by GH in Scenario 2 (p-value = .0522, which means that the confidence level for this comparison is 90%), performs similarly with GH in Scenarios 1.1 (p-value = .2755), and outperforms GH in scenarios 1.2 and 1.3 (p-value = .003 and .0106, respectively).

As a brief summary, when comparing the performance of AD-AH (AD-GH) and AH (GH), the results demonstrate that the offline tuning of the LLPs tends to be sensitive to the training instances. If the training instances capture the distributions of all the benchmark instances, AH (GH) outperforms AD-AH (AD-GH). We attribute this observation to the fact that the adaptation of the LLPs expands the scale of the search space, which makes it more difficult for AD-AH (AD-GH) to achieve promising LLP configurations and high quality solutions simultaneously. On the other hand, AD-AH (AD-GH) is able to obtain better performance if this precondition does not hold.

Next, we investigate the SAR mechanism. In order to test the effectiveness of this mechanism, comparisons are conducted between AHSAR (GHSAR) and AH (GH). From Figure 3(h), we observe that most points lie above the reference line, regardless of the training scenarios. Meanwhile, from Table 4, we see that in both Scenarios 1 and 2, AHSAR consistently outperforms AH (with p-value $< .01$). When we compare GHSAR and GH, we observe that GHSAR also outperforms GH in both Scenarios 1 and 2 (p-value $< .02$). These observations demonstrate the effectiveness of the reduction mechanism. However, since the LLPs are statically assigned in AHSAR (GHSAR), one potential risk is that AHSAR (GHSAR) is also instance dependent. Taking Figure 3(h), for instance, in Scenario 1.1, the maximum $\%err$ of AHSAR is less than 0.15%, while in Scenario 1.3, the maximum $\%err$ of AHSAR is around 0.3%.

6.3.5.4 Answer to Q2

The experiments in this group give positive answers to Q2. Both the LLP adaptation and the SAR mechanism are feasible and effective. However, both mechanisms have drawbacks. On one hand, in AD-AH (AD-GH), the introduction of the extra optimization variables expands the search space. On the other hand, in AHSAR (GHSAR), the LLPs are statically assigned, which may lead to the instance dependent problem. In the following experiments, we investigate the combination of the two mechanisms.

6.3.5.5 Investigation of Q3

Now we investigate the influence of the combination of the LLP adaptation and the SAR mechanism. To answer Q3, AD-AHSAR (AD-GHSAR) is compared with two baseline algorithms, that is, AHSAR (GHSAR) and AD-AH (AD-GH). Each baseline algorithm differs from AD-AHSAR (AD-GHSAR) in only one mechanism. For these two baseline algorithms, AHSAR (GHSAR) is selected to test the influence of the LLP adaptation on the whole framework, and AD-AH (AD-GH) is selected to test whether the SAR mechanism is able to prevent the search space from drastic expansion, after the LLP adaptation has introduced extra variables to be optimized.

First, we compare AD-AHSAR with AHSAR. As predicted, AHSAR is instance dependent, which is similar to what has been observed. From Figure 3(c), we observe that most points lie either around or above the reference line. More specifically, in Scenarios 1.2 and 1.3, AD-AHSAR outperforms AHSAR (p-value $< .004$), while in Scenarios 1.1 and 2, AD-AHSAR performs similarly to AHSAR (p-value $= .3008$ and .8311, respectively). Next, we compare AD-AHSAR with AD-AH. As shown in Figure 3(i), it is obvious that AD-AHSAR outperforms AD-AH (p-value $= .0005$). Similar observations can be drawn when we compare AD-GHSAR with GHSAR and AD-GH, that is AD-GHSAR performs at least as well as GHSAR, and statistically outperforms AD-GH.

6.3.5.6 Answer to Q3

The experiments give a positive answer to Q3, that the combination of the two mechanisms is useful and beneficial. Through the combination, the proposed framework benefits from both mechanisms, meanwhile partially avoiding their drawbacks.

In conclusion, in this section, extensive experiments were carried out, so as to evaluate the effectiveness of the LLP adaptation, the SAR mechanism, as well as their combination as a whole framework. Through the experiments, we demonstrated that

both the LLP adaptation and the SAR mechanism are effective. Furthermore, the combination of these two mechanisms contributes greatly to the competitive results.

6.4 Efficiency Evaluation

In this section, we intend to examine the runtime behaviors of the framework, so as to analyze the dynamic properties of the LLP adaptation, the SAR mechanism, as well as their combination. Following Hoos and Stützle (2005), the run time distribution (RTD) is usually employed to capture various properties of heuristic algorithms, such as the convergence speed, the average solution quality, and so on. More specifically, the comparisons are carried out by running each algorithm over typical benchmark instances for multiple times, and examining the cumulative completions that the algorithm achieves a certain quality threshold as time elapses. In this section, the algorithms for comparison include AD-AH (AD-GH), AD-AHSAR (AD-GHSAR), AH (GH), and AHSAR (GHSAR), so as to concentrate on the mechanisms within the framework.

The RTD analysis is conducted over two typical instances, that is fl1400 with $p = 500$ and rw1000 with $p = 200$, from the TSPLIB and the RW instance set, respectively. The reason we do not choose the ORLIB instances is that these instances are relatively easy, and thus cannot be used to distinguish the runtime behaviors of the algorithms, which is similar to what was found in Section 6.3.2. Over the two instances, each algorithm is executed for 100 independent trials. For each algorithm, we set the cutoff time to be 200 s, and eliminate the maximum iteration stopping criterion. For those hyper-heuristics with static LLP configurations, the LLPs are set with respect to Scenario 2 (see Table 3).

For each algorithm, its runtime behavior is represented by a RTD curve determined from the 100 runs of the algorithm. In each part of Figure 5, the RTD curve of each algorithm is presented as follows. The x axis indicates the log-scale time, and the y axis represents the cumulative probability (denoted as P_{RTD}) that the algorithm achieves the predefined solution quality threshold. In this study, the threshold is set to be 0.1% over the best known upper bound, because during the experiments, we find this threshold is generally effective in distinguishing the performance of the algorithms.

From Figure 5, the following observations can be drawn. First, we observe that AH outperforms AD-AH in terms of both the convergence speed and the solution quality. For example, in Figure 5(a), the RTD curve of AH is always above that of AD-AH after 10 s. Besides, at the cutoff time, P_{RTD} of AH is 59%, while the corresponding probability of AD-AH is 36%. This observation again confirms the prediction that in the LLP adaptation mechanism, due to the expansion of the search space, it would take a longer time for the search process to converge, and the solution quality may not be quite satisfying.

When we compare the RTD curves of AHSAR and AH, we observe that AHSAR outperforms AH over both the TSPLIB instance (see Figure 5(a)) and the RW instance (see Figure 5(c)), which demonstrates the effectiveness and the efficiency of the SAR mechanism. The reason may be that by bipartitioning the heuristic space, the reduction mechanism explicitly considers the balance between the intensification and the diversification of the search process. As a result, AHSAR is able to achieve higher P_{RTD} compared with AH.

Finally, by comparing the RTD curves of AD-AHSAR and AHSAR, we examine the influence of the combination of the LLP adaptation and the SAR mechanism. In Figure 5(a), we observe that at the beginning of the search process, the RTD curve of

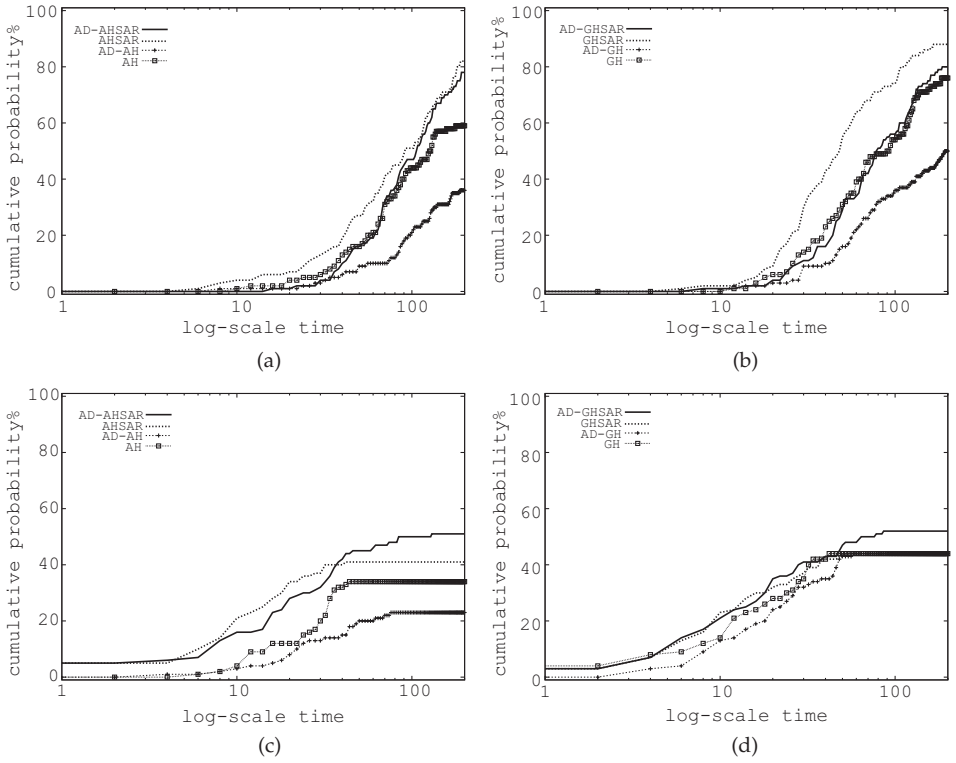


Figure 5: RTD curves of various hyper-heuristics over TSPLIB and RW instances (fl1400 with $p = 500$ and rw1000 with $p = 200$). The solution quality threshold over each instance is assigned with 0.1% over the best known upper bound. (a) AH and its variants, fl1400 with $p = 500$. (b) GH and it variants, fl1400 with $p = 500$. (c) AH and its variants, rw1000 with $p = 200$. (d) GH and its variants, rw1000 with $p = 200$).

AHSAR lies above that of AD-AHSAR. However, at the cutoff time, the cumulative probability P_{RTD} of AHSAR is similar to that of AD-AHSAR. Another interesting observation can be drawn from Figure 5(c). Over the RW instance, AHSAR converges after around 30 s, with $P_{\text{RTD}} = 41\%$. On the other hand, although AD-AHSAR converges slower, the final P_{RTD} reaches 51%.

Again, a similar observation can be drawn by GH and its variants, which is obvious from Figures 5(b) and 5(d). In conclusion, in this section, we evaluate the runtime behaviors of the framework, and the observations demonstrate the effects of each mechanism, as well as the impact of the combination of the two mechanisms.

7 Conclusion and Future Work

Our contributions in this study can be summarized as follows. First, to the best of our knowledge, this is the first study that considers the online adaptation of the LLP in a hyper-heuristic framework. We demonstrate that it is possible to embed a search-based algorithm (in this study an ant-based model) to adapt the LLPs, so as to alleviate the time-consuming and domain specific LLP tuning. Second, with the LLP adaptation,

we propose a general framework in which most of the existing hyper-heuristics can be embedded. Third, in order to prevent the search space from drastic expansion due to the adaptation of the LLPs, we apply a heuristic SAR mechanism to improve the search efficiency. Finally, we use the p-median problem as a case study, which is a new domain for hyper-heuristics. Extensive experiments demonstrate that the proposed framework is able to obtain encouraging results.

Despite the promising results, there are still several potential directions that deserve future research. (1) For the p-median problem, there are not many parameterized LLHs that can be extracted. As a result, some hypotheses (e.g., in the ant model, we assume the LLP selection should be conducted with respect to the previous LLH transition) in this study are not conclusively determined. In the future, we plan to test these hypotheses in the context of a larger LLP search space. (2) In this study, we do not investigate the influences of the HLS parameters. As discussed, some of these parameters may have an impact on the quality of the solutions. One potential research direction is to investigate whether the HLS parameters should be adaptively maintained or manually tuned. (3) In the SAR mechanism, we classify the LLHs into two subsets, which is similar to other approaches (Özcan et al., 2006; Meignan et al., 2010; Burke, Curtois, Kendall et al., 2009). However, this LLH division criterion may not be quite precise. Also, the division of the LLHs requires the understanding of the functionalities of the LLHs, which might be domain specific for certain problems. In future work, we intend to investigate whether there are more precise classification criteria, and whether those criteria can be learned in the exploration of the heuristic space.

Acknowledgments

We greatly thank our anonymous reviewers for their insightful comments and suggestions. This paper extends our previous study (Ren et al., 2010) presented at the 11th International Conference on Parallel Problem Solving from Nature. This work is partially supported by the National Natural Science Foundation of China under grants 61175062, 60805024, and 61033012, and the "Software + X" funding of Dalian University of Technology.

References

Aarts, E. H. L., and Lenstra, J. K. (1997). *Local search in combinatorial optimization*. New York: Wiley.

Beasley, J. E. (1985). A note on solving large p-median problems. *European Journal of Operational Research*, 21(2):270–273.

Bilchev, G., and Parmee, I. (1995). The ant colony metaphor for searching continuous design spaces. In T. Fogarty (Ed.), *Evolutionary computing*. *Lecture Notes in Computer Science*, Vol. 993 (pp. 25–39). Berlin: Springer.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-Race and iterated F-Race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.), *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Berlin: Springer.

Blum, C., and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

Box, G., and Muller, M. (1958). A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611.

Burke, E. K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., and Vázquez-Rodríguez, J. A. (2009). HyFlex: A flexible framework for the design and analysis of hyper-heuristics. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications*, pp. 790–797.

Burke, E. K., Curtois, T., Kendall, G., Hyde, M., Ochoa, G., and Vázquez-Rodríguez, J. A. (2009). Towards the decathlon challenge of search heuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pp. 2205–2208.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of metaheuristics*. *International Series in Operations Research & Management Science*, Vol. 146 (pp. 449–468). Berlin: Springer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2010). Hyper-heuristics: A survey of the state of the art. Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. R. (2010). A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942 –958.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. R. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1).

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*. *Intelligent Systems Reference Library*, Vol. 1 (pp. 177–201). Berlin: Springer.

Burke, E. K., Kendall, G., Landa Silva, D., O'Brien, R., and Soubeiga, E. (2005). An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 2263–2270.

Burke, E. K., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger (Eds.), *Handbook of metaheuristics*. *International Series in Operations Research & Management Science*, Vol. 57 (pp. 457–474). Berlin: Springer.

Burke, E. K., Kendall, G., and Soubeiga, E. (2003). A Tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470.

Chen, P.-C., Kendall, G., and Berghe, G. (2007). An ant based hyper-heuristic for the travelling tournament problem. In *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling (SCIS 2007)*, pp. 19 –26.

Correa, E., Steiner, M., Freitas, A., and Carnieri, C. (2001). A genetic algorithm for the p-median problem. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 1268–1275.

Cowling, P., Kendall, G., and Han, L. (2002). An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation 2002 (CEC 2002)*, pp. 1185–1190.

Cowling, P., Kendall, G., and Soubeiga, E. (2001a). A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*. *Lecture Notes in Computer Science*, Vol. 2079 (pp. 176–190). Berlin: Springer.

Cowling, P., Kendall, G., and Soubeiga, E. (2001b). A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference (MIC 2001)*, pp. 127–131.

Crawford, K. D., Hoelting, C. J., Wainwright, R. L., and Schoenefeld, D. A. (1997). A study of fixed length subset recombination. In *Proceedings of the Fourth Foundations of Genetic Algorithms Workshop*, pp. 365–378.

Cuesta-Cañada, A., Garrido, L., and Terashima-Marín, H. (2005). Building hyper-heuristics through ant colony optimization for the 2D bin packing problem. In *Knowledge-based intelligent information and engineering systems*, Vol. 3684 (pp. 907–907). Berlin: Springer.

DaCosta, L., Fialho, Á., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 2008 Conference on Genetic and Evolutionary Computation (GECCO '08)*, pp. 913–920.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):29–41.

Dowsland, K., Soubeiga, E., and Burke, E. (2007). A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759–774.

Eiben, A., Michalewicz, Z., Schoenauer, M., and Smith, J. (2007). Parameter control in evolutionary algorithms. In F. Lobo, C. Lima, and Z. Michalewicz (Eds.), *Parameter setting in evolutionary algorithms*. *Studies in Computational Intelligence*, Vol. 54 (pp. 19–46). Berlin: Springer.

Feo, T. A., and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61.

García, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 15(6):617–644.

Han, L., Kendall, G., and Cowling, P. (2002). An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002)*, pp. 267–271.

Hansen, N. (2006). An analysis of mutative σ-self-adaptation on linear fitness functions. *Evolutionary Computation*, 14(3):255–275.

Hansen, P., and Mladenović, N. (1997). Variable neighborhood search for the p-median. *Location Science*, 5(4):207–226.

Hansen, P., and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

Ho, N. B., and Tay, J. C. (2005). Evolving dispatching rules for solving the flexible job-shop problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pp. 2848–2855.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press.

Hoos, H. H., and Stützle, T. (2005). *Stochastic local search: Foundations and applications*. San Mateo, CA: Morgan Kaufmann.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306.

Kariv, O., and Hakimi, S. (1979). An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560.

Kendall, G., Soubeiga, E., and Cowling, P. (2002). Choice function and random hyperheuristics. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning, SEAL*.

Kochetov, Y., Levanova, T., Alekseeva, E., and Loresh, M. (2005). Large neighborhood local search for the *p*-median problem. *Yugoslav Journal of Operations Research*, 15(1):53–63.

Li, Y., and Li, W. (2007). Adaptive ant colony optimization algorithm based on information entropy: Foundation and application. *Fundamenta Informaticae*, 77(3):229–242.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Meignan, D., Koukam, A., and Créput, J.-C. (2010). Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346.

Meyer-Nieberg, S., and Beyer, H.-G. (2007). Self-adaptation in evolutionary algorithms. In F. G. Lobo, C. F. Lima, and Z. Michalewicz (Eds.), *Parameter setting in evolutionary algorithms. Studies in computational intelligence*, Vol. 54 (pp. 47–75). Berlin: Springer.

Ochoa, G., Qu, R., and Burke, E. K. (2009). Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '09)*, pp. 341–348.

Ong, Y.-S., Lim, M.-H., Zhu, N., and Wong, K.-W. (2006). Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):141–152.

Osman, I. H., and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):513–623.

Özcan, E., Bilgin, B., and Korkmaz, E. (2006). Hill climbers and mutational heuristics in hyperheuristics. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006). Lecture Notes in Computer Science*, Vol. 4193 (pp. 202–211). Berlin: Springer.

Özcan, E., Bilgin, B., and Korkmaz, E. (2008). A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23.

Pillay, N., and Banzhaf, W. (2009). A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research*, 197(2):482–491.

Poli, R., Woodward, J. R., and Burke, E. K. (2007). A histogram-matching approach to the evolution of bin-packing strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pp. 3500–3507.

Pullan, W. (2008). A population based hybrid metaheuristic for the *p*-median problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008)*, pp. 75–82.

Qu, R., and Burke, E. K. (2009). Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60:1273–1285.

Qu, R., Burke, E. K., and McCollum, B. (2009). Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384.

Ren, Z., Jiang, H., Xuan, J., and Luo, Z. (2010). Ant based hyper heuristics with space reduction: A case study of the *p*-median problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN 2010)*. *Lecture Notes in Computer Science*, Vol. 6238 (pp. 546–555). Berlin: Springer.

Resende, M. G. C., and Werneck, R. F. (2003). On the implementation of a swap-based local search procedure for the *p*-median problem. In *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments*, pp. 119–127.

Resende, M. G. C., and Werneck, R. F. (2004). A hybrid heuristic for the *p*-median problem. *Journal of Heuristics*, 10(1):59–88.

Ross, P. (2005). Hyper-heuristics. In E. K. Burke and G. Kendall (Eds.), *Search methodologies*, (pp. 529–556). Berlin: Springer.

Serpell, M. C., and Smith, J. E. (2010). Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, 18(3):491–514.

Socha, K. (2004). ACO for continuous and mixed-variable optimization. *Ant colony, optimization and swarm intelligence*. *Lecture Notes in Computer Science*, Vol. 3172 (pp. 53–61). Berlin: Springer.

Socha, K., and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173.

Taillard, É. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16.

Teitz, M. B., and Bart, P. (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961.

Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pp. 1539–1546.

Vazquez-Rodriguez, J., and Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793.

Whitaker, R. (1983). A fast algorithm for the greedy interchange for large-scale clustering and median location problems. *INFOR*, 21(2):95–108.