New Insights Into Diversification of Hyper-Heuristics

Zhilei Ren, He Jiang, *Member, IEEE,* Jifeng Xuan, Yan Hu, and Zhongxuan Luo

***Abstract*—There has been a growing research trend of applying hyper-heuristics for problem solving, due to their ability of balancing the intensification and the diversification with low level heuristics. Traditionally, the diversification mechanism is mostly realized by perturbing the incumbent solutions to escape from local optima. In this paper, we report our attempt toward providing a new diversification mechanism, which is based on the concept of instance perturbation. In contrast to existing approaches, the proposed mechanism achieves the diversification by perturbing the instance under solving, rather than the solutions. To tackle the challenge of incorporating instance perturbation into hyper-heuristics, we also design a new hyper-heuristic framework HIP-HOP (recursive acronym of HIP-HOP is an instance perturbation-based hyper-heuristic optimization procedure), which employs a grammar guided high level strategy to manipulate the low level heuristics. With the expressive power of the grammar, the constraints, such as the feasibility of the output solution could be easily satisfied. Numerical results and statistical tests over both the Ising spin glass problem and the *p*-median problem instances show that HIP-HOP is able to achieve promising performances. Furthermore, runtime distribution analysis reveals that, although being relatively slow at the beginning, HIP-HOP is able to achieve competitive solutions once given sufficient time.**

***Index Terms*—Hyper-heuristics, instance perturbation, Ising spin glass, linear genetic programming, *p*-median.**

LIST OF ACRONYMS

LLH	Low Level Heuristic
$_{IP}$LLH	Instance Perturbation based LLH
$_{SP}$LLH	Solution Perturbation based LLH
SOPHY	SOlution Perturbation based HYper-heuristic
SOPHY-LONG	SOPHY with longer cut off time
HIPHOP	HIP-HOP is an Instance Perturbation based Hyper-heuristic Optimization Procedure

Manuscript received March 14, 2013; revised July 31, 2013 and November 1, 2013; accepted November 20, 2013. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant DUT13RC(3)53, in part by the New Century Excellent Talents in University under Grant NCET-13-0073, and in part by the National Natural Science Foundation of China under Grant 61370144, Grant 61175062, and Grant 61300017. This paper was recommended by Associate Editor Y. S. Ong.

Z. Ren, H. Jiang, Y. Hu, and Z. Luo are with the School of Software, Dalian University of Technology, Dalian 116621, China (e-mail: zren@dlut.edu.cn; jianghe@dlut.edu.cn; huyan@dlut.edu.cn; zxluo@dlut.edu.cn).

J. Xuan is with INRIA Lille–Nord Europe, Lille 59650, France (e-mail: jifeng.xuan@inria.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2013.2294185

I. INTRODUCTION

HYPER-HEURISTICS have attracted much research attention in recent years [1]–[4]. A hyper-heuristic is defined as an automated methodology for selecting or generating heuristics to solve hard computational search problems [5]. In general, a hyper-heuristic consists of a high-level strategy and a set of low-level heuristics (LLHs). Many well known algorithms are employed as the high level strategy to adaptively manage the LLHs in solving cross-domain problems, including genetic algorithm [6], ant colony optimization [7], and genetic programming [1]–[3], [8]. Accordingly, the mostly used LLHs can be summarized as follows [9].

1) Local search: conducts iterative neighborhood moves to improve the quality of the input solution, until a local optimum is reached.
2) Mutation: performs small modifications on the solution, by changing the variable values of solution components.
3) Ruin-recreate: partially destroys, and then repairs the solutions. For example, the shake operator in variable neighborhood search falls into this category [10].
4) Crossover: takes two solutions as parents, and returns a new solution (offspring) as the output. Typical examples include uniform crossover and two-point crossover.

As stated in [4] and [11], the above LLHs can be classified into two categories, i.e., the intensifying LLHs that provide the intensification mechanisms, and the diversifying LLHs that implement the diversification mechanisms. In general, the local search operators belong to intensifying LLHs, in that they are mostly employed to improve the solution quality. Accordingly, mutation, ruin-recreate, and crossover fall into the diversifying LLH category. More specifically, since these three types of LLHs help the search escape from the attraction basin by perturbing the solutions [9], in this paper, these LLHs are also called solution perturbation-based LLHs ($_{SP}$LLHs). Similar as traditional evolutionary algorithms that balance the intensification and the diversification in the search process [12], a challenging issue in hyper-heuristics is how to find new effective intensification or diversification mechanisms.

In this paper, we introduce a new diversification mechanism for hyper-heuristics by proposing a set of novel diversifying LLHs, namely, the instance perturbation-based LLHs ($_{IP}$LLHs). Our basic idea is motivated by the great success of the instance perturbation-based heuristic algorithms in the literature, including search space smoothing [13], [14], weight annealing [15], [16], and fine tuned learning [17]. These algorithms share a common characteristic, i.e., instead

of perturbing the solutions to escape from the local optima, the instance[1] is perturbed to create plausible new searching directions [19]. However, the idea of perturbing the instance has not been systematically investigated in the context of hyper-heuristics. In this paper, we propose a set of $_{\text{IP}}$LLHs by perturbing the instance, including i-mutation, i-ruin-recreate, and i-crossover. All the proposed $_{\text{IP}}$LLHs can achieve the diversification in a different way, as compared with mutation, ruin-recreate, and crossover.

1) i-mutation: assigns weights to the instance components to obtain perturbed instances, and local optima over the instance may not be locally optimal over the perturbed ones.
2) i-ruin-recreate: derives perturbed instances by partially destroying the components of the original instance.
3) i-crossover: is an extension of i-mutation and takes the information of two solutions into account.

Second, we propose a new hyper-heuristic named HIP-HOP to manipulate these new $_{\text{IP}}$LLHs. Instance perturbation-based hyper-heuristic optimization procedure (HIP-HOP) (inspired by the definition of "GNU's Not Unix" [20]). The difficulty of incorporating $_{\text{IP}}$LLHs into existing hyper-heuristics lies in the fact that, most existing instance perturbation-based heuristic algorithms employ certain prescheduled schemes that control the degree/strength of the perturbation, so that the final output is a feasible solution to the instance [13], [17]. However, existing hyper-heuristics usually employ mutation or crossover operators in their high level strategies to modify the LLH sequences [4], [8]. Hence, certain maintenance strategies are required to handle the perturbation schedule so that $_{\text{IP}}$LLHs could be incorporated into hyper-heuristics. Following the hierarchy of many other hyper-heuristics [2], [21], HIP-HOP is designed to be a two-layered framework. In the high level, a grammar guided high level strategy is developed to manipulate the LLHs, including intensifying LLHs and $_{\text{IP}}$LLHs. This grammar guided high level strategy has the following features. 1) With the expressive power of the grammar, the constraints, such as the feasibility of the output solution could be easily satisfied. 2) A good tradeoff between the diversification and the intensification could be achieved. 3) $_{\text{IP}}$LLHs and $_{\text{SP}}$LLHs could be managed in a similar way. With minor modifications to the grammar, HIP-HOP could be transformed into a sOlution perturbation-based hyper-heuristic (SOPHY), a comparative hyper-heuristic in which only $_{\text{SP}}$LLHs are employed to provide the diversification mechanism. This feature also enables the fair comparisons between $_{\text{IP}}$LLHs and $_{\text{SP}}$LLHs.

To evaluate the generality of HIP-HOP, we consider the applications to the Ising spin glass problem and the p-median problem. Extensive experiments over the two minimization problems are conducted to examine HIP-HOP from both the effectiveness and the efficiency perspectives. On the one hand, from the effectiveness perspective, numerical experiments are carried out over 60 benchmark instances for each problem domain. The results show that, HIP-HOP is able to achieve promising performances, which are comparable to the state-of-the-art results. In particular, for the p-median problem, HIP-HOP achieves better results than the best known results over three instances. Meanwhile, HIP-HOP statistically outperforms SOPHY, which to some extent demonstrates the effectiveness of the $_{\text{IP}}$LLHs. On the other hand, from the efficiency perspective, runtime distribution analysis is employed to investigate the dynamic properties of HIP-HOP. Experiments over representative instances indicate that, although being relatively slow at the beginning, once given sufficient time, HIP-HOP is able to achieve very competitive results.

The rest of this paper is organized as follows. In Section II, we briefly introduce the related work of both hyper-heuristics and instance perturbation-based algorithms. In Section III, we propose the $_{\text{IP}}$LLHs with applications to the two problem domains. In Section IV, the HIP-HOP framework is presented. After that, we develop a grammar guided high level strategy in Section V. Experimental results are given in Section VI. Finally, Section VII concludes the paper and presents several potential research directions.

II. Related Work

A. Hyper-Heuristics

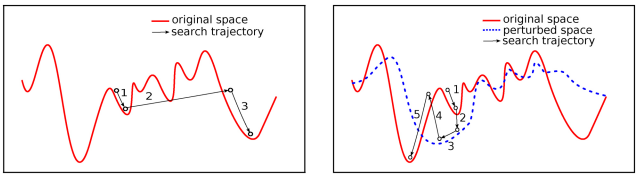
The motivation of hyper-heuristics is to raise the level of generality at which search methodologies can handle [22]. This objective is achieved by introducing a domain barrier, such that the domain-specific LLHs could be separated from the domain independent high level strategy. As stated in [5], hyper-heuristics could be classified from multiple dimensions.

From the perspective of the high level strategy, hyper-heuristics could be classified into the following four categories [5]: 1) simple random and choice function [23]; 2) greedy and peckish [24]; 3) metaheuristics-based approaches [2]; and 4) learning-based algorithms [25]. Furthermore, the hypothesis that evolutionary algorithms are able to evolve high-quality heuristics has been empirically validated by the fitness landscape analysis [26], [27]. In HIP-HOP, we adopt a genetic programming-based design, due to the constraint handling ability [28] and the promising expressiveness [29] of genetic programming.

From the perspective of LLHs, hyper-heuristics could be classified into two categories, i.e., those which use constructive LLHs and those based on perturbative LLHs. For the constructive LLH-based hyper-heuristics, the input is an empty solution, and each LLH represents a variable assignment strategy. After all the LLHs have been applied incrementally, the output is a feasible solution to the problem instance. For example, the algorithms proposed in [30] and [31] fall into this category. On the contrary, in a perturbative LLH-based hyper-heuristic [4], [8], the LLHs are used to iteratively improve the input solution, which is a feasible solution already. The framework we propose in this paper falls into this category. Since the perturbative LLH-based hyper-heuristics are closely related to the adaptive memetic algorithms [32], many ideas for problem solving could be borrowed.

In this paper, we are interested in evolving heuristics with an evolutionary high level strategy. The major difference between the proposed framework and the existing perturbative

[1]Given the problem formulations, an instance could be obtained by specifying all the problem parameters [18].



(a) (b)

Fig. 1. Two approaches to escaping from local optima. (a) Solution perturbation. (b) Instance perturbation.

LLH-based hyper-heuristics lies in the realization of the diversification mechanisms. Instead of perturbing the incumbent solutions to escape from local optima, we intend to investigate the possibility of integrating hyper-heuristics and instance perturbation based methodologies, which could implement the diversification functionalities in a novel paradigm.

B. Instance Perturbation-Based Problem Solving

In this subsection, we briefly introduce the instance perturbation-based problem solving techniques. Fig. 1 illustrates the difference between the solution perturbation and the instance perturbation-based techniques [33]. More specifically, Fig. 1(a) describes a typical diversification-intensification cycle over an imaginary search space (suppose that the problem is a minimization problem). First, the local search guides the solution to reach a local optimum. Then, diversifying operators (e.g., mutation, crossover) attempt to perturb the solution to a different region of the search space. After that, local search could be applied again, in the hope that the search could escape from the former attraction basin. In contrast, the instance perturbation-based approaches try to solve the problem in a different scheme. Fig. 1(b) depicts the main idea of the diversification-intensification cycle in instance perturbation-based approaches [15], [33]. When the search gets stuck at a local optimum, instead of perturbing the solution directly, the search space is perturbed, so that the local optimum on the original search space is not locally optimal on the perturbed one. Consequently, local search could be executed to continue the search procedure. After the local search finishes, the solution is transferred back to the original search space, where it might be further improved. Within the perturbation, it is possible that the instance-related information could be extracted and leveraged.

For example, Gu and Huang [13] proposed the search space smoothing algorithm for the traveling salesman problem. In the algorithm, the search landscape changes gradually from a smooth one to the original, rugged one. By this strategy, search space smoothing aims to prevent the search procedure from premature convergence. Similar ideas could be found in fine-tuned learning [17], noising methods [34], and several backbone guided algorithms [35]–[38]. Recently, the weight annealing-based algorithms have been applied to many problem domains, such as the traveling salesman problem and the Ising spin glass problem [15], and 1-D bin packing [16]. In weight annealing approaches, the instance perturbation is realized by assigning weights to each local part of the instance. Furthermore, Ninio and Schneider [15] developed several knowledge exploitation strategies, such as random reweighting and adversarial reweighting.

We could summarize several commonalities in these instance perturbation-based approaches. 1) These approaches have the potential to exploit the instance information. 2) Most of these approaches could be formulated under a generic framework, with a relatively unified interface. 3) The instance perturbation-based methodologies are flexible, as demonstrated by the promising results from various problem domains.

However, despite of the promising performances, the idea of perturbing the instance has not been systematically investigated in the context of hyper-heuristics, to the best of our knowledge. The difficulty, as mentioned in Section I, is that the constraints, such as the solution feasibility, have to be carefully handled by certain prescheduled scheme. In this paper, we intend to incorporate $_{\text{IP}}$LLHs and genetic programming-based high level strategy into an integrated framework. On the one hand, the $_{\text{IP}}$LLHs are able to provide novel diversification mechanisms. On the other hand, with the expressive power of genetic programming, the feasibility constraint could be implicitly satisfied.

III. Instance Perturbation-Based Low Level Heuristics

In this section, we develop a set of $_{\text{IP}}$LLHs. In Section III-A, we first introduce the general forms of the $_{\text{IP}}$LLHs. Then, in Sections III-B and III-C, respectively, we use the Ising spin glass problem and the p-median problem as two casestudies, to demonstrate the generality of these $_{\text{IP}}$LLHs.

A. General Forms of $_{IP}$LLHs

1) *i-Mutation:* The i-mutation is inspired by the reweighting strategies in [15]. Given an instance space Π (the set consists of all the possible instances of a given problem), and the solution space S, the objective function is denoted as a mapping $f : S \times \Pi \to \mathbb{R}$. First, the objective function f is written as the sum of several subfunction, each of which corresponds with a local part of the instance

$$f(s, \pi) = \sum_{i=1}^{N} f_i(s, \pi) \tag{1}$$

where $\pi \in \Pi$ is an instance, and $s \in S$ is a feasible solution to π. Then, by introducing a weight vector w, a weighted objective function f^w is defined as

$$f^w(s, \pi) = \sum_{i=1}^{N} w_i \times f_i(s, \pi). \tag{2}$$

By assigning the values of the weight vector $w \in \mathbb{R}^N$, the search landscape of the solution could be fine-tuned. For example, if all the weights equal to 0, the search landscape would be flat, while if all the values equal to 1, the weighted objective function f^w would degenerate to the original objective function. With proper weight assignment strategy, a former local optimum might not be locally optimal under the same neighborhood structure, when considering the weighted

objective function. Thus, local search could be applied, to continue the search process.

Optionally, an instance mapping function $T : \Pi \times \mathbb{R}^N \to \Pi$ could be constructed, so that $f(s, T(\pi, w)) = f^w(s, \pi)$ holds for all s, w and π. With this step, we are able to construct a perturbed instance.[2]

After the perturbed instance has been constructed, local search is applied over the instance $T(\pi, w)$. When the local search terminates, the obtained solution is evaluated again over the original instance π. Hopefully, local search could be further applied to the solution, over π.

2) *i-Ruin-Recreate:* The motivation of this operator is intuitive, which has also been proposed in this paper. For example, in [33], an instance perturbation-based algorithm for the traveling salesman problem is proposed. In the algorithm, when a solution is stuck in a local optimum, several instance variables (represented by coordinates of cities) are randomly selected and removed from the instance, then local search is applied over the reduced instance. When the local search finishes, the solution is repaired to be a feasible solution to the original instance, which hopefully is a nonlocal optimum.

3) *i-Crossover:* In this paper, the instance perturbation-based crossover could be considered as a special case of i-mutation, i.e., i-crossover is based on the weight assignment strategy as well. The main difference is that, instead of considering only one solution, in i-crossover, the information from both the two parent solutions is considered. For instance, the restricted weight modification could be conducted over part of the solutions (e.g., the intersection of the two parent solutions).

B. $_{IP}$LLHs for Ising Spin Glass Problem

Finding the ground states of the Ising spin glass is a classic problem, which has been well investigated in statistical physics. The Ising spin glass problem aims to find a state of spins called the ground state for given coupling constants J_{ij} that minimizes the energy. Searching for the ground states of the Ising spin glass is equivalent to locating the global optimum for the minimum-weight cut problem [40]. Since minimum-weight cut is NP-hard [41], the task of finding a ground state of an unconstrained Ising spin glass instance is NP-hard. In this paper, we consider a special case, in which the spins are located on a 2-D grid, and each spin interacts with its nearest neighbors.

Each Ising spin glass instance is represented by an interaction matrix $(J_{ij})_{N\times N}$, where J_{ij} indicates the coupling between spin i and spin j. A feasible solution is represented by a vector σ of length N, in which σ_i takes the value from $\{1, -1\}$. The goal is to minimize the objective function

$$h(\sigma) = -\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij} \times \sigma_i \times \sigma_j. \tag{3}$$

1) *i-Mutation for Ising Spin Glass Problem:* Following the sketch in Section III-A, the i-mutation for the Ising spin glass problem is described as follows. First, the objective function is rewritten as

$$h(\sigma) = \sum_{i=1}^{N}(-\frac{1}{2N}\sum_{j=1}^{N}\frac{J_{ij}+J_{ji}}{2} \times \sigma_i \times \sigma_j). \tag{4}$$

By introducing the vector w, the weighted objective function is given by

$$h^w(\sigma) = \sum_{i=1}^{N} w_i \times (-\frac{1}{2N}\sum_{j=1}^{N}\frac{J_{ij}+J_{ji}}{2} \times \sigma_i \times \sigma_j). \tag{5}$$

[2]Otherwise, if the instance mapping function is not available, modifications have to be made to local search operators by changing the objective function from f to f^w, which is similar to the strategies in guided local search [39].

Algorithm 1: *i-mutation* for Ising Spin Glass

Input: interaction matrix J, number of spins N, i-mutation-portion η, i-mutation-strength μ
Output: perturbed interaction matrix J'
begin
  for *each spin i* **do** $w_i \leftarrow 1$;
  Randomly select $N \times \eta$ spins as the candidate;
  for *each spin i in the candidate* **do**
    $w_i \leftarrow 1 - \mu$;
  for $i \leftarrow 1$ *to* N **do**
    for $j \leftarrow 1$ *to* N **do**
      $J'_{ij} \leftarrow \frac{1}{2}(w_i + w_j) \times J_{ij}$;
  return J';
end

Algorithm 2: *i-ruin-recreate* for Ising Spin Glass

Input: interaction matrix J, number of spins N, i-remove-portion θ
Output: perturbed interaction matrix J'
begin
  $J' \leftarrow J$;
  Randomly select $N \times \theta$ spins as the candidate;
  Delete the columns and the rows of J', with respect to the candidate set;
  return J';
end

Algorithm 3: *i-crossover* for Ising Spin Glass

Input: interaction matrix J, number of spins N, feasible solution $s1$, feasible solution $s2$, i-crossover-strength τ
Output: perturbed interaction matrix J'
begin
  for *each spin i* **do**
    if $s1_i = s2_i$ **then** $w_i \leftarrow 1$;
    else $w_i \leftarrow 1 - \tau$;
  for $i \leftarrow 1$ *to* N **do**
    for $j \leftarrow 1$ *to* N **do**
      $J'_{ij} \leftarrow \frac{1}{2}(w_i + w_j) \times J_{ij}$;
  return J';
end

Then, the perturbed instance can be derived by

$$J'_{ij} = \frac{w_i + w_j}{2} J_{ij}. \tag{6}$$

Given a solution s, it is easy to verify that its objective value over the perturbed instance J' equals its weighted objective value over the original instance J [15]. Thus, with the weight vector w, the perturbed instance could be derived easily. The pseudo code for the LLH is presented in Algorithm 1.

2) *i-Ruin-Recreate for Ising Spin Glass Problem:* The motivation is very simple. The search landscape of the original instance is perturbed by partially destroying the local parts of the instance, over which the former local optimum may not be locally optimal anymore. Thus, local search could be applied to proceed the search. The pseudo code is presented in Algorithm 2.

3) *i-Crossover for Ising Spin Glass Problem:* This operator is very similar with i-mutation, except for the candidate set of the weight vector for perturbation. Instead of randomly selecting the elements with predetermined number, in i-crossover, the candidate set is adaptively selected. The perturbation is only conducted over the variables (local parts) that are different between the two parent solutions. By this strategy, the output instance of the LLH take the information of both the parents into account. The pseudo code is given in Algorithm 3.

C. IPLLHs for p-Median Problem

As a basic problem in location theory, the p-median problem has attracted much research attention in combinatorial optimization. In this paper, we focus on the symmetric p-median problem, which has been proved to be NP-hard [42].

Given a set F of m facilities, and an $m \times m$ matrix D with the traversing distance d_{ij} between facility i and facility j, for all $i, j \in F$. The objective of the p-median problem is to minimize the sum of these distances

$$g(x) = \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} \times x_{ij} \tag{7}$$

subject to

$$\sum_{j=1}^{m} x_{ij} = 1, i \in \{1, 2, \ldots, m\} \tag{8}$$

$$\sum_{j=1}^{m} y_j = p \tag{9}$$

$$x_{ij} \leq y_j, i, j \in \{1, 2, \ldots, m\} \tag{10}$$

$$x_{ij}, y_j \in \{0, 1\}, i, j \in \{1, 2, \ldots, m\}. \tag{11}$$

In the formulations, a solution is represented by a matrix $X_{m \times m}$, as well as a vector Y_m indicating the chosen facilities, where $x_{ij} = 1$ means that the facility i is assigned to the facility j, and $x_{ij} = 0$ otherwise; $y_j = 1$ indicates that the facility j is selected as a median, and $y_j = 0$ otherwise. Constraints (8) and (10) mean that each facility is allocated to only one median. Constraint (9) ensures that p facilities are selected as the medians, and Constraint (11) presents the integer conditions.

The IPLLHs for the p-median problem are as follows.

Algorithm 4: i-mutation for p-Median

Input: distance matrix D, number of facilities m, solution s, i-mutation-portion η, i-mutation-strength μ
Output: perturbed distance matrix D'
begin
  for *each facility* i **do** $w_i \leftarrow 1$;
  Randomly select $m \times \eta$ facilities as the candidate;
  for *each facility* i *in the candidate* **do**
    $w_i \leftarrow 1 - \mu$;
  for $i \leftarrow 1$ *to* m **do**
    for $j \leftarrow 1$ *to* m **do**
      $D'_{ij} \leftarrow \frac{1}{2}(w_i + w_j) \times D_{ij}$;
  return D';
end

Algorithm 5: *i-ruin-recreate* for p-Median

Input: distance matrix D, number of facilities m, i-remove-portion θ
Output: perturbed distance matrix D'
begin
  $D' \leftarrow D$;
  Randomly select $m \times \theta$ facilities as the candidate;
  Delete the columns and the rows of D', with respect to the candidate set;
  return D';
end

1) *i-Mutation for p-Median Problem:* Following the sketch in Section III-A, the i-mutation for the p-median problem is described as follows.

First, the objective function is rewritten as

$$g(x) = \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{d_{ij} + d_{ji}}{2} \times x_{ij} \tag{12}$$

subject to Constraints (8)–(11).

By introducing the vector w, the weighted objective function is given by

$$g^w(x) = \sum_{i=1}^{m} w_i \times \sum_{j=1}^{m} \frac{d_{ij} + d_{ji}}{2} \times x_{ij}. \tag{13}$$

Then, the perturbed instance can be derived by

$$D'_{ij} = \frac{w_i + w_j}{2} D_{ij}. \tag{14}$$

Similar to the i-mutation for the Ising spin glass problem, the verification is straightforward for the p-median problem. The pseudo code for the LLH is presented in Algorithm 4.

2) *i-Ruin-Recreate for p-Median Problem:* The main motivation behind i-ruin-recreate for the p-median problem is similar with that for the Ising spin glass problem, i.e., perturbing the search landscape of the original instance by partially destroying the local parts of the instance. The pseudo code is presented in Algorithm 5.

Algorithm 6: *i-crossover* for *p*-Median

Input: distance matrix D, number of facilities m, feasible solution $s1$, feasible solution $s2$, i-crossover-strength τ
Output: perturbed distance matrix D'
begin
  for *each facility* i **do**
    if *facility* i *is a median in both* $s1$ *and* $s2$ **then**
      $w_i \leftarrow 1 - \tau$;
    else $w_i \leftarrow 1$;
  for $i \leftarrow 1$ *to* m **do**
    for $j \leftarrow 1$ *to* m **do**
      $D'_{ij} \leftarrow \frac{1}{2}(w_i + w_j) \times D_{ij}$;
  return D';
end

3) *i-Crossover for p-Median Problem:* The main idea behind the i-crossover for the p-median problem is that, if a facility i is selected as a median for both the two parents, it is highly possible that this facility might contribute to high quality solutions. Thus, by decreasing the corresponding weight, this facility may be selected as a median during the local search over the offspring solution with higher probability. Otherwise, the corresponding weight remains unchanged. The pseudo code is presented in Algorithm 6.

IV. HIP-HOP Framework

In this section, we present the HIP-HOP framework. Following the hierarchy of most other hyper-heuristics [2], [21], HIP-HOP is designed as a two-layered framework, which consists of the domain-independent high level strategy and the domain-specific modules. The framework diagram is illustrated in Fig. 2. In the figure, the modules are represented by rounded rectangles, and the arrows indicate the interactions between the modules. In particular, the functionalities of the main components are described as follows.

1) The population consists of a set of LLH sequences. Each sequence is associated with a solution, over which the LLHs are applied. Meanwhile, the fitness of the LLH sequence is evaluated using the objective value of the corresponding solution.
2) The initialization module is used to construct the initial LLH sequences.
3) The high-level-mutation module is used to modify the LLH sequences, to make the search more exploratory.
4) The selection module is used to select the LLH sequences for the next generation. In this paper, the binary tournament selection is adopted.
5) The IPLLHs are used to construct the perturbed instances.
6) The intensifying LLHs are mainly local search operators.

With each module presented, we proceed to describe the interactions between these components in Fig. 2, to explain how HIP-HOP works. Meanwhile, following the diagram, the pseudo code of HIP-HOP is presented in Algorithm 7.

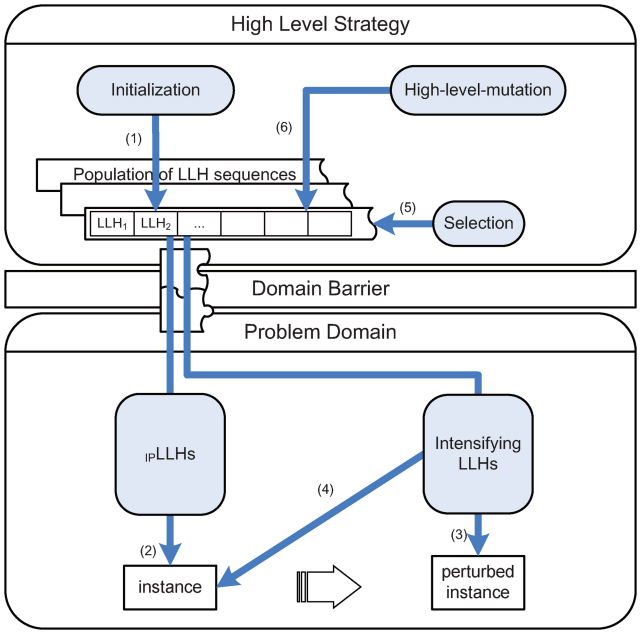


Fig. 2. HIP-HOP framework diagram.

During the initialization phase, a set of LLH sequences are generated by the Initialization module (step 1). After that, the LLHs in each sequence are executed for problem solving. Similar with existing work [4], we adopt the design in which the diversifying LLHs and intensifying LLHs are applied consecutively. However, in this paper, the diversification mechanism is realized by applying intensifying LLHs over perturbed instances. More specifically, in each diversification-intensification cycle (steps 2–4), a perturbed instance is firstly constructed in step 2, by applying an IPLLH over the original instance. Then, in step 3, an intensifying LLH is applied over the perturbed instance. In step 4, the instance is switched back to the original one, and an intensifying LLH is executed again to improve the solution quality. When all the LLHs of a sequence are conducted, the objective value of the associated solution is used to evaluate the quality of the LLH sequences. After all the LLH sequences are evaluated, the LLH sequences of the next generation are selected (step 5). Besides, to make the search over the heuristic space more exploratory, we also apply High-level-mutation to modify the LLH sequences (step 6). The iteration continues (steps 2–6), until certain stopping criterion is met. Examples of stopping criteria include the maximum number of local search executions, the maximum cut off time, etc. (see Sections VI-C and VI-D).

Note that the pseudo code described in Algorithm 7 illustrates a generic framework. To instantiate the algorithm for problem solving, several modules have to be implemented, such as initialization, high-level-mutation, and various domain-specific LLHs. More importantly, several constraints (e.g., the feasibility of the output solution and the consecutive execution of diversifying LLHs and intensifying LLHs) have to be satisfied within the modules. In the following sections, we shall discuss how to develop these modules, so that the HIP-HOP framework could be applied to different problem domains.

Algorithm 7: HIP-HOP

Input: problem instance π, number of individuals *pop-size*, maximum derivation depth *dep*
Output: optimized solution *s*
begin
  `// Initialization, step (1) in Fig. 2`
  Randomly initialize a population of LLH sequences;
  for *each LLH sequence of the population* **do**
    Randomly initialize an associated solution;
  `// Main loop`
  while *stopping criterion not met* **do**
    for *each LLH sequence* ***seq*** *of the offspring population* **do**
      Assign the associated solution from the parent population;
      for *each LLH of* ***seq*** **do**
        switch *type of the current LLH* **do**
          case *IPLLH*
            `// step (2) in Fig. 2`
            Apply IPLLH to generate a perturbed instance;
          case *intensifying LLH*
            if *the current instance is a perturbed instance* **then**
              `// step (3) in Fig. 2`
              Apply intensifying LLH over the perturbed instance;
            else
              `// step (4) in Fig. 2`
              Apply the intensifying LLH over the original instance;
      Evaluate **seq**;
    `// step (5) in Fig. 2`
    Select the survival LLH sequences;
    `// step (6) in Fig. 2`
    Modify the offspring population LLH sequences using High-level-mutation;
  return the best solution *s* achieved by the algorithm;
end

$$
\begin{aligned}
\langle loop \rangle &\rightarrow \langle divers \rangle \cdot \langle intens \rangle \cdot \langle loop \rangle \mid \epsilon && (1)\\
\langle intens \rangle &\rightarrow localsearch_1 \mid localsearch_2 \mid \ldots && (2)\\
\langle divers \rangle &\rightarrow \langle_{IP} LLH \rangle \cdot transfer \cdot \langle intens \rangle \cdot repair && (3)\\
\langle_{IP} LLH \rangle &\rightarrow i\text{-}mutation \mid i\text{-}crossover \mid i\text{-}ruin\text{-}recreate \mid \ldots && (4)
\end{aligned}
$$

Fig. 3. Production rules for the HIP-HOP framework.

V. Grammar Guided High Level Strategy

In this section, we discuss the development issues of the high level strategy. In this paper, the high level strategy is inspired by the linear genetic programming-based hyper-heuristic framework [29], in which a grammar guided high level strategy is incorporated.

The reasons we choose the grammar guided high level strategy are twofold. First, most existing instance perturbation based algorithms employ certain prescheduled schemes that control the degree/strength of the perturbation [13], [17], [19], [15]. For example, in search space smoothing [13], there is an order parameter that controls the ruggedness of the perturbed landscape. During the early stage, the search is conducted over a smooth landscape. As the search proceeds, the order parameter gradually decreases, and the landscape becomes more and more rugged, toward the original landscape. Contrarily, hyper-heuristics usually employ mutation or crossover to modify the LLH sequences [4], [8]. Without certain maintenance strategies, it is difficult to guarantee that the final solution obtained is a feasible solution to the original instance. With a grammar to guide the initialization and modification routines, however, the constraints could be implicitly satisfied [28]. Second, with the grammar, it would be possible that IPLLHs and the existing SPLLHs could be incorporated into a unified framework. Hence, this design enables the fair comparisons between the two variants.

Following the notation of linear genetic programming [29], we define the grammar as a tuple $G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$. In the definition, $\mathcal{N}$ represents the set of nonterminals, $\mathcal{T}$ represents the terminal set, $\mathcal{P}$ represents the production rules, and $\mathcal{S}$ represents the start symbol. A minor difference from the grammar in [29] is that, in this paper, all the terminals indicate LLHs (e.g., local search operators). While in [29], the terminals are primitives extracted from LLHs (e.g., 2-CHANGE neighborhood move from the 2-opt local search operator for the traveling salesman problem). By raising the granularity of the abstraction, we intend to incorporate instance perturbation based techniques into hyper-heuristics, meanwhile keep the framework general for cross domain problem solving.

In the grammar G, the nonterminal set is defined as $\mathcal{N} = \{\langle loop \rangle, \langle divers \rangle, \langle intens \rangle, \langle_{IP} LLH \rangle\}$. The terminal set $\mathcal{T}$ consists of various LLHs. The production rules employed in this paper are described in the Backus-Naur Form, as shown in Fig. 3, and the start symbol is represented by $\langle loop \rangle$. In particular, the production rules are explained as follows.

1) Production (1) indicates that, the LLH sequences constructed by the grammar reflect the main characteristic of the metaheuristic, which is defined as iterative generation process, which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [43]. More specifically, the tail recursion in the rule implies that the sequence is executed in an iterative paradigm. Meanwhile, within each iteration, LLHs that realize the diversification mechanism and the intensification mechanism are applied consecutively.
2) Production (2) implies that, the intensification mechanism is mostly carried out by local search operators in this paper.

Algorithm 8: High-level-mutation

Input: Grammar G, LLH sequence *seq*, high level mutation rate α
Output: modified LLH sequence seq'
begin
  $seq' \leftarrow seq$;
  for *each* LLH_i *of* seq' **do**
    if LLH_i *is a local search operator* **then**
      Reselect LLH_i with respect to Production (2) of G, with probability α;
    if LLH_i *is an* $_{IP}LLH$ **then**
      Reselect LLH_i with respect to Production (4) of G, with probability α;
end
return seq';

3) Production (3) means that, the diversification mechanism in HIP-HOP is realized following the scheme of most instance perturbation based techniques. First, the original instance is perturbed with an $_{IP}$LLH. After that, the solution is transferred into the solution space derived by the perturbed instance, and intensifying LLHs are carried out over the perturbed instance. Since the goal of the framework is to obtain an optimal or near-optimal solution to the original instance, the solution should then be repaired so that it is feasible in the original search space. To this point, the solution might have escaped from the local optimum, so that local search could be applied to improve its quality.
4) Production (4) lists the domain specific $_{IP}$LLHs, which we have discussed in detail in Section III.

Given the grammar G, LLH sequences represented by sentences of G could be easily derived. More specifically, the derivation begins from the start symbol $\langle loop \rangle$. At each iteration, the left most nonterminal is replaced with the right side of the corresponding production rules. If there are more than one rule with respect to the nonterminal, random selection is applied. By setting a constraint on the maximum depth of Production (1), the derivation could be prevented from being conducted infinitely. Meanwhile, we should note that, the maximum derivation depth *dep* does not imply that each solution is only optimized for only *dep* iterations. Instead, it means that each LLH sequence is evaluated and selected every *dep* diversification-intensification cycles. At the subsequent iteration, the search is continued by applying the survival LLH sequences over their corresponding solutions. Fig. 4 depicts an example of the LLH sequence construction, with maximum depth of 2. When a sentence is derived, it can be observed that the LLHs could be executed sequentially. We also develop a simple high-level-mutation module over the LLH sequences, so that the search over the heuristic space could be more exploratory. The pseudo code of high-level-mutation is presented in Algorithm 8. For each LLH of a sequence, if there exists an alternative LLH of the same functionality [such as $_{IP}$LLHs of Production (4) and intensifying LLHs of Production (2)], the corresponding LLH is randomly reselected with a probability α. Since the mutation is conducted under the guidance of the grammar G, it is obvious that solution feasibility could be easily maintained.

$$
\begin{aligned}
&\langle loop \rangle \\
&\rightarrow \langle divers \rangle \cdot \langle intens \rangle \cdot \langle loop \rangle \\
&\rightarrow \langle {}_{IP}LLH \rangle \cdot transfer \cdot \langle intens \rangle \cdot repair \cdot \langle intens \rangle \cdot \langle loop \rangle \\
&\rightarrow i\text{-}mutation \cdot transfer \cdot \langle intens \rangle \cdot repair \cdot \langle intens \rangle \cdot \langle loop \rangle \\
&\rightarrow i\text{-}mutation \cdot transfer \cdot localsearch_2 \cdot repair \cdot \langle intens \rangle \cdot \langle loop \rangle \\
&\rightarrow i\text{-}mutation \cdot transfer \cdot localsearch_2 \cdot repair \cdot localsearch_1 \cdot \langle loop \rangle \\
&\rightarrow \cdots \\
&\rightarrow i\text{-}mutation \cdot transfer \cdot localsearch_2 \cdot repair \cdot localsearch_1 \cdot \\
&\quad i\text{-}crossover \cdot transfer \cdot localsearch_1 \cdot repair \cdot localsearch_1
\end{aligned}
$$

Fig. 4. Illustration of derivation procedure.

$$
\begin{aligned}
\langle divers \rangle &\rightarrow \langle {}_{SP}LLH \rangle && (5) \\
\langle {}_{SP}LLH \rangle &\rightarrow mutation \mid \\
&\quad\ crossover \mid \\
&\quad\ ruin\text{-}recreate \mid && (6) \\
&\quad\ \cdots
\end{aligned}
$$

Fig. 5. Production rules for $_{SP}$LLHs.

By embedding the grammar guided initialization and high-level-mutation into Algorithm 7, the functionalities of the high level strategy part could be realized. In the proposed framework, the grammar plays the most important role, from two aspects. First, during the initialization phase, the LLH sequences are derived with respect to the grammar. Second, the grammar is used to guide the high-level-mutation routine. The features of the grammar guided high level strategy could be summarized as follows. First, by using a grammar, the solution feasibility could be implicitly satisfied. Second, the grammar also reflects the characteristics of metaheuristics, i.e., the balance between the intensification and the diversification [12], [43]. Third, interestingly, a byproduct of the grammar guided framework is that, by modifying the grammar [e.g., by replacing the Productions (3) and (4) in Fig. 3 with Productions (5) and (6) in Fig. 5], the diversification mechanisms could be provided by the traditional $_{SP}$LLHs, such as mutation and crossover. In this sense, HIP-HOP could easily be transformed to SOPHY. Consequently, $_{IP}$LLHs and $_{SP}$LLHs could be described under a unified framework. Also, it would be convenient to conduct fair comparisons and analysis about the two variants.

VI. Empirical Study

A. Preliminaries

All the experiments in this paper are performed on a Pentium IV 3.2 GHz PC with 4GB memory, running GNU/Linux with kernel 3.2.0. All the codes are implemented in C++, compiled using g++ 4.6. The running time is measured in seconds. To examine the generality of the algorithms, we consider the Ising spin glass problem and the p-median problem as two case studies. For each problem domain, 60 benchmark instances[3] are employed to evaluate various properties of the

[3]http://oscar-lab.org/people/~zren/hip-hop/

proposed framework. For the Ising spin glass problem, we focus on the 2-D instances with Gaussian interactions. In particular, we employ rudy,[4] which is a publicly available instance generator for the Ising spin glass problem, with the parameter—spinglass2g. We generate 60 Ising spin glass instances with the number of variables ranging from 225 to 400 (e.g., 15×15-* indicates the instances with 225 spins, which are placed on a 15×15 grid).

On the other hand, for the p-median problem, we focus on the symmetric instances. The instances include 40 graph-based instances from ORLIB [44], and 20 Euclidean-based instances from TSPLIB [45]. The TSPLIB instances are first introduced in the context of the p-median problem in [10], and have been widely used to test the performance of various algorithms. For each instance, every point represents a facility, and the distance between any two facilities is the Euclidean distance between the two points. In this paper, the distance matrix are derived from two TSPLIB instances, i.e., fl1400 and fl1577, with varying values of p from 50 to 500.

In this paper, the comparative results are drawn from two sources, i.e., SOPHY (see Section V) and the best known results. The reason we choose these two forms of baseline results are twofold. First, SOPHY serves as a baseline for comparison, in that this variant differs from HIP-HOP only in the LLH perspective. By comparing HIP-HOP with SOPHY, we would be able to examine various aspects of the IPLLHs. Second, we employ the best known results to evaluate the performances of HIP-HOP objectively. The detailed information about these baseline results are introduced as follows.

As the first baseline algorithm, SOPHY has the same high level strategy as HIP-HOP, which is a grammar guided framework. Meanwhile, for each category of LLH, a typical operator is incorporated in SOPHY. The LLHs in SOPHY include the following.

1) Local search for the Ising spin glass problem. In this paper, the discrete hill climber with the flipping neighborhood described in [46] is employed as the intensifying LLH. This LLH is also used in HIP-HOP.
2) Mutation for the Ising spin glass problem. In this paper, the bit-flip mutation is adopted. In this LLH, each spin is randomly flipped with probability indicated by a parameter mutation-rate.
3) Ruin-recreate for the Ising spin glass problem. In this LLH, several spins are randomly selected, for which the values are randomly assigned. The number of the spins to be ruined is shake-strength $\times$ N, where N indicates the total number of spins, and shake-strength is a parameter.
4) Crossover for the Ising spin glass problem. In this paper, the uniform crossover is adopted, i.e., the value of each spin of the offspring solution is randomly assigned from one of its parent solution.
5) Local search for the p-median problem. In this paper, interchange is employed as the local search operator. In particular, the implementation in [47] is adopted, because of its high efficiency. However, we should note that in this LLH, a preprocessing routine has to be conducted so that for each facility i, all the other facilities j are ranked with respect to its distance to facility i. This LLH is also used in HIP-HOP.
6) Mutation for the p-median problem. This LLH is extracted from [48]. In particular, for mutation, each median is swapped with a random nonmedian facility with a probability indicated by mutation-rate.
7) Ruin-recreate for the p-Median Problem. Proposed in [10], shake can be viewed as a special case of ruin-recreate. This operator use a parameter to represents the distance (shake-strength $\times m$, indicated by the number of different medians) between the input solution and the output solution.
8) Crossover for the p-median problem. This LLH is extracted from [48] as well, which is a uniform crossover.

[4]http://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz

TABLE I
HIGH LEVEL STRATEGY PARAMETER CONFIGURATIONS

Parameter	Value
Maximum derivation depth *dep*	3
Population size *pop-size*	20
Maximum number of local search *num*	10000
High level mutation rate α	0.1

Besides SOPHY, we also consider the best known results in the literature to evaluate the performances of HIP-HOP and SOPHY. In particular, the best known results may be provided by global optima, best known solutions in the literature, or solutions obtained by the state-of-the-art heuristic solvers, based on the problem domain and the type of the instances. For each Ising spin glass instance, we obtain the global optima using the spin glass ground state server.[5] Meanwhile, among the p-median instances, the ORLIB instances have been exactly solved [44]. Hence, the optimal objective values are suitable for the evaluation of HIP-HOP. For the TSPLIB instances, the global optima are not available, due to the scale of the instances. More specifically, the instances derived from fl1400 have been studied in [10], [49], and [37], thus, the best known objective values for these instances could be employed as the baseline results. Accordingly, since fl1577 is used to derive p-median instance for the first time, we adopt the state-of-the-art heuristic solvers we could obtain to achieve the baseline results over these instances. The solvers include POPSTAR,[6] which originates from [49], as well as ALCMA [37]. For each fl1577 instance, we run both POPSTAR and ALCMA for 30 times, with their default parameter configurations, to obtain the baseline results.

B. Parameter Setting

In the proposed framework, there are two categories of parameters, i.e., the parameters in high level strategy and those in LLHs. In this paper, we choose to fix the parameters in the high level strategy, meanwhile use an automated parameter tuning tool for the parameter settings of the LLHs.

[5]http://www.informatik.uni-koeln.de/ls_juenger/projects/sgs.html
[6]http://www2.research.att.com/~mgcr/popstar/popstar.html

TABLE II
LLH PARAMETER CONFIGURATIONS FOR HIP-HOP

Problem domain	Parameter	Value	Range
Ising Spin Glass	*i-mutation-portion*	0.7150	[0.1, 0.9]
	i-mutation-strength	1.0920	[0.1, 1.9]
	i-remove-portion	0.4254	[0.1, 0.9]
	i-crossover-strength	0.1565	[0.1, 1.9]
p-Median	*i-mutation-portion*	0.1638	[0.1, 0.9]
	i-mutation-strength	0.1046	[0.1, 0.9]
	i-remove-portion	0.1254	[0.1, 0.9]
	i-crossover-strength	0.3378	[0.1, 0.9]

TABLE III
LLH PARAMETER CONFIGURATIONS FOR SOPHY

Problem domain	Parameter	Value	Range
Spin Glass	*mutation-rate*	0.2292	[0.1, 0.9]
	shake-strength	0.4387	[0.1, 0.9]
p-Median	*mutation-rate*	0.8668	[0.1, 0.9]
	shake-strength	0.2690	[0.1, 0.9]

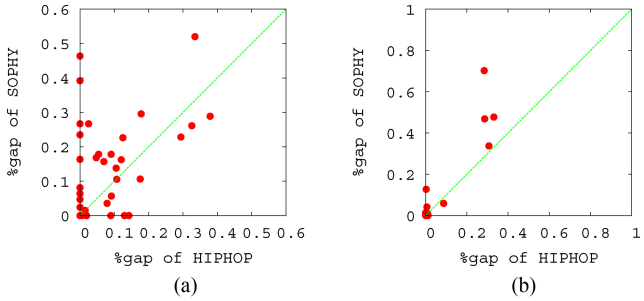


Fig. 6. *%gap* comparison between HIP-HOP and SOPHY. (a) Ising Spin Glass instances. (b) p-Median instances.

The reasons for this parameter configuration scheme are twofold. On the one hand, the high level strategy parameters are assigned with the same values for both HIP-HOP and SOPHY so that the comparisons could be conducted in a fair way. For example, the high level strategy parameters, such as the population size and maximum derivation depth of HIP-HOP are set with the same values as SOPHY, for the convenience of comparison. Especially, we shall note that the parameter *num* in HIP-HOP takes the local search executions over both the original instance and the perturbed instance into account, so that the search effort in HIP-HOP and SOPHY are the same. On the other hand, for the LLH parameters, we prefer the offline tuning to the online control paradigm, in that we are more interested in the behavior of the LLHs and their parameters as whole components, rather than the impact of the LLH parameters on the framework. Besides, it would be more difficult for experiments design and analysis, if the online parameter control strategies is incorporated. More specifically, we use irace to obtain the LLH parameter configurations for both HIP-HOP and SOPHY. irace is an R implementation of iterated F-Race [50], which is a well known offline parameter tuning package.[7]

We adopt the default configurations of irace, except for setting the maximum number of local search in HIP-HOP and SOPHY to be 1000, so that the total tuning time is acceptable. After the tuning process, the LLH parameter configurations are presented in Tables II and III. In both tables, the first column specifies the problem domain. The second column presents the names of the LLH parameters. Then, the values and the feasible ranges of the parameters are given in columns 3 and 4, respectively. Note that for the Ising spin glass problem, the ranges of i-mutation-strength and i-crossover-strength are [0.1,1.9], in that the interactions between spins could be negative. Besides these two parameters, all the other parameters lie within the range [0.1,0.9]. Through preliminary experiments,[8] we find that irace is able to obtain parameter configurations that are robust and effective.

C. Effectiveness Evaluation

In this subsection, we first evaluate the solution quality that HIP-HOP and SOPHY could achieve, given the same amount of search effort. After that, we investigate the dynamic behavior of the two algorithms, by examining the trend of the average error along the search procedure, over several representative instances.

For HIP-HOP and SOPHY, the parameters are assigned with respect to Tables I–III, respectively. The instances are specified in Section VI-A. Over each instance, HIP-HOP and SOPHY are executed for 30 independent runs, and the comparisons between the two algorithms are illustrated in Figs. 6 and 7.

We first present the relative comparison between HIP-HOP and SOPHY, in terms of the solution quality obtained by the algorithms. More specifically, we employ the relative gap and the average error as the performance measurements, which are defined as

$$\%gap = \frac{C_{best} - C_{opt}}{|C_{opt}|} \times 100 \tag{15}$$

$$\%err = \frac{C_{avg} - C_{opt}}{|C_{opt}|} \times 100 \tag{16}$$

where C_{best} and C_{avg} indicate the best and the average objective value obtained by each algorithm within the multiple independent executions, and C_{opt} represents the best known objective value.

Taking Fig. 6(a) as an example, we shall now discuss the performance comparison between HIP-HOP and SOPHY over Ising Spin Glass instances. In the subfigure, the x-axis and the y-axis indicate the *%gap* of HIP-HOP and SOPHY, respectively. More specifically, each point (x, y) in the subfigure indicates that there are one or more instances over which HIP-HOP's *%gap* and SOPHY's *%gap* are x and y, respectively. For clarity, we plot the reference line $y = x$. Accordingly, a point above the line implies that over the corresponding instance(s), HIP-HOP outperforms SOPHY, since HIP-HOP obtains smaller *%gap*.

From Fig. 6, several interesting phenomena could be observed. First, in Fig. 6(a), most points lie above the reference line, which implies that over the Ising spin glass instances, the best solutions obtained by HIP-HOP are better than those

[7] http://iridia.ulb.ac.be/irace/

[8] The results of the preliminary experimental analysis could be found at http://oscar-lab.org/people/~zren/hip-hop/.

TABLE IV
NUMERICAL RESULTS OVER TSPLIB p-MEDIAN INSTANCES

instance	best known	reference	SOPHY best	SOPHY MED	SOPHY $\%err$	SOPHY SD	SOPHY time	HIP-HOP best	HIP-HOP MED	HIP-HOP $\%err$	HIP-HOP SD	HIP-HOP time
fl1400, p = 350	5719.03	[4]	5718.01	5719.91	0.0519	1.38	91.08	**5717.86**	5722.37	0.0916	1.95	701.78
fl1400, p = 450	4468.29	[37]	4476.57	4479.92	0.2671	2.29	112.70	**4468.28**	4468.30	0.0005	0.02	777.43
fl1400, p = 500	4046.39	[4]	4046.17	4047.17	0.0842	1.14	115.73	**4044.10**	4044.78	0.0175	0.51	742.69

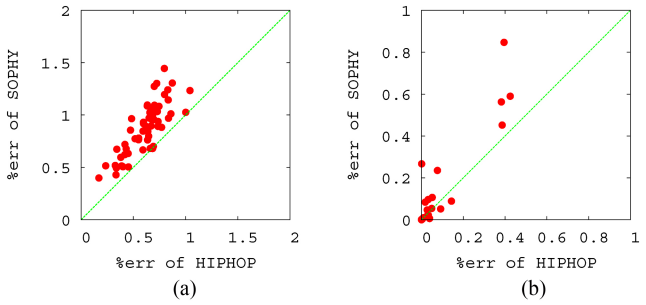


Fig. 7. $\%err$ comparison between HIP-HOP and SOPHY. (a) Ising Spin Glass instances. (b) p-Median instances.

of SOPHY, in that over the majority of the Ising spin glass instances, the relative gaps achieved by HIP-HOP tend to be less than those of SOPHY. Second, in Fig. 6(b), it can be observed that the points are more sparse. The reason is that the 40 ORLIB instances used in the experiments are relatively easy, which could not tell the difference between the two algorithms. For example, the origin of the figure represents the comparison results over many ORLIB instances.

In particular, HIP-HOP is able to achieve better solutions than the best known results in the literature over three p-median benchmark instances. Hence, we report the detailed results over these instances in Table IV.[9] In the table, the first column indicates the three instances. Columns 2–3 present the best known objective values in the literature, as well as the sources where the results are drawn. Then, the results for SOPHY and HIP-HOP are given in columns 4–8 and columns 9–13, respectively. For each algorithm, the results consist of the best objective value among the 30 independent executions, the median objective value (indicated by MED), the average error (indicated by $\%err$), the standard deviation (indicated by SD), as well as the average execution time (measured in seconds). From Table IV, we could observe that the best results obtained by HIP-HOP are better than the best known results in the literature. However, we can see that HIP-HOP is slower than SOPHY, due to the fact that in IPLLHs, extra subroutines are usually required to construct perturbed instances, and/or maintain data structures for the solutions (see Section VI-D for a more detailed discussion).

After comparing the best results obtained by HIP-HOP and SOPHY, we proceed to compare the average performances of the two algorithms, which is presented in Fig. 7. The organization of Fig. 7 is the same as Fig. 6, except that $\%gap$ is replaced by $\%err$. We could observe that, the comparison of

[9]The complete numerical results could be found at http://oscar-lab.org/people/~zren/hip-hop/.

TABLE V
p-VALUE OBTAINED BY PAIRED WILCOXON'S SIGNED RANK TEST, HIP-HOP VERSUS SOPHY

Problem domain	$\%gap$ comparison	$\%err$ comparison
Ising Spin Glass	0.0494	< 0.0001
p-Median	0.0171	0.0283

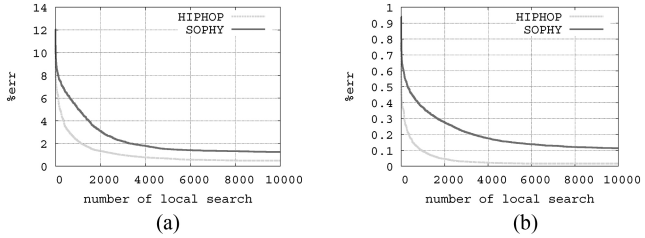


Fig. 8. Performance against number of local search executions. (a) Ising spin glass, 19×19-3. (b) p-Median, fl1400, p = 500.

$\%err$ exhibits similar trend as that of $\%gap$, i.e., HIP-HOP tends to obtain solutions with smaller average error than SOPHY, over the instances of both the Ising spin glass problem and the p-median problem.

Associated with the comparisons in Figs. 6–7, we also conduct statistical tests, to draw confident conclusions whether one algorithm outperforms the other. For the statistical test, we employ nonparametric statistical test to detect the potential difference between the performances of the two algorithms. More specifically, the paired Wilcoxon's signed rank test is employed, with a null hypothesis stating that there exists no difference between the results of the algorithms in comparison. We consider the 95% confidence level (i.e., p-values below 0.05 are considered statistically significant), unless otherwise stated. In Table V, we report the p-values returned by the Wilcoxon's test for both the $\%gap$ and $\%err$ comparisons. From Table V, we could observe that for the two problem domains, HIP-HOP outperforms SOPHY, in terms of both the relative gap achieved and the average error obtained. In each comparison scenario, the p-value returned by the Wilcoxon's test is always less than 0.05.

To gain more insight about the algorithms' behavior, we compare the solution quality trend as the search proceeds. Fig. 8 presents the $\%err$ versus the number of local search executions, for both HIP-HOP and SOPHY. In the figure, the x-axis indicates the number of local search conducted, and the y-axis represents the $\%err$ that an algorithm could achieve. It can be observed that over both instances, the curve corresponding with HIP-HOP always lies beneath that of SOPHY, which implies that HIP-HOP is able to achieve better

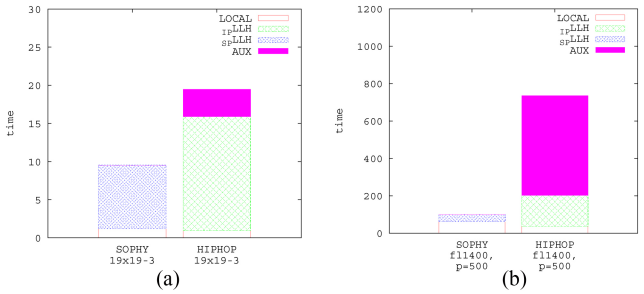


Fig. 9. Proportion of time consumed by different components. (a) Ising spin glass instance. (b) p-Median instance.

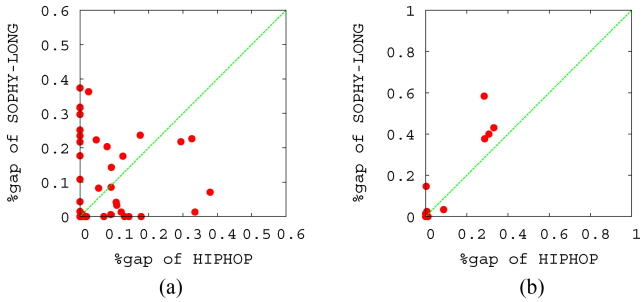


Fig. 10. *%gap* comparison, HIP-HOP versus SOPHY-LONG. (a) Ising spin glass instances. (b) p-median instances.

results than SOPHY, given the same number of local search executions. Besides, it can be observed that over all the representative instances, after 10 000 executions of local search, both HIP-HOP and SOPHY tend to have converged, which to some extent implies that the numerical results comparison between the two algorithms is reasonable.

D. Efficiency Evaluation

In Section VI-C, we observe that given the same amount of search effort (indicated by the number of local search executions), HIP-HOP is able to achieve better solution quality than SOPHY. However, due to the maintenance overhead introduced by $_{\text{IP}}$LLHs, HIP-HOP requires more time to terminate. Hence, in this subsection, we intend to examine the runtime behavior of both HIP-HOP and SOPHY. In more detail, we first investigate the reason why HIP-HOP is slow, by comparing the average time cost by each LLH related component over representative instances. Second, since SOPHY is faster than HIP-HOP, we proceed to check whether the comparison results in Section VI-C would change, if SOPHY is given more running time. Finally, we employ the run-time distribution analysis [51], [52] to investigate the runtime behaviors of the two algorithms.

Fig. 9 presents the average time that each component in the two algorithms spent over the representative instances. For example, Fig. 9(a) illustrates the comparison between HIP-HOP and SOPHY over the Ising Spin glass instance 19×19-3. The components include local search (indicated by LOCAL), $_{\text{IP}}$LLHs, $_{\text{SP}}$LLHs, as well as auxiliary routines (indicated by AUX, including the transfer operator and the repair operator described in Section V, as well as other data structure maintenance procedures). From Fig. 9(a), the following observations could be found over the spin glass instance 19×19-3. First, the time of local search in HIP-HOP is almost the same as that in SOPHY. The local search operator spends 0.94s and 1.18s in HIP-HOP and SOPHY, respectively. Second, $_{\text{IP}}$LLHs are more time consuming than $_{\text{SP}}$LLHs. $_{\text{IP}}$LLHs and $_{\text{SP}}$LLHs spend 14.94s and 8.35s in HIP-HOP and SOPHY, respectively. In total, HIP-HOP is more time consuming, but roughly of the same order of magnitude as SOPHY. Meanwhile, over the p-median instance fl1400 with p = 500, different observations could be drawn. 1) Surprisingly, over the p-median instance, local search costs less time in HIP-HOP than in SOPHY. In Fig. 9(b), local search consumes 36.89s and 61.84s in HIP-HOP and SOPHY, respectively. Although the $_{\text{IP}}$LLHs are time consuming, with these LLHs, local search could be conducted more efficiently. 2) $_{\text{IP}}$LLHs are more time consuming than $_{\text{SP}}$LLHs. This phenomenon mainly results from the complexity of these two types of LLHs. Over fl1400 with p = 500, $_{\text{IP}}$LLHs consume 206.08s in average, while $_{\text{SP}}$LLHs cost 43.32s in average. 3) Over the p-median instance, the auxiliary routines consumes the majority proportion of the time in HIP-HOP. For example, over fl1400 with p = 500, these routines costs 652.50s in average. The main reason for this observation is that for the p-median instances, whenever a perturbed instance is constructed, an auxiliary data structure has to be maintained so that the local search could be conducted efficiently [47]. The complexity of this maintenance routine is $\mathcal{O}(m^2 \log m)$. However, in SOPHY, this maintenance routine is only executed once during the preprocessing phase. As a result, the repeated executions of this routine lead to the fact that HIP-HOP is much slower than SOPHY over the p-median instances.

Thus, a question naturally arises whether SOPHY could outperform HIP-HOP, if given more search effort. To investigate this issue, a set of experiments are conducted as follows. Over each instance, we assign more search effort to SOPHY, so that the maximum cut off time of SOPHY equals the average running time of HIP-HOP. Note that since SOPHY is given longer execution time, the variant version of SOPHY is indicated by SOPHY-LONG in this experiment. Meanwhile, the results of HIP-HOP are kept the same as in Section VI-C. Through this comparison scheme, we could properly answer the question. The comparison results are organized in the same way as in Section VI-C, i.e., Figs. 10 and 11 present the visual comparisons between HIP-HOP and SOPHY-LONG, and Table VI lists the p-values of the Wilcoxon's test, in which the null hypothesis states that both algorithms in comparison have similar performances. From the comparison results, we observe that with more search effort, the performances of SOPHY could be improved over several instances. For example, when we compare the *%err* of HIP-HOP and SOPHY-LONG over p-median instances [see Fig. 11(b)], the statistical test confirms that HIP-HOP outperforms SOPHY-LONG. However, the confidence level decreases to 90% ($0.05 < p$-value< 0.1). HIP-HOP also performs better than SOPHY-LONG in other comparison scenarios, except when we compare the *%gap* of the two algorithms over Ising Spin Glass instances [see Fig. 10(a)]. In this case,

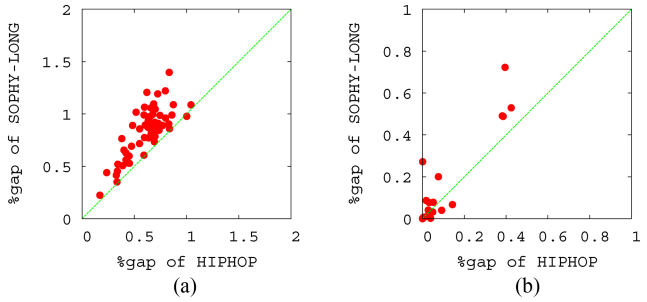


Fig. 11. *%err* comparison, HIP-HOP versus SOPHY-LONG. (a) Ising spin glass instances. (b) p-median instances.

TABLE VI
p-VALUE OBTAINED BY PAIRED WILCOXON'S SIGNED RANK TEST, HIP-HOP VERSUS SOPHY-LONG

Problem domain	$\%gap$ comparison	$\%err$ comparison
Ising Spin Glass	0.3633	< 0.0001
p-Median	0.0235	0.0885

the null hypothesis could not be rejected, which means that the two algorithms perform similarly over this problem domain. In summary, the statistical test implies that HIP-HOP compares favorably to, or at least similar as SOPHY, even if more search effort is assigned to SOPHY. To further investigate the dynamic characteristics of HIP-HOP and SOPHY, we would examine the efficiency characteristics of the two algorithms, by comparing the runtime behavior of HIP-HOP with that of SOPHY.

To achieve this, the runtime distribution analysis is introduced. More specifically, the analysis is conducted over two typical instances, i.e., 19×19-3 for the Ising spin glass problem, as well as fl1400 with $p = 500$ for the p-median problem, respectively. Given an instance, the runtime distribution is illustrated as follows. Each algorithm is executed for 100 independent trials. For both HIP-HOP and SOPHY, the parameters in the high level strategy are set with respect to Table I, except for the stopping criterion. In the runtime distribution analysis, the stopping criterion is changed from the maximum number of local search executions to the maximum cut off time (50s for the Ising spin glass problem, and 1000s for the p-median problem, respectively). Besides, the LLH parameters are set with respect to Tables II and III.

For each algorithm, its runtime distribution is represented by a set of cumulative probability distribution curves determined from the 100 runs of each algorithm. For example, Fig. 12(a) illustrates the runtime distribution curves of HIP-HOP and SOPHY over the Ising spin glass instance 19×19-3. In the plot, the x-axis specifies the runtime, the y-axis indicates the upper bound threshold, and the z-axis represents the probability that the algorithm achieves the corresponding solution quality threshold (denoted as p_{rtd}). For each algorithm, the point (x, y, z) in the plot indicates after time x, the algorithm could achieve the solution quality better than y above the best known upper bound with probability z.

Taking the p-median instance fl1400 with $p = 500$ as an example [see Fig. 12(b)], the following observations can be drawn. First, for different values of threshold, the runtime

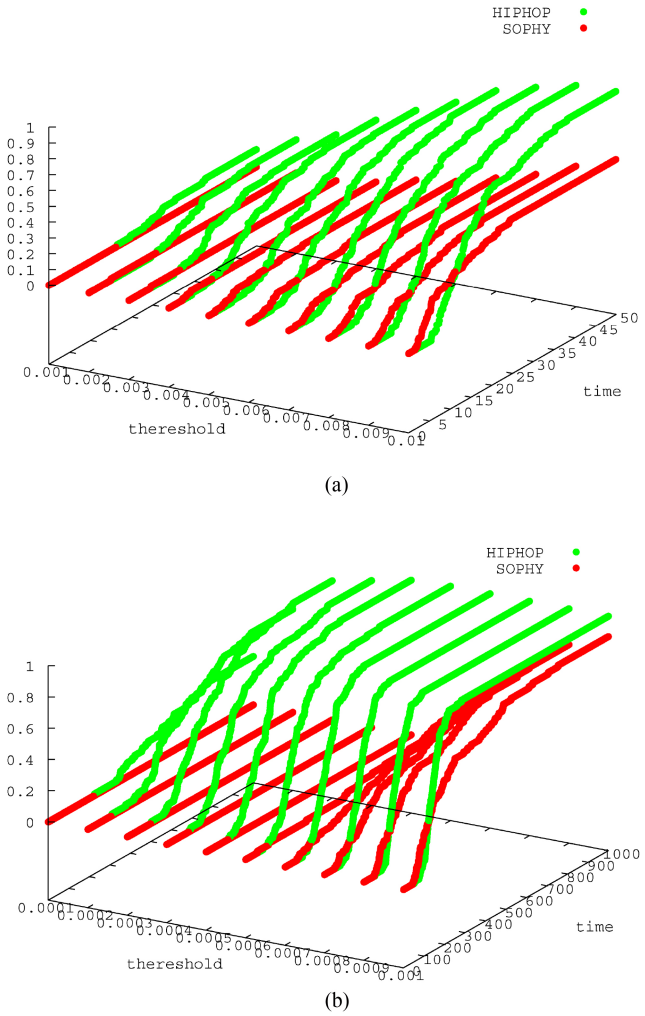


Fig. 12. Run-time distribution plots. (a) Ising spin glass, 19×19−3. (b) p-meidan, fl1400, $p = 500$.

distribution curves of HIP-HOP and SOPHY exhibit similar trend as time elapses. At the beginning of the search process, the runtime distribution curves of HIP-HOP lie below those of SOPHY. This phenomenon holds for most values of threshold. For example, when the value of threshold equals to 0.1%, p_{rtd} of HIP-HOP is always less than that of SOPHY before 90s. This implies that HIP-HOP converges slower than SOPHY, which is mainly due to the relatively high complexity of the IPLLHs and the auxiliary routines in HIP-HOP. However, as the search proceeds, HIP-HOP is able to achieve higher p_{rtd} compared with SOPHY. For example, over fl1400 with $p = 500$, after 122s, the p_{rtd} of HIP-HOP is always higher than SOPHY, when the value of threshold equals to 0.1%. Another interesting observation is that, even if the threshold is very small (e.g., 0.01%), p_{rtd} of HIP-HOP could still reach 31%, while p_{rtd} of SOPHY is 0. This to some extent demonstrates the effectiveness of the IPLLHs. Over the Ising spin glass instance 19×19-3, similar observations could also be obtained. Over these instances, HIP-HOP tends to less effective than SOPHY for the beginning of the search process. However, if given sufficient time, HIP-HOP is able to achieve better solution quality.

VII. Conclusion

In this paper, we systematically investigate how to incorporate the instance perturbation methodologies into hyper-heuristics. In particular, we propose the HIP-HOP framework, which combines the generality of hyper-heuristics and the potential ability of instance characteristics exploitation provided by the instance perturbation methodologies. The contributions of this paper could be summarized as follows.

1) We propose a set of instance perturbation-based low level heuristics ($_{\text{IP}}$LLHs), which are able to provide the diversification mechanisms, meanwhile provide an interface of exploiting the information of the problem instances.
2) We develop a grammar guided high level strategy. Unlike many existing instance perturbation-based methodologies [13], [17] in which the perturbation schedules are predetermined, with a simple grammar, the feasibility constraints of the output solution could be implicitly satisfied.
3) We demonstrate the generality of the HIP-HOP framework by the applications to two problem domains, i.e., the Ising spin glass problem and the p-median problem.
4) Extensive experiments validate the effectiveness of the framework. Comparisons with SOPHY, as well as the state-of-the-art results demonstrate that HIP-HOP is able to achieve competitive results over various benchmark instances.

Despite the promising results and the generality of the HIP-HOP framework, there are still several limitations within the framework, which deserve more future work. For example, in this paper, the complexity of the proposed $_{\text{IP}}$LLHs is relatively high, especially for the p-median problem. This drawback might be overcome by mechanisms, such as the surrogate model [53]. Meanwhile, in this paper, the LLH parameters are offline-tuned with an automatic tuning tool. In the future, we shall investigate the impact of the online-adaptation of the parameters [4]. Finally, we shall investigate the possibility of combining the $_{\text{IP}}$LLHs and the $_{\text{SP}}$LLHs, using methodologies, such as coevolution [54], hybridization, etc.

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evol. Comput.*, vol. 16, no. 1, pp. 31–61, 2008.
[2] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evol. Comput.*, vol. 20, no. 1, pp. 63–89, 2012.
[3] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 942–958, Dec. 2010.
[4] Z. Ren, H. Jiang, J. Xuan, and Z. Luo, "Hyper-heuristics with low level parameter adaptation," *Evol. Comput.*, vol. 20, no. 2, pp. 189–227, 2012.
[5] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2010, pp. 449–468.
[6] P. Cowling, G. Kendall, and L. Han, "An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem," in *Proc. IEEE Congr. Evol. Comput.*, vol. 2. May 2002, pp. 1185–1190.
[7] P.-C. Chen, G. Kendall, and G. Berghe, "An ant based hyper-heuristic for the travelling tournament problem," in *Proc. IEEE Symp. Comput. Intell. Schedul.*, Apr. 2007, pp. 19–26.
[8] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical evolution of local search heuristics," *IEEE Trans. Evol. Comput.*, vol. 16, no. 3, pp. 406–417, Jun. 2012.
[9] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, *et al.*, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Proc. Eur. Conf. Evol. Comput. in Combinatorial Optimisation*, 2012, pp. 136–147.
[10] P. Hansen and N. Mladenović, "Variable neighborhood search for the p-median," *Loc. Sci.*, vol. 5, no. 4, pp. 207–226, 1997.
[11] E. Özcan, B. Bilgin, and E. Korkmaz, "Hill climbers and mutational heuristics in hyperheuristics," in *Proc. PPSN*., Sep. 2006, pp. 202–211.
[12] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.
[13] J. Gu and X. Huang, "Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP)," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 5, pp. 728–735, May 1994.
[14] J. Schneider, M. Dankesreiter, W. Fettes, I. Morgenstern, M. Schmid, and J. Maria Singer, "Search-space smoothing for combinatorial optimization problems," *Physica A*, vol. 243, no. 1, pp. 77–112, 1997.
[15] M. Ninio and J. J. Schneider, "Weight annealing," *Physica A*, vol. 349, no. 3, pp. 649–666, 2005.
[16] K.-H. Loh, B. Golden, and E. Wasil, "Solving the one-dimensional bin packing problem with a weight annealing heuristic," *Comput. Oper. Res.*, vol. 35, no. 7, pp. 2283–2291, 2008.
[17] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil, "See the forest before the trees: Fine-tuned learning and its application to the traveling salesman problem," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 28, no. 4, pp. 454–464, Jul. 1998.
[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[19] G. Elidan, M. Ninio, N. Friedman, and D. Schuurmans, "Data perturbation for escaping local maxima in learning," in *Proc. 18th Nat. Conf. Artif. Intell.*, 2002, pp. 132–139.
[20] R. Stallman, "The GNU manifesto," *Dr. Dobbs J. Softw. Tools*, vol. 10, no. 3, pp. 30–35, 1985.
[21] R. Qu, E. K. Burke, and B. McCollum, "Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems," *Eur. J. Oper. Res.*, vol. 198, no. 2, pp. 392–404, 2009.
[22] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, *et al.*, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013.
[23] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Proc. 3rd Prac. Theory Autom. Timetabling*, 2001, pp. 176–190.
[24] P. Cowling and K. Chakhlevitch, "Hyperheuristics for managing a large collection of low level heuristics to schedule personnel," in *Proc. IEEE Congr. Evol. Comput.*, Dec. 2003, pp. 1214–1221.
[25] D. Meignan, A. Koukam, and J.-C. Créput, "Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism," *J. Heurist.*, vol. 16, no. 6, pp. 859–879, 2010.
[26] G. Ochoa, J. Vázquez-Rodríguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: A hyper-heuristic landscape analysis," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 1873–1880.
[27] G. Ochoa, R. Qu, and E. K. Burke, "Analyzing the landscape of a graph based hyper-heuristic for timetabling problems," in *Proc. 11th Ann. Conf. Genetic Evol. Comput.*, 2009, pp. 341–348.
[28] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: A survey," *Genetic Program. Evol. Mach.*, vol. 11, no. 3, pp. 365–396, 2010.
[29] R. E. Keller and R. Poli, "Linear genetic programming of parsimonious metaheuristics," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 4508–4515.
[30] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart, "Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems," in *Proc. 4th Ann. Genetic Evol. Comput.*, Jul. 2002, pp. 942–948.
[31] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *Eur. J. Oper. Res.*, vol. 176, no. 1, pp. 177–192, 2007.

[32] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, "Classification of adaptive memetic algorithms: A comparative study," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 1, pp. 141–152, Feb. 2006.

[33] B. Codenotti, G. Manzini, L. Margara, and G. Resta, "Perturbation: An efficient technique for the solution of very large instances of the euclidean TSP," *INFORMS J. Comput.*, vol. 8, no. 2, pp. 125–133, 1996.

[34] I. Charon and O. Hudry, "Application of the noising method to the travelling salesman problem," *Eur. J. Oper. Res.*, vol. 125, no. 2, pp. 266–277, 2000.

[35] H. Jiang, X. Zhang, G. Chen, and M. Li, "Backbone analysis and algorithm design for the quadratic assignment problem," *Sci. China Ser. F*, vol. 51, no. 5, pp. 476–488, May 2008.

[36] U. Benlic and J. K. Hao, "A multilevel memetic approach for improving graph k-partitions," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 624–642, Oct. 2011.

[37] Z. Ren, H. Jiang, J. Xuan, and Z. Luo, "An accelerated-limit-crossing-based multilevel algorithm for the p-median problem," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 4, pp. 1187–1202, Aug. 2012.

[38] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone-based multilevel algorithm," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1195–1212, Sep.–Oct. 2012.

[39] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *Eur. J. Oper. Res.*, vol. 113, no. 2, pp. 469–499, 1999.

[40] M. Pelikan and D. Goldberg, "Hierarchical BOA solves ising spin glasses and MAXSAT," in *Proc. 5th Ann. Conf. Genetic Evol. Comput.*, 2003, p. 213.

[41] B. Monien and I. H. Sudborough, "Min cut is NP-complete for edge weighted trees," *Theor. Comput. Sci.*, vol. 58, no. 1, pp. 209–229, 1988.

[42] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems. II: The *p*-medians," *SIAM J. Appl. Math.*, vol. 37, no. 3, pp. 539–560, 1979.

[43] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Ann. Oper. Res.*, vol. 63, no. 5, pp. 511–623, 1996.

[44] J. E. Beasley, "A note on solving large *p*-median problems," *Eur. J. Oper. Res.*, vol. 21, no. 2, pp. 270–273, 1985.

[45] G. Reinelt, "TSPLIB: A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[46] M. Pelikan and A. Hartmann, "Searching for ground states of ising spin glasses with hierarchical BOA and cluster exact approximation," *Scalable Optimization Via Probabilistic Modeling*. Berlin/Heidelberg, Germany: Springer, 2006, pp. 333–349.

[47] M. G. C. Resende and R. F. Werneck, "On the implementation of a swap-based local search procedure for the *p*-median problem," in *Proc. 5th Workshop Algor. Eng. Exp.*, 2003, pp. 119–127.

[48] E. Correa, M. Steiner, A. Freitas, and C. Carnieri, "A genetic algorithm for the *p*-median problem," in *Proc. 3th Genetic Evol. Comput. Conf.*, 2001, pp. 1268–1275.

[49] M. G. C. Resende and R. F. Werneck, "A hybrid heuristic for the p-median problem," *J. Heurist.*, vol. 10, no. 1, pp. 59–88, 2004.

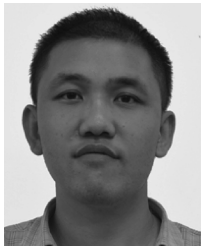
[50] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated f-race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*. Berlin/Heidelberg, Germany: Springer, 2010, pp. 311–336.

[51] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. San Mateo, CA, USA: Morgan Kaufmann, 2005.

[52] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo, "Frankenstein's PSO: A composite particle swarm optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1120–1132, Oct. 2009.

[53] D. Lim, Y. Jin, Y. S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 329–355, Jun. 2010.

[54] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.



Zhilei Ren received the B.Sc. degree in software engineering and the Ph.D. degree in computational mathematics from the Dalian University of Technology, Dalian, China, in 2007 and 2013, respectively.

He is currently a Post-Doctoral Researcher with the Dalian University of Technology. His current research interests include evolutionary computation and its applications in software engineering.

Dr. Ren is a member of the ACM and the CCF.

He Jiang (M'10) received the Ph.D. degree in computer science from the University of Science and Technology of China, Hefei, China.

He is currently a Professor with the Dalian University of Technology, Dalian, China. His current research interests include computational intelligence and its applications in software engineering and data mining.

Dr. Jiang is also a member of the ACM and the CCF.

Jifeng Xuan received the B.Sc. degree in software engineering and the Ph.D. degree in computational mathematics from the Dalian University of Technology, Dalian, China, in 2007 and 2013, respectively.

He is currently a Post-Doctoral Researcher with INRIA Lille–Nord Europe, Lille, France. His current research interests include software debugging, mining software repositories, and search based software engineering.

Dr. Xuan is a member of the ACM and the China Computer Federation.

Yan Hu received the B.Sc. and Ph.D. degrees in computer science from the University of Science and Technology of China, Hefei, China, in 2002 and 2007, respectively.

He is currently an Assistant Professor with the Dalian University of Technology, Dalian, China. His current research interests include model checking, program analysis, and software engineering.

Dr. Hu is a member of the ACM and the CCF.

Zhongxuan Luo received the B.Sc. and M.Sc. degrees in computational mathematics from Jilin University, Jilin, China, in 1985 and 1988, respectively, and the Ph.D. degree in computational mathematics the Dalian University of Technology, Dalian, China, in 1991.

He is currently a Professor with the School of Software, Dalian University of Technology. His current research interests include multivariate approximation theory and computational geometry.