Hyper-Heuristics with Low Level Parameter Adaptation

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

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

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

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

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 with the domain independent motivation of hyper-heuristics. In this paper, we show how to automatically maintain the 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 et al., 2010a). 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 enough and cheap enough”) (Burke et al., 2003a), and (3) automate the process of algorithm design (Burke et al., 2010c). 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.

†Corresponding author.

©200X by the Massachusetts Institute of Technology Evolutionary Computation x(x): xxx-xxx
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, e.g., 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 that proposes and implements the domain independent strategies, the domain expert who provides the domain specific LLHs and knowledge, as well as the end user. For generality and flexibility considerations, the HLS designer and the domain expert do not collaborate directly. On the contrary, they communicate in an indirect way. On the 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 be extended to new problem domains.

Since the 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), etc. However, despite the great success of hyper-heuristics, there are still several difficulties in the process towards the automation of algorithm design. Among these difficulties, the parameterization of the LLHs poses great challenges to hyper-heuristics. The reason is that it is common that the LLHs are 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 the 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 et al., 2003a). 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 with (Cowling et al., 2001b), (Kendall et al., 2002), etc. We shall note that AD-HH differs from these parameter-free approaches from both the LLH and the HLS perspectives. On the 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 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). These, summing up the above discussion, are the main reasons why 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 the intensification and the diversification functionalities, respectively. 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, while retaining 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 are 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), 5 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 keep 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, i.e., 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 prevent 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 shall introduce the background about hyper-heuristics, as well as the existing parameter setting approaches.
2.1 Hyper-Heuristics

Burke et al. (2010a) define a hyper-heuristic as “an automated methodology for selecting or generating heuristics to solve hard computational search problems”. In the same paper, hyper-heuristics are classified into two broad categories: the heuristic selection approaches and the heuristic generation approach. The classification is based on the nature of the heuristic space, i.e., 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 et al. (2003b) propose a Tabu Search (TS) based hyper-heuristic for the timetabling problem and the rostering problem. Cuesta-Cañada et al. (2005) apply an ant based algorithm to guide the LLH selection, in the domain of the 2D bin packing problem, and Dowsland et al. (2007) develop a Simulated Annealing (SA) based hyper-heuristic to select the LLHs for the shippersize decision problem.

On the other hand, unlike the heuristic selection approaches in which the LLHs are pre-existing, 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 et al., 2009c), 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., 2011), etc.

Besides 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, i.e., 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 et al., 2010c), 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 sub-problem decomposition (Ong et al., 2006). Furthermore, Aarts and Lenstra (1997) 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. Similar idea has also been mentioned in Burke et al. (2010a).

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 with the goal of hyper-heuristics. First, if the LLPs are manually tuned, it would violate the objective of the 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 et al., 2009a,b) 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 et al., 2009a), thus may pose more challenges to the domain experts, e.g., 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, i.e., the offline parameter tuning and the online parameter control (Eiben et al., 2007). Traditionally, the parameter tuning task is manually conducted in a trial-and-error paradigm, which is usually time consuming and error prone. Consequently, extensive interests have been focused on the automation of the 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), advanced local search (Hutter et al., 2009), etc. 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 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 well 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 pre-scheduled strategies. For example, Merkle et al. (2002) propose an ant based algorithm, in which the evaporation rate $\rho$ starts from a small value, and gradually increases so as to achieve the 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, etc. Thus, this approach can also be considered similar with 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). Li and Li (2007) develop an adaptive ant based algorithm, in which the parameters $\alpha$ and $\beta$ 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 $\sigma$-self-adaptation in ES investigated by Hansen (2006). Serpell and Smith (2010) discuss the self adaptation of the mutation operator for the permutation representations. See (Meyer-Nieberg and Beyer, 2007) for a survey of the self-adaptive parameter control.

In summary, in this section we introduce the background of both hyper-heuristics and parameter setting techniques. Interestingly, we can observe that these two research directions share much in common. From the perspective of motivations, both directions aim at preventing the algorithms from being instance and/or problem-dependent. While 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, the 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, i.e., the LLH management Module (denoted as $M_{LLH}$), and the LLP management Module (denoted as $M_{LLP}$). On the one hand, $M_{LLH}$ selects, applies, and evaluates the LLHs in a similar way as most of the existing hyper-heuristics. On the other hand, $M_{LLP}$ is responsible for the selection and the evaluation of the 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, etc. Note that since the maintenance of the LLPs is achieved through the information exchange between $M_{LLH}$ and $M_{LLP}$, 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: $H$, $Q$, $S$, $M_{LLH}$, and $M_{LLP}$. Among these components, $H = \{LLH_1, LLH_2, \ldots, LLH_N\}$ indicates the set of LLHs, where $N$ is the number of the LLHs, $Q = \{q_1, q_2, \ldots, q_{num}\}$ is a population of $num$ LLH sequences. In the population, each sequence is defined as $q_i = (q_i^1, q_i^2, \ldots, q_i^{len})$, where $q_i^j \in H$ is the $j$th LLH of the $i$th sequence in $Q$, and $len$ 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 $S = \{s_1, s_2, \ldots, s_{num}\}$.

Finally, $M_{LLH}$ and $M_{LLP}$ are the modules to manipulate the population of LLH sequences and LLPs, respectively.

With each component of the framework specified, we now present the pseudo code of AD-HH in Algorithm 1. Before the main loop, several components are initialized, including the initial LLH sequence population $Q'$, the initial solution population $S'$, $M_{LLH}$, and $M_{LLP}$ (Lines 2-4). Then at each generation of the main loop, a new LLH sequence population $Q$ is first constructed with respect to the population $Q'$ of the previous generation, and/or some other auxiliary information (Line 6). After that, each LLH sequence $q_i \in Q$ 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,
Algorithm 1: AD-HH

Input: number of the LLH sequences $num$, length of each LLH sequence $len$, LLH management module $M_{LLH}$, LLP management module $M_{LLP}$

Output: the best solution achieved $best$

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

before invoking the parameterized LLH, the corresponding LLP is to be selected using $M_{LLP}$ (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 $Q'$ for the next generation is to be selected (Line 14). In this study, the binary tournament selection is employed. Meanwhile, the selection of $S'$ is implicitly conducted, since each LLH sequence corresponds with a feasible solution. Finally, if necessary, the structures of $M_{LLH}$ and $M_{LLP}$ are updated (Line 16).

Note that AD-HH is a generic framework, in which several interfaces (indicated by underlines in Algorithm 1) have to be implemented, so that the framework can be instantiated for problem solving. For example, we have to provide a $M_{LLH}$ specific method to construct new LLH sequences (Line 6), a $M_{LLP}$ specific method to assign LLP values (Line 11), and (possibly) structure updating methods for $M_{LLH}$ and $M_{LLP}$ (Line 16). In the following sections, we shall discuss the design and the 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 contexts of two different $M_{LLH}$ models, so as to demonstrate the flexibility of the framework.

3.1 Ant based Low Level Parameter Adaptation

In this subsection, we shall discuss the details of $M_{LLP}$, which is inspired by ant based algorithms. The reasons we choose the ant model to achieve the LLP adaptation are 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 the 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 $M_{LLH}$. 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, thus is not directly applicable to the 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 sub-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 pseudo code of the LLP adaptation are presented in detail.

In order to deal with the continuous parameters, some modifications should be introduced. Since the proposal of ACO, there has been much study that intends to modify the ant based algorithms for the problem domains other than the combinatorial optimization problems, such as Continuous ACO (CACO) (Bilchev and Parmee, 1995), ACO$_B$ (Socha, 2004; Socha and Dorigo, 2008) and ACO$_{MV}$ (Socha, 2004). Among these approaches, ACO$_B$ and its extension ACO$_{MV}$ provide a general model that is close to ACO for the discrete domain problems. Besides, 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, the 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 are 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 a LLP archive. Associated with each LLH transition $(LLH_i, LLH_j)$, if $LLH_j$ is parameterized, we keep track of a number of LLP tuples in $A_{ij}$. The $l$th tuple of $A_{ij}$ consists of the value $x_l$ of the parameter and its corresponding

\[2\] If there are more than one parameter for $LLH_j$, $x_l$ should be replaced with a vector.
objective value $v_l$ (Suppose 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 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:

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

where $w$ is the vector of the weights corresponding with each tuple, $\mu$ is the vector of the means, and $\sigma$ 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 $l$th tuple is defined as:

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

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

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

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

$$\mu = x_l,$$ 

$$\sigma = \frac{1}{\sqrt{2\pi}}.$$
Hyper-Heuristics with Low Level Parameter Adaptation

\[ \sigma = \frac{\sum_{i=1}^{L} |x_r - x_l|}{L - 1}. \]  

(5)

With \( \mu \) and \( \sigma \), the parameter value can be generated using the Box-Muller method (Box and Muller, 1958). The process of the 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. Besides, since the initialization of the archive matrix \( A \) is trivial, i.e., each archive \( A_{ij} \) is initialized to be empty, the corresponding pseudo code 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 \( M_{LLP} \) proposed in this subsection, we adopt a two-dimensional archive matrix \( A \) to capture the distribution of LLPs. Alternatively, if the LLPs are to be globally set, rather than dependent of the LLH transition, we can simply set the dimension of \( A \) to be 1. In the experiments in Section 6, we shall briefly compare these two design strategies. Besides, we shall also examine whether the ant model is capable of learning effective LLP configurations.

**Algorithm 2: GenerateParameter**

Input: the index \( i \) and \( j \) of the two LLHs  
Output: the parameter value \( p \)

begin
1. if The number of the tuples in \( A_{ij} \) is less than \( L \) then
2. Uniformly generate the value of \( p \) within the range of the parameter
3. return \( p \)
4. Select the \( l \)th Gaussian function with probability calculated by Equation (3)
5. Set the mean value of the PDF as \( \mu = x_l \)
6. Calculate the standard deviation \( \sigma \) by Equation (5)
7. Generate the value of the parameter \( p \) with Box-Muller method, with \( \mu \) and \( \sigma \) as input
8. if The parameter is required to be discrete then
9. Round \( p \) to its nearest discrete value
10. 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
1. if There exists a tuple \( (x_l, v_l) \) such that \( x_l = p \) then
2. \( v_l \leftarrow v \)
3. else
4. Construct a new tuple \( (p, v) \)
5. Insert the tuple into \( A_{ij} \)
6. Sort \( A_{ij} \) in ascending order of the objective values of each tuple
7. if The number of tuples exceeds \( L \) then
8. Exclude the tuple with the largest objective value
9. end
10. end

4 Management of Low Level Heuristics

In this section, we instantiate the AD-HH framework in the contexts of two different \( M_{LLH} \) 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 \( M_{LLH} \), 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 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 interests. Burke et al. (2005) propose an AH to solve the project presentation problem. Cuesta-Cañada et al. (2005) apply an AH with multiple pheromone matrices for the 2D bin packing problem. A recent AH algorithm is proposed by Chen et al. (2007) to solve the travelling tournament problem. Various applications have demonstrated the generality of AH. In this subsection, the modifications that have to be made to incorporate AH into AD-HH are discussed, and described in pseudo code.

In existing AHs (Burke et al., 2005; Chen et al., 2007), the LLH management is conducted as follows. First, a fully connected graph is constructed, where each vertex represents a 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 $q_k^{\text{len}}$, where $\text{len}$ 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 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 $\tau_{ij}$ represents the desirability of guiding the ants from LLH$_i$ to LLH$_j$. With the pheromone $\tau$, we can define the probability of the transition from LLH$_i$ to LLH$_j$ at each iteration:

$$P_{ij} = \frac{\tau_{ij}}{\sum_{LLH_l \in \mathcal{H}} \tau_{il}}.$$  \hspace{1cm} (6)

Note that during the initialization stage, each element of $\tau_{ij}$ is uniformly assigned with 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 those promising LLH transitions.

More detailed, with the pheromone $\tau$ illustrated, the pseudo code that fulfills the LLH sequences constructing requirement of AD-HH is presented in Algorithm 4. 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 $\tau$ to describe the LLH transition probability, at each iteration, after a LLH sequence is applied, we have to update $\tau$ 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, af-
ter the LLHs are selected and applied on the solution associated with each ant $k$, the pheromone $\tau$ is updated using Equation (7):

$$
\tau_{ij} = \begin{cases} 
\rho \cdot \tau_{ij} + \frac{v_{best}}{v_k} & \text{LLH}_i \text{ and } \text{LLH}_j \text{ are along the journey of ant } k, \\
\rho \cdot \tau_{ij} & \text{otherwise},
\end{cases}
$$

(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 $\rho$ indicates the evaporation rate. The pseudo code for updating $\tau$ 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 shall obtain AD-AH.

Algorithm 4: ConstructSequences-AH

Input: A population of LLH sequences $Q'$, A pheromone matrix $\tau$
Output: A new population of LLH sequence $Q$

begin
1 $Q \leftarrow Q'$
2 foreach LLH sequence $q_i \in Q'$ do
3 $q_1^i \leftarrow q_{len}^i$
4 for $j = 2$ to len do
5 Select the next LLH $q_j^i$ with respect to Equation (6)
6 return $Q$
end

Algorithm 5: UpdateStructure-AH

Input: A population of LLH sequences $Q$, A pheromone matrix $\tau$

begin
1 foreach LLH sequence $q_i \in Q$ do
2 for $j = 2$ to len do
3 Update $\tau$ corresponding to $q_{j-1}^i$ and $q_j^i$ using Equation (7)
4
end

4.2 Genetic Algorithm based $M_{LLH}$

In this subsection, we focus on the GA based $M_{LLH}$, 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 is first addressed by Cowling et al. (2002), to solve the trainer scheduling problem. In the paper, the authors propose 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 and Kendall (2002) improve Hyper-GA by adaptively maintaining the lengths of the LLH sequences, and propose ALChyper-GA for the same problem. Since then, there have been various GH variants. For simplicity, the GH model in this subsection is similar with Hyper-GA.

In GH, each LLH sequence $q_i \in Q$ 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 with 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 pseudo code 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 $q_i \in Q'$ of the population, another LLH sequence $q_j \in Q'$ 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 $\text{offspring}$, with respect to its parents (Line 5). After that, the offspring undergoes the mutation procedure (Line 6) to obtain $\text{offspring}'$, to achieve more diversity in the search within the heuristic space. Finally, $\text{offspring}'$ is inserted into $Q$, which is returned as the LLH sequence population of the next iteration. More detailed, the implementation of the crossover and the mutation operators is also listed (Lines 10-22). Besides, 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 shall obtain AD-GH.

**Algorithm 6: ConstructSequences-GH**

```
Input: A population of LLH sequences $Q'$
Output: A Modified population of LLH sequence $Q$
1 begin
2     $Q \leftarrow \emptyset$
3 foreach LLH sequence $q_i \in Q'$ do
4     Randomly select another sequence $q_j$ such that $q_j \in Q'$, $q_j \neq q_i$
5     $\text{offspring} \leftarrow \text{crossover-hyper}(q_i, q_j)$
6     $\text{offspring}' \leftarrow \text{mutate-hyper} (\text{offspring})$
7     $Q \leftarrow Q \cup \{\text{offspring}'\}$
8 return $Q$
9 end
10 Procedure crossover-hyper (LLH sequence $q_i$, LLH sequence $q_j$)
11 begin
12 $\text{offspring}' \leftarrow q_i^{c-1}$
13 $c \leftarrow \text{random}(2, \text{len})$
14 for $k = 2$ to $c$ do $\text{offspring}' \leftarrow q_k^c$
15 for $k = c + 1$ to len do $\text{offspring}' \leftarrow q_k^c$
16 return $\text{offspring}$
17 end
18 Procedure mutate-hyper (LLH sequence $\text{offspring}$, mutation rate $\theta$)
19 begin
20 for $i = 1$ to len do Randomly change the LLH of $\text{offspring}'$ with probability $\theta$
21 return $\text{offspring}$
22 end
```

In summary, in this section, we instantiate AD-HH in the contexts 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 Bi-partition

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 the 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 there may exist redundancy in existing LLH selection based hyper-heuristics. As stated in (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. As the solution, the SAR mechanisms is 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), etc. 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), or the combination of the intensification and the diversification (Blum and Roli, 2003). In the definition, the exploitation (also called intensification) indicates those strategies that conduct the intensive search in order to improve the solution quality; while the 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 the solution quality; while 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 contexts 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 subsection, we shall 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 the heuristic space reduction, we replace the fully connected graph induced by all the LLHs with a bipartite graph derived 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 \( \Pi \) 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 study as well. For example, Özcan et al. (2006) classify the LLHs into the hill climbers and the mutational heuristics; Meignan et al. (2010) partition the LLH set into the intensifier set and the diversifier set. Another division criterion is proposed in (Burke et al., 2009b), in which the diversification LLHs are further classified into mutational, ruin-recreate, crossover, and others. For simplicity, we only bi-partition 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 for LLHs, at each iteration of the search process, the path of each solution is constructed by traversing the bipartite graph, and the LLH transition probability is modified as:

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

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 with AD-AH, in essence the AM criterion is employed in AD-GH. Thus, in this subsection we discuss how to conduct the SAR mechanism for AD-GH. In order to achieve the reduction of the heuristic space, we modify Procedure \texttt{mutate-hyper} in Algorithm 6, so as to reduce the search space, meanwhile balance the intensification and the diversification of the search process. More detailed, the modifications are presented in Procedure \texttt{mutate-hyper-reduction}.
Hyper-Heuristics with Low Level Parameter Adaptation

**Procedure** mutate-hyper-reduction (LLH sequence $q$, mutation rate $\theta$)

**Output**: mutated LLH sequence

1. **begin**
2. for $i = 2$ to $\text{len}$ do
3.   if $(\text{len} - i) \% 2 = 0$ then
4.     if $q_i \notin \mathcal{I}$ then Randomly select $q'_i \in \mathcal{I}$
5.     else if random $(0,1) < \theta$ then Randomly select $q'_i \in \mathcal{I}$
6.   else
7.     if $q_i \notin \mathcal{D}$ then Randomly select $q'_i \in \mathcal{D}$
8.     else if random $(0,1) < \theta$ then Randomly select $q'_i \in \mathcal{D}$
9. **return** $q$
10. **end**

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 LLHs that the constraint is not violated, the mutation is carried out with probability $\theta$, which is similar as in 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 will 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). Besides, there are several parameterized LLHs for the $p$-median problem, which makes it suitable as the test problem.

Given a set $F$ of $m$ facilities, a set $U$ of $n$ users, and a $n \times m$ 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 set $F$, so as to minimize the sum of these costs:

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

Evolutionary Computation Volume x, Number x 17
Since each solution to a \( p \)-median instance consists of a subset of medians with fixed number \( p \), in this study we adopt the 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.

**Interchange:** Interchange is 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), respectively. 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 latter implementation.

**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 a 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 a 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 \( k \) is sampled with three typical values. In this paper, \( k \) is treated as a parameter, which is adaptively maintained instead.

**Crossover and mutation:** These two heuristics 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 non-median facility with a probability indicated by \( \text{mutation-rate} \).

Initialization with pheromone (\text{AntInit}): At each iteration of ACO (Kochetov et al., 2005), each solution is constructed with probability according to the pheromone trail.

**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 \( \text{shake-strength} \) that represents the distance (indicated by the number of different medians) between the input solution and the output solution of the shake operation.

**Random:** Random is a restart operator, which can provide diversification functionality as well.

**Random plus greedy (RPG):** First proposed in (Resende and Werneck, 2004), RPG is also a restart operator in which randomness is combined with 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 are performed on a Pentium IV 3.2 GHz PC with 4GB memory, running GNU/Linux with kernel 3.0.0. All the codes are implemented in C++, compiled using g++ 4.5 with flag -O2. The running 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 ORLIB (Beasley, 1985) class, 5 random instances from RW (Resende and Werneck, 2003), and 10 Euclidean based instances from TSPLIB (Reinelt, 1991) class. Each instance from ORLIB is repre-
Table 1: HLS Parameter Configurations

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

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 the 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, which are set with the same values as (Han and Kendall, 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 the 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 time it will take before our algorithms could achieve the convergence. Thus for efficiency consideration, the parameters len is set to 5 in this study.

6.2 Numerical Results

In this subsection, we present the performance of our algorithms, i.e., 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 PBS (Pullan, 2008). PBS achieves the best known upper bounds for most benchmark instances. However, since
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-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.
standard deviation (denoted as $\text{avg} \pm \text{stddev}$), the average percentage error rate (denoted as $\%\text{err}$), and the average time elapsed for AD-GHSAR and AD-AHSAR, respectively. Among these measurement, $\%\text{err}$ is defined following (Hansen and Mladenović, 1997):

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

(12)

where $v_{\text{avg}}$ and $v_{\text{opt}}$ 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 the optimal solutions ($\%\text{err} = 0$ 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 5 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. More detailed, AD-AHSAR outperforms PBS over 5 instances, obtains 1 currently best known upper bound, and gets outperformed by PBS over 4 instances. For AD-GHSAR, similar observations can be drawn. Over all the 10 TSPLIB instances, AD-GHSAR outperforms PBS over 3 instances, achieves 2 currently best known upper bound, and there are 5 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 can 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 6 instances over which AD-AHSAR outperforms AD-GHSAR, and 2 instances over which AD-GHSAR performs better. Besides, we can observe that both algorithms are stable, and have small standard deviations.

As a brief summary, in this subsection, we present 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 shall investigate the underlying reasons for the encouraging performance.

### 6.3 Effectiveness Evaluation

In this subsection, we examine the effectiveness of the proposed framework from various aspects, so as to investigate the reasons for the competitive performance. By conducting the experiments, we intend to answer the following questions:

**Q1: Is the design of the LLP adaptation reasonable and beneficial?**

This question investigates the two hypotheses raised in Section 3.1, i.e., 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, the hyper-heuristics with randomly selected LLPs (denoted as R-AH and R-GH). These two
algorithms are similar with 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, 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).

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, respectively. More specifically, the comparisons are conducted between the hyper-heuristics with/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 shall 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.

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, i.e., 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 to investigate the influence of each mechanism on the whole framework.

To investigate these questions, this subsection 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.

6.3.1 Experimental Setup
In this subsection, we shall introduce the background of the experiments. The performance measurement and statistical tests. When comparing the performance of the algorithms in this subsection, we concentrate on the effectiveness of the framework. More precisely, for all the comparisons in this subsection, 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 0.05 are treated to be statistically significant), unless otherwise stated.

**Test instances and LLP tuning.** To conduct the statistical tests, a set of benchmark instances has to be specified. In this subsection, 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 the 55 instances. On the contrary, 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, i.e., the training set and the test set. After the tuning of the LLPs over the training instances, the performance of the algorithms are 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 3 heterogeneous classes of instances in this study, this scenario is further divided into 3 sub-scenarios, depending on the training instances (1.1 for TSPLIB, 1.2 for OR, and 1.3 for RW, respectively). For example, in Scenario 1.1, the training instances are selected from the TSPLIB instances, and all the rest 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 the cross domain problem solving as in Hyflex (Burke et al., 2009a,b), or the 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 cover all the three instance sets, so as to simulate the situation in which the practitioners have the 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 5, taken into account that there are 5 RW instances. For each scenario, the training instances are randomly selected. 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 irace4 (López-Ibáñez et al., 2011) with limited execution time for each algorithm, i.e., we restrict each algorithm to be performed no longer than 60 seconds, 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 within [1, p]. However, since the range of the latter two LLPs are dependent of the number of medians p, during the LLP tuning, we introduce two real-valued parameter shake-rate and LK-rate, both of which ranging within [0.1, 0.9]. After the pre-tuning task, the actual LLPs used are ⌈shake-rate × p⌉ and ⌈LK-rate × p⌉.

The tuning results are summarized in Table 3, which is organized as follows. Col-

---

5Note that in the same scenario, the training instances are the same for all the algorithms to be tuned.
6irace is an R implementation of iterated F-Race, available at http://iridia.ulb.ac.be/irace/
7Although with such restriction, the tuning is still very time consuming. For example, it takes more than 20 hours for the tuning of AHSAR in Scenario 1.1.
Table 3: Pre-tuned LLP Configurations

<table>
<thead>
<tr>
<th>Scenario</th>
<th>LLP Name</th>
<th>AHSAR</th>
<th>GHSAR</th>
<th>AH</th>
<th>GH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1.1 (TSPLIB)</td>
<td>shake-strength</td>
<td>0.5830</td>
<td>0.5038</td>
<td>0.1982</td>
<td>0.3637</td>
</tr>
<tr>
<td></td>
<td>mutation-rate</td>
<td>0.4076</td>
<td>0.7033</td>
<td>0.4230</td>
<td>0.2861</td>
</tr>
<tr>
<td></td>
<td>LK-rate</td>
<td>0.4397</td>
<td>0.5072</td>
<td>0.4615</td>
<td>0.4309</td>
</tr>
<tr>
<td>Scenario 1.2 (ORLIB)</td>
<td>shake-strength</td>
<td>0.3114</td>
<td>0.3033</td>
<td>0.2255</td>
<td>0.1573</td>
</tr>
<tr>
<td></td>
<td>mutation-rate</td>
<td>0.4001</td>
<td>0.1001</td>
<td>0.2745</td>
<td>0.7204</td>
</tr>
<tr>
<td></td>
<td>LK-rate</td>
<td>0.1125</td>
<td>0.1045</td>
<td>0.1987</td>
<td>0.1492</td>
</tr>
<tr>
<td>Scenario 1.3 (RW)</td>
<td>shake-strength</td>
<td>0.3121</td>
<td>0.3235</td>
<td>0.7113</td>
<td>0.2240</td>
</tr>
<tr>
<td></td>
<td>mutation-rate</td>
<td>0.1998</td>
<td>0.2788</td>
<td>0.1884</td>
<td>0.3744</td>
</tr>
<tr>
<td></td>
<td>LK-rate</td>
<td>0.2012</td>
<td>0.1472</td>
<td>0.2640</td>
<td>0.3265</td>
</tr>
<tr>
<td>Scenario 2 (ALL)</td>
<td>shake-strength</td>
<td>0.4008</td>
<td>0.5000</td>
<td>0.2614</td>
<td>0.5101</td>
</tr>
<tr>
<td></td>
<td>mutation-rate</td>
<td>0.8129</td>
<td>0.3127</td>
<td>0.3665</td>
<td>0.3012</td>
</tr>
<tr>
<td></td>
<td>LK-rate</td>
<td>0.6123</td>
<td>0.5009</td>
<td>0.5396</td>
<td>0.4989</td>
</tr>
</tbody>
</table>

Column 1 indicates the scenarios described above. Column 2 represents the LLPs’ 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 values of the LLPs 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 of 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.

6.3.2 Comparison Results and Discussion

After introducing the background information of the experiments, we shall now conduct the comparisons between various algorithms, so as to evaluate the performance of the framework. In order to gain an intuitive understanding about 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. The sub-figures in the first column (i.e., Figures 3(a), 3(d), 3(g) and 3(j)) illustrate the results for the comparisons that correspond to Q1. The sub-figures in Columns 2 and 3 are presented for Q2 and Q3, respectively. Take Figure 3(b) as an example, we shall describe the content of each sub-figure. Figure 3(b) illustrates the comparison between AD-AH and AH. In the sub-figure, the x-axis and the y-axis indicate the \(\%\text{err}\) of AD-AH and AH over the test instances, respectively. More specifically, each point \((x, y)\) in the sub-figure indicates that there are one or more instances over which AD-AH’s \(\%\text{err}\) and AH’s \(\%\text{err}\) 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 smaller \(\%\text{err}\). 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 sub-figures in Figure 3 are relatively sparse. The reason is that each point in the sub-figures may represent multiple instances. For example, since the ORLIB instances are relatively easy, the origin \((0, 0)\) represents the comparisons on multiple instances.

Companion with Figure 3, the results of the statistical tests are presented in Table 4, which are organized as follows. The first column indicates the three questions, as described above. The second column presents all the comparisons. Then in Columns 3-
Hyper-Heuristics with Low Level Parameter Adaptation

Figure 3: Comparison of the $\%err$ of each algorithm in different LLP tuning scenarios.
Table 4: Performance comparison using Wilcoxon test

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Scenario 1.1 (TSPLIB) p-value</th>
<th>Scenario 1.2 (ORLIB) p-value</th>
<th>Scenario 1.3 (RW) p-value</th>
<th>Scenario 2 (ALL) p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD-AH vs. R-AH</td>
<td>0.0171 (AD-AH)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD-GH vs. R-GH</td>
<td>0.0347 (AD-GH)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD-AH vs. AD-AH-1D</td>
<td></td>
<td></td>
<td>0.2661 (-)</td>
<td></td>
</tr>
<tr>
<td>AD-GH vs. AD-AH-1D</td>
<td></td>
<td></td>
<td>0.3575 (-)</td>
<td></td>
</tr>
<tr>
<td>AD-AH vs. AH</td>
<td>0.5693 (-)</td>
<td>0.0097 (AD-AH)</td>
<td>0.1475 (-)</td>
<td>0.0479 (AH)</td>
</tr>
<tr>
<td>AD-GH vs. GH</td>
<td>0.2775 (-)</td>
<td>0.003 (AD-GH)</td>
<td>0.0106 (AD-GH)</td>
<td>0.0522 (GH)</td>
</tr>
<tr>
<td>AHSAR vs. AH</td>
<td>0.0005 (AHSAR)</td>
<td>0.0001 (AHSAR)</td>
<td>0.0098 (AHSAR)</td>
<td>0.0002 (AHSAR)</td>
</tr>
<tr>
<td>GHSA vs. GH</td>
<td>0.0038 (GHSA)</td>
<td>0.0015 (GHSA)</td>
<td>0.0057 (GHSA)</td>
<td>0.0179 (GHSA)</td>
</tr>
<tr>
<td>AD-AHSAR vs. AHSAR</td>
<td>0.3008 (-)</td>
<td>0.0017 (AD-AHSAR)</td>
<td>0.0039 (AD-AHSAR)</td>
<td>0.8311 (-)</td>
</tr>
<tr>
<td>AD-GHSAR vs. GHSAR</td>
<td>0.4548 (-)</td>
<td>0.0021 (AD-GHSAR)</td>
<td>0.0137 (AD-GHSAR)</td>
<td>0.4630 (-)</td>
</tr>
<tr>
<td>AD-AHSAR vs. AD-AH</td>
<td></td>
<td>0.0005 (AD-AHSAR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD-GHSAR vs. AD-GH</td>
<td></td>
<td>0.0034 (AD-GHSAR)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6, the results of the Wilcoxon tests are reported in each scenario. The comparison results consist of the p-value, as well as the name of the algorithm that performs better in the comparison (we present a hyphenation for those comparisons in which the difference is not significant).

After presenting and describing the comparison results, we shall 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.

Investigation of Q1. To answer Q1, we investigate the two hypotheses raised in Section 3.1, i.e., 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 can observe that most points in the sub-figure 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 95% confidence level (p-value = 0.0171). This means that the effectiveness of the LLP adaptation does not result from the random selection of the LLP values. Contrarily, the ant model is able to intelligently adapt the LLP configurations. This observation also partially confirms the assumption that the ant based adaptation has the learning capability, which is suitable for our framework. Then, we proceed to examine the second hypothesis. From Figure 3(g), we can 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 = 0.2611), which means that the two algorithms perform similarly. Meanwhile, similar observation can be drawn for AD-GH. For example, AD-GH outperforms R-GH (p-value = 0.0347), and performs similarly with AD-GH-1D (p-value = 0.3575).

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 differences between the two variants. We suppose that the difference between the two variants might be more significant if there are more parameterized LLPs, which
Hyper-Heuristics with Low Level Parameter Adaptation

shall deserve more future work.

Although the statistical tests could not tell the 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 $p = 200$), in a typical execution of the algorithm. In each sub-figure, the two lines represent the mutation-rate’s value, when applied after LK($k$) and interchange, respectively. We can observe that AD-AH (AD-GH) prefers 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 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.

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 shall 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.

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

---

7The reason we do not choose the ORLIB instances is that these instances are relatively easy, thus the algorithms may converge too fast.
Investigation of Q2. Now we shall 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 can 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 \((p\text{-value } = 0.0479)\) 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 \((p\text{-value } = 0.5693 \text{ and } 0.1475)\). Besides, in Scenario 1.2, most points (denoted as crosses) lie above the reference line, indicating that AD-AH outperforms AH in this scenario \((p\text{-value } = 0.0097)\).

These observations may imply that if all the instance distributions are known \textit{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 3 instance sets, ORLIB instances are relatively easy, thus 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 the knowledge of about the instance distribution, and tend to be more stable, compared with AH. Similarly, when we compare AD-GH and GH, we can observe that AD-GH gets outperformed by GH in Scenario 2 \((p\text{-value } = 0.0522, \text{ which means that the confidence level for this comparison is } 90\%)\), performs similarly with GH in Scenarios 1.1 \((p\text{-value } = 0.2755)\), and outperforms GH in scenarios 1.2 and 1.3 \(p\text{-value } = 0.003 \text{ and } 0.0106\), 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 tend 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 the high quality solutions simultaneously. On the other hand, AD-AH (AD-GH) is able to obtain better performance if this pre-condition does not hold.

Then, we intend to investigate the SAR mechanism. In order to test the effectiveness of this mechanism, the comparisons are conducted between AHSAR (GHSAR) and AH (GH). From Figure 3(h), we can observe that most points lie above the reference line, regardless of the training scenarios. Meanwhile, from Table 4, we can see that in both Scenarios 1 and 2, AHSAR consistently outperforms AH (with \textit{p-value} < 0.01). When we compare GHSAR and GH, we can observe that GHSAR also outperforms GH in both Scenarios 1 and 2 \((p\text{-value } < 0.02)\). 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. Take 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%.

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 their drawbacks. On the one hand, in AD-AH (AD-GH), the intro-
duction 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 shall investigate the combination of the two mechanisms.

**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, i.e., 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 with what has been observed. From Figure 3(c), we can 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 $< 0.004$), while in Scenarios 1.1 and 2, AD-AHSAR performs similarly with AHSAR ($p$-value $= 0.3008$ and 0.8311, respectively). Then, we compare AD-AHSAR with AD-AH. As shown in Figure 3(i), it is obvious that AD-AHSAR outperforms AD-AH ($p$-value $= 0.0005$). Similar observations can be drawn when we compare AD-GHSAR with GHSAR and AD-GH, i.e., AD-GHSAR performs at least as well as GHSAR, and statistically outperforms AD-GH.

**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 avoids their drawbacks.

In conclusion, in this subsection, extensive experiments are 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 demonstrate 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 subsection, we intend to examine the run-time 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, etc. 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 certain quality threshold as time elapses. In this subsection, 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, i.e., fl1400 with $p = 500$ and rw1000 with $p = 200$, from TSPLIB and RW instance set, respectively. The reason we do not choose the ORLIB instances is that these instances are relatively easy, thus cannot be used to distinguish the run-time behaviors of the algorithms, which is similar as 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 seconds, 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 run-time behavior is represented by a RTD curve determined from the 100 runs of the algorithm. In each sub-figure 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 $P_{\text{RTD}}$) that the algorithm achieves the pre-defined 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 can 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 seconds. Besides, at the cutoff time, $P_{\text{RTD}}$ 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 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 can 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 bi-partitioning 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 $P_{\text{RTD}}$, 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 can observe that at the beginning of the search process, the RTD curve of AHSAR lies above that of AD-AHSAR. However, at the cutoff time, the cumulative probability $P_{\text{RTD}}$ of AHSAR is similar with that of AD-AHSAR. Another interesting observation can be drawn from Figure 5(c). Over the RW instance, AHSAR converges after around 30 seconds, with $P_{\text{RTD}} = 41\%$. On the other hand, although AD-AHSAR converges slower, the final $P_{\text{RTD}}$ reaches 51%.

Again, similar observation can be drawn by GH and its variants, which is obvious from Figures 5(b) and 5(d). In conclusion, in this subsection, we evaluate the run-time 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 Low Level Parameters (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. Finally, we use $p$-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 shall test these hypotheses in the context of a larger LLP search space. (2) In this study, we do not investigate the influences of the High Level Strategy (HLS) parameters. As discussed, some of these parameters may have an impact on the quality of the solutions. It is a potential research direction 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 with other approaches (Özcan et al., 2006; Meignan et al., 2010; Burke et al., 2009b). 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. For the 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


