2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Hyper-Heuristics with Penalty Parameter Adaptation for Constrained Optimization

Yu-Jun Zheng, Bei Zhang and Zhen Cheng
College of Computer Science & Technology
Zhejiang University of Technology
Hangzhou, Zhejiang, China 310023
Email: yujun.zheng@computer.org

***Abstract*—Penalty functions are widely used in constrained optimization, but determining optimal penalty parameters or weights turns out to be a difficult optimization problem itself. The paper proposes a hyper-heuristic approach, which searches the optimal penalty weight setting for low-level heuristics, taking the performance of those heuristics with specialized penalty weight settings as feedback to adjust the high-level search. The proposed approach can either be used for merely improving low-level heuristics, or be combined into a common hyper-heuristic framework for constrained optimization. Experiments on a set of well-known benchmark problems show that the hyper-heuristic approach with penalty parameter adaptation is effective in both aspects.**

I. Introduction

Taking inspiration from natural evolution processes, evolutionary algorithms (EAs) are a class of heuristic methods for solving complex optimization problems which typically have non-convex and highly nonlinear solution spaces, and which are otherwise computationally difficult to solve by conventional mathematical programming methods [1]. However, an aspect normally disregarded when using them for optimization (a rather common trend) is that these algorithms are typically unconstrained optimization procedures, and therefore it is necessary to find ways of incorporating the constraints (normally existing in any real-world application) into their fitness functions [2].

In this paper we consider the general constrained continuous optimization problem formulated as follows:

$$\textbf{min} \quad f(\mathbf{x}) \tag{1}$$

$$\textbf{s.t.} \quad \underline{x}_i \le x_i \le \overline{x}_i, \quad i = 1, ..., n \tag{2}$$

$$g_j(\mathbf{x}) \le 0, \quad j = 1, ..., m \tag{3}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the decision vector, Eq. (1) defines the objective function (which is usually nonlinear), Eq. (2) defines the lower and upper bounds of each dimension of the vector, and Eq. (3) defines j inequality constraints (equality constraints can be approximated by inequality constraints).

There have been different methods proposed for handling constraints in evolutionary optimization, including penalty functions, special representations and/or operators, repair procedures, separation of objectives and constraints, and hybrid methods [2]. Among them, penalty functions, due to their simplicity, are the oldest and the most widely used approach in constraint handling [3]. The common formulation of penalties transforms a constrained optimization problem into an unconstrained one with the following new objective function:

$$\textbf{min} \ f(\mathbf{x}) + w_0 \sum_{j=1}^{m} w_j (p_j(\mathbf{x}))^{\alpha} \tag{4}$$

where w_j are the penalty weights, $p_j(\mathbf{x}) = \max(0, g_j(\mathbf{x}))$ are the constraint violations ($0 \le j \le m$), and β is the exponent often set to 1 or 2. Typically, we denote $\phi(\mathbf{x}) = \sum_{j=1}^{m} w_j (p_j(\mathbf{x}))^{\alpha}$ as the penalty function.

Several researchers have studied heuristics on the design of penalty functions, and probably the most well-known of these studies is that conducted by Richardson et al. [4] from which the following guidelines were derived:

1) Penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints.
2) For a problem having few constraints and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find solutions.
3) Good penalty functions can be constructed from two quantities: the *maximum completion cost* and the *expected completion cost* (the completion cost is the cost of making feasible an infeasible solution).
4) Penalties should be close to the *expected completion cost*, but should not frequently fall below it. The more accurate the penalty, the better will be the solution found. When a penalty often underestimates the completion cost, then the search may fail to find a solution.

However, in practice, these guidelines are often difficult to follow [2], i.e., it is difficult to find the optimal penalty weights to balance the objective function and constraint violations for a given problem. As pointed out by Runarsson and Yao [5], if the weights are too small, an infeasible solution may not be penalized enough and hence may be evolved by EA; if they are too large, a feasible solution is very likely to be found, but could be of very poor quality. The critical issue here is how much exploration of infeasible regions should be

This work was supported by National Natural Science Foundation of China under Grant No. 61020106009, 61105073 and 61202204.

considered as reasonable, and the answer to this question is problem dependent.

Hyper-heuristics are approaches "using heuristics to choose or generate heuristics" [6]. That is, a hyper-heuristic operates on the search space of heuristics rather than directly on the search space of the underlying problem, using feedback of the low-level heuristics to adjust the "hyper" search process, and thus increasing not only the performance but also the level of generality on a variety of problems. In recent years, hyper-heuristics have attracted increasing attention and have been successfully applied to many real-world optimization problems [7]–[9].

However, until now, most research on hyper-heuristics focuses on the selection or generation of low-level *heuristics operating on the solutions* to the given problem, and few studies have looked at *heuristics operating on other components of EAs*. In this paper, we propose a novel hyper-heuristic approach that operates on low-level *heuristics on the design of penalty functions*. That is, our approach uses a metaheuristic EA to search the optimal penalty weight setting for low-level heuristics on constrained optimization, taking the performance of those heuristics with specialized penalty weight settings as feedback to adjust the high-level search. To the best of our knowledge, this is the first study that considers the penalty parameter adaptation in the context of hyper-heuristics. Computational experiments indicate that the proposed approach is very competitive on a set of well-known benchmark problems.

The rest of this paper is organized as follows: Section II describes some related work, Section III proposes our hyper-heuristic approach with penalty parameter adaptation for constrained optimization. Section IV presents the experimental results, and Section V concludes with discussion.

II. Related Work

In recent years, hyper-heuristics have been applied to a variety of problems including bin packing [10]–[12], job shop scheduling [13], [14] and project scheduling [15], [16], timetabling [17], [18], etc., most of which are combinatorial optimization problems.

Terashima-Marín et al. [19] presented a genetic algorithm (GA) based hyper-heuristic for the dynamic variable ordering in constraint satisfaction problems (CSP). The GA uses a variable-length representation and evolves combinations of condition-action rules (representing problem states) to go through a learning process (including training and testing) and produce efficient heuristics for the problem. For variable and value ordering in binary CSP, Bittle and Fox [20] used a symbolic cognitive architecture, augmented with constraint based reasoning as the hyper-heuristic machine learning framework. The approach seeks to minimize the number of low-level heuristics encoded yet dramatically expand the expressiveness of the hyper-heuristic by encoding the constituent measures of each heuristic, thereby providing more opportunities to achieve improved solutions.

Hyper-heuristics is used not only for producing a heuristic for controlling other heuristics, but also for generating new heuristics [9]. Ortiz-Bayliss et al. [21] studied the use of learning classifier systems (e.g., neural networks) to generate hyper-heuristics for variable ordering within CSP. During a training phase, the system constructs state-heuristic rules as it explores the search space, and heuristics with good performance at certain points are rewarded and become more likely to be applied in similar situations.

Up to now, the research on hyper-heuristics for constrained continuous optimization is relatively few. Villela Tinoco and Coello Coello [22] proposed a differential evolution (DE) [23] based hyper-heuristic for solving such problems. The approach adopts twelve DE models as low-level heuristics and four selection mechanisms for choosing the low-level heuristics. In the approach, constraints are handled by stochastic ranking [5] which can provide information to the mutation operator about the most appropriate direction of movement.

Most research on hyper-heuristics focuses on the manipulation of low-level heuristics for evolving the solutions to the given problem, neglecting other adaptable components of EAs such as initialization, parameterization, and population topologies. Recently, observing that the parameterization of the low-level heuristics poses great challenges to hyper-heuristics, Ren et al. [24] developed a hyper-heuristic framework with adaptive low-level parameters. In the framework, high-level search consists of two modules for managing the low-level heuristics and the low-level parameters respectively and simultaneously. A case study of the p-median problem shows that the approach is able to achieve competitive results with the state-of-the-art methods.

III. Hyper-heuristics with Low-level Penalty Parameter Adaptation

As mentioned above, in constrained optimization, deciding an optimal set of penalty weights can be a difficult optimization problem itself. Thus it is natural to incorporate the low-level penalty parameter adaptation into the context of hyper-heuristics.

The main idea of the proposed penalty-parameter-adaptation hyper-heuristic approach (denoted by PPA-HH) is very simple: it considers the setting of penalty weights as an $(m+1)$-dimensional continuous optimization problem, where m is the number of constraints, and each weight vector $\mathbf{w} = (w_0, w_1, ..., w_m)$ determines a penalty function in Eq. (4) of the underlying problem.

As illustrated in Fig. 1, suppose that hyper-heuristic evolves K penalty weight vectors, at each generation we can derive K versions of unconstrained optimization problems $P_1, P_2, ..., P_K$ for the underlying constrained problem, and use up to K low-level heuristics each for solving one unconstrained version. Typically, the best solution obtained by each low-level heuristic for the underlying problem is used for evaluating the fitness of the corresponding weight vector for the hyper-heuristic.

The PPA-HH can use any metaheuristic for continuous optimization to explore the $(m+1)$-dimensional high-level search space, taking the performance of low-level heuristics equipped with different weight vectors as feedback. Algorithm 1 presents the generic framework of PPA-HH.

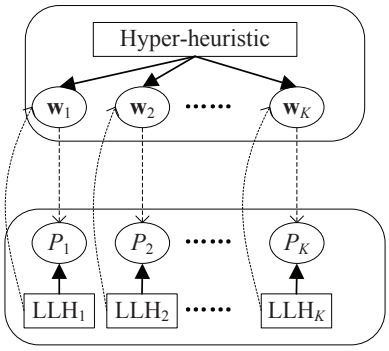

Fig. 1. The structure of the proposed PPA-HH, which evolves a set of penalty weight vectors used by low-level heuristics (LLH).

Without loss of generality, except for the outside penalty coefficient w_0, the search range of other m weight components $w_1, ..., w_m$ can all be bounded to [0,1]. According to empirical tests, giving a large enough range for w_0 also works; but here we employ a more effective strategy that transforms the search range by replacing w_0 with another variable w' such that:

$$w_0 = w' \times g^{\beta} \tag{5}$$

where g is the current generation number of the corresponding low-level heuristic and β is a constant larger than 1. Under this schema, we find that it is also appropriate to set the search range of w' to [0,1] in practice, and consequently the real search range of w_0 dynamically increases with generation, as suggested in [25].

Algorithm 1 The generic framework of PPA-HH.

```
1  Randomly initialize a population W of weight vectors;
2  initialize a low-level heuristic h;
3  while stop criterion is not satisfied do
4    for each w ∈ W do
5      Assign w to h;
6      Run h evolve one or more solutions to the problem;
7      Evaluate the achievement of h as the fitness of w;
8    Use the high-level heuristic to update the weight vectors;
9  return the best known solution.
```

Note that Algorithm 1 is a very general framework. The low-level heuristic h can be a population-based EA, or an algorithm operating on a single solution (such as tabu search). To appropriately evaluate the fitness of the weight vectors, it is important to make each run of the low-level heuristic consume the same computational resource, e.g., the same allowable generations. Typically, before and after each run of h with a specified $\mathbf{w}$, we respectively record the current best solutions of h, the objective values of which are denoted as f_h and f'_h. Then the achievement of this run of h can be evaluated as:

$$a_h = g^{\gamma}(f_h - f'_h) \tag{6}$$

where γ is a coefficient usually set to 1 or 0.5. The calculation of the achievement does not need any additional evaluation of the objective function of the underlying problem.

It should be noted that, f_h and f'_h should be evaluated based on the original objective function (1) rather than the penalized objective function (4). That is, if a run of h does not find a new better *feasible* solution, its achievement is considered as zero.

The PPA-HH can also be combined with any common hyper-heuristic (denoted by C-HH) that operates on a set of solution-oriented low-level heuristics. In such a hybrid hyper-heuristic framework, the C-HH operates in a similar way as most of the existing hyper-heuristics, while the PPA-HH optimizes the penalty weight setting for each low-level heuristic independently. Algorithm 2 presents the generic framework of a hybrid C-HH and PPA-HH approach.

Algorithm 2 The generic framework of C-HH combined with PPA-HH.

```
1  Prepare a set H of low-level heuristics to the problem;
2  Initialize a population Q of low-level heuristic sequences;
3  Initialize a population W of weight vectors;
4  while stop criterion is not satisfied do
5    for each q ∈ Q do
6      for each w ∈ W do
7        Assign w to q;
8        Run q to evolve one or more solutions to the problem;
9        Evaluate the achievement of q as the fitness of w;
10     Use the PPA-HH to update W;
11     Evaluate the performance of q;
12   Use the C-HH to update Q;
13 return the best known solution.
```

Note that Lines 6-11 of Algorithm 2 can be executed in parallel for each low-level heuristic sequence q, as long as the C-HH is parallelized. In this case, a set of PPA-HH instances simultaneously optimize the penalty weight settings for each $q \in Q$.

IV. Numerical Experiments

A. The Selection of PPA-HH

Generally speaking, any heuristic method for continuous optimization can be used for the considered high-level penalty parameter adaptation problem. However, to limit the additional cost incurred by the high-level search, the following guidelines are suggested for the selection of the hyper-heuristic for PPA-HH:

- If the hyper-heuristic is population based, it should perform well with a relatively small population size. Otherwise, a large number of inner loops (Lines 5-7 in Algorithm 1 or Lines 7-9 in Algorithm 2) can prolong the period of penalty weight update and thus decrease the algorithm efficiency.
- The exploration (global search) ability is more important than the exploitation (local search) ability of the hyper-heuristic, because the evaluation of the penalty weight fitness is itself stochastic and not very accurate.
- The hyper-heuristic itself should not involve too many control parameters and complex internal adjustment mechanisms.

According to the above considerations, here we employ the comprehensive learning particle swarm optimizer (CLPSO) [26] as the PPA-HH for evaluating our approach. CLPSO is one of the most prominent PSO methods, which replaces the velocity updating equation in basic PSO [27], [28] with the following one:

$$\mathbf{v}_i^j = w \cdot \mathbf{v}_i^j + c \cdot \mathit{rand}(0,1) \cdot (\mathbf{pbest}_{f(i)}^j - \mathbf{x}_i^j) \tag{7}$$

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are respectively the ith particle's position vector and velocity vector, w and c are two parameters named inertial weight and learning coefficient, and $\mathbf{pbest}_{f(i)}$ is the history best position of the $f(i)$-th particle, which is chosen for each dimension j of $\mathbf{x}_i$ by the following procedure [26]:

1) Generate a random number uniformly distributed in [0,1]. If the number is less than a predefined self-learning probability P_i, let $f(i)$ be 1 itself.
2) Otherwise, randomly choose two weight vectors from the population, compare the fitness values of their personal bests, and select the better one as $f(i)$.
3) If all the exemplars of $\mathbf{x}_i$ are its own, randomly choose one dimension to learn from the corresponding dimension of the personal best of another randomly selected weight vector.

And the self-learning probability P_i of $\mathbf{x}_i$ is calculated as follows (where *NP* denotes the population size):

$$P_i = 0.05 + 0.45\frac{\exp\ \frac{10k}{NP-1})}{\exp(10)-1} \tag{8}$$

In PPA-HH, the jth dimension of the position of ith weight vector is denoted by $\mathbf{w}_i^j$. Moreover, we simplify and improve the CLPSO by combining its velocity updating equation and position updating equation into the following one (inspired by the bare bones PSO [29]):

$$\mathbf{w}_i^j = N(\frac{\mathbf{pbest}_{f(i)}^j + \mathbf{w}_i^j)}{2}, |\mathbf{pbest}_{f(i)}^j - \mathbf{w}_i^j|)$$

where $N(\mu, \sigma^2)$ is a Gaussian random number with mean μ and standard deviation σ^2.

B. Benchmark Problems and Experimental Settings

To evaluate our PPA-HH approach, we select a set of well-known benchmark constrained optimization problems from [30] and [5], which cover various types of objective functions: linear, quadratic, cubic, polynomial and nonlinear, and the ratio between the size of the feasible search space and the size of the whole search space varies from 0% to nearly 99%. Table I presents a summary of the benchmark problems. For these problems, we consider a solution as a feasible one if $\sum_{j=1}^{m} \max(0, g_j(\mathbf{x}))\big) \leq \delta$, where δ is set to 10^{-8} in our experiments.

The experiments are conducted on a computer of Intel Core i5-2520M processor and 4GB DDR3 memory. For our CLPSO-based PAA-HH, we set $\alpha = 1$, $\beta = 2$ and $\gamma = 0.5$. The population size *NP* is set to 10 for problems g01 and g07, and 5 for the other problems. The maximum allowable number of generations for each run of the low-level heuristic is set to 5. As the stop criterion, the maximum number of function evaluations (nfe) is set to 300,000 for all the problems.

C. Evaluation of Single PPA-HH

We first evaluate the performance of the single PPA-HH approach on the test problems. The low-level heuristic is the basic DE algorithm [23]. At each generation, DE generates for each individual $\mathbf{x}_i$ in the population a mutant vector $\mathbf{v}_i$, and then generates a trial vector $\mathbf{u}_i$ is by mixing the components of the mutant vector and the original one, and finally selects the fitter one between $\mathbf{u}_i$ and $\mathbf{x}_i$ into the next generation. Here we equip PPA-HH with the simple DE/rand/1/bin method that uses the following mutation operator:

$$\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \tag{9}$$

where F is a scaling factor in the range [0,1], and r_1, r_2 and r_3 are three mutually exclusive random indexes in $[1, NP]$.

For comparison, we implement the modified constrained differential evolution algorithm (mCDE) [31], which uses the global competitive ranking method [32] for constraint handling. The method gives ranks to each individual $\mathbf{x}_i$ based on its objective function f_i and constraint violation ζ_i, separately, and calculates the fitness of $\mathbf{x}_i$ as:

$$\Phi_{GR}(\mathbf{x}_i) = P_f\frac{I_{i,f}-1}{NP-1} + (1-P_f)\frac{I_{i,\zeta}-1}{NP-1} \tag{10}$$

where $I_{i,f}$ and $I_{i,\zeta}$ are the ranks of point $\mathbf{x}_i$ based on f_i and ζ_i, respectively, and P_f is the probability that the fitness is calculated based on f_i. As suggested by [31], here we set $P_f = 0.45$.

mCDE also uses self-adaptive parameter mechanisms for the scaling factor and crossover rate, and a mixture of DE/rand/1/bin (9) and the following DE/best/1/bin mutation scheme:

$$\mathbf{v}_i = \mathbf{x}_{\text{best}} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \tag{11}$$

where $\mathbf{x}_{\text{best}}$ denotes the best solution vector in the population. Based on above strategies, mCDE has shown performance advantages over some state-of-the-art metaheuristics.

We run Both PAA-HH and mCDE 60 times on each problem. Table II presents the experimental results of the 13 problems. As we can see, although using one of the simplest form of DE as the low-level heuristic, our PPA-HH approach shows obvious performance advantage over the mCDE that employs more complex mutation operators and constraint-handling techniques. PPA-HH reaches the best known solutions on 7 problems, while mCDE only does so on 2 problems, i.e., g01 and g13. In fact, except on g01 and g13 the two methods always reach the optimum, both the median and mean values of PAA-HH are always better than that of mCDE on the other 11 problems.

TABLE I. THE SUMMARY OF THE BENCHMARK TEST PROBLEMS.

Problem	n	m	$f(\mathbf{x}^*)$
g01	13	9	-15.00000
g02	20	2	-0.803600
g03	10	1	-1.00000
g04	5	6	-30665.53900
g05	4	5	5126.49810
g06	2	2	-6961.81388
g07	9	8	24.30621
g08	2	2	-0.09583
g09	7	4	680.63006
g10	3	6	7049.33070
g11	2	1	0.75000
g12	3	1	-1.00000
g13	5	3	0.0539498

Moreover, we conduct nonparametric Wilcoxon rank sum test on the results of PPA-HH and mCED, and present the p-values in the last column of Table II. The test results show that, except on g01 and g13 there is no statistical difference between PPA-HH and mCED, on the remaining 11 problems PPA-HH always has statistical performance improvement over mCDE.

D. Evaluation of C-HH Combined with PPA-HH

Next, we evaluate the performance of the hybrid C-HH and PPA-HH approach. Here we choose *hyp*DE [22] as the C-HH, which adopts twelve different DE models as low-level heuristics, and uses the stochastic ranking method [5] for constraint handling. The method uses a bubble-sort-like procedure where *NP* individuals are ranked by comparing adjacent individuals in at least *NP* sweeps, and the probability of an adjacent individual winning a comparison is:

$$P_w = P_f P_{fw} + (1 - P_f) P_{\phi w} \tag{12}$$

where P_{fw} and $P_{\phi w}$ are probabilities of the individual winning according to the objective function and the penalty function, respectively, and P_f is the probability of using only the objective function for comparison ($P_f = 1$ if both the individuals are feasible).

To construct the hybrid hyper-heuristic method, we remove the original constraint handling mechanism from *hyp*DE, and then embed the CLPSO-based PPA-HH into *hyp*DE, as described in Algorithm 2. The low-level heuristics include DE/rand/1/bin, DE/best/1/bin, DE/rand/2/bin, DE/best/2/bin, DE/current-to-rand/1/bin, DE/current-to-best/1/bin, DE/rand/1/exp, DE/best/1/exp, DE/rand/2/exp, DE/best/2/exp, DE/current-to-rand/1/exp, and DE/current-to-best/1/exp [23].

We run the hybrid C-HH and PPA-HH method 60 times on the 13 test problems. For the original *hyp*DE method, the results are directly taken from [22]. Table II compares the results of the hypDE with and without PPA-HH.

As we can see from the results, the hybrid C-HH and PPA-HH reaches the best known solutions on 9 problems, while the single C-HH only does so on 6 problems, i.e., g01, g04, g06, g08, g09, and g12. As shown by the Wilcoxon rank sum test results, except on these 6 problems there is no statistical difference between the two methods, on the remaining 7 problems the hybrid C-HH and PPA-HH always has significant performance improvement over the single C-HH. Moreover, in terms of either median or mean values, the single C-HH never obtains a better result than the hybrid approach on any test problem.

In summary, from the two series of experiments, we find that our PPA-HH approach can effectively improve not only the performance of a single metaheuristic optimization algorithm, but also the performance of other common hyper-heuristic methods. This demonstrates that the proposed penalty parameter adaptation hyper-heuristic is a very useful approach for constrained optimization problems.

V. CONCLUSIONS

Hyper-heuristics of solution-oriented low-level heuristics have been widely investigated in the literature. However, research on hyper-heuristics for optimizing other aspects of low-level heuristics is very few. In this paper, we propose a hyper-heuristic approach for penalty parameter adaptation of low-level heuristics for constrained optimization. Our approach uses a high-level heuristic to search the optimal penalty weight setting, taking the performance of low-level heuristics with different penalty weights as feedback. Computational experiments demonstrate that the proposed approach is very effective in improving either a metaheuristic or a hyper-heuristic for constrained optimization.

In this paper, our PPA-HH is only implemented with the CLPSO, and used for penalty parameter adaptation of low-level DE-based heuristics. We are currently testing more other state-of-the-art metaheuristics as the high-level search methods, as well as integrating the PPA-HH into other common hyper-heuristic frameworks. Our ongoing work also including validating the solution accuracy of PPA-HH by comparing with offline methods such as automatic algorithm configuration methods [33].

This study focuses on the hyper-heuristic for penalty parameter adaptation. Except penalty functions, there are also other efficient methods for constraint handling, such as special representations and operators, repair procedures, stochastic ranking, multiobjectivization, etc. Thus, our future work also include developing a higher-level hyper-heuristic approach which can adaptively choose and/or combine these different constraint handling methods.

REFERENCES

[1] K. A. De Jong, *Evolutionary computation: a unified approach*. MIT press Cambridge, 2006.

[2] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.

[3] T. Runarsson and X. Yao, "Search biases in constrained evolutionary optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 35, no. 2, pp. 233–243, 2005.

[4] J. T. Richardson, M. R. Palmer, G. E. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in *Proceedings of the Third International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 191–197.

[5] T. Runarsson and X. Yao, "Stochastic ranking for constrained evolutionary optimization," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.

[6] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern research technolology," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer, 2003, ch. 16, pp. 457–474.

[7] S. A. Bittle and M. S. Fox, "Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, ser. GECCO '09. New York, NY, USA: ACM, 2009, pp. 2209–2212.

[8] H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Renón, "Generalized hyper-heuristics for solving 2D regular and irregular packing problems," *Annals of Operations Research*, vol. 179, no. 1, pp. 369–392, 2010.

TABLE II. THE EXPERIMENTAL RESULTS OF THE SINGLE PPA-HH AND THE MCDE.

	mCDE				PPA-HH				
#	best	median	mean	std	best	median	mean	std	p-value
g01	-15.00000	**-15.00000**	-15.00000	0.00E+00	-15.00000	**-15.00000**	-15.00000	0.00E+00	1
g02	-0.80360	-0.80103	-0.80083	3.10E-03	-0.80360	-0.80168	-0.80151	9.17E-04	1.92E-15†
g03	-1.0000	-0.99997	-0.99996	5.24E-05	-0.99999999	-0.99999997	-0.99999995	3.93E-08	2.09E-24†
g04	-30665.538720	-30665.53852	-30665.53869	5.02E-05	-30665.53900	**-30665.53900**	-30665.53900	0.00E+00	6.05E-20†
g05	5126.49826	5126.49854	5126.49863	7.29E-04	5126.49810	5126.49818	5126.49819	6.98E-05	5.05E-04†
g06	-6961.80973	6953.15337	-6950.25752	5.06E+01	-6961.81388	**-6961.81388**	-6961.81388	0.00E+00	1.25E-18†
g07	24.30656	24.30692	24.30687	5.81E-04	24.30628	24.30631	24.30630	3.10E-05	1.39E-09†
g08	-0.09582	-0.09582	-0.09582	1.89E-06	-0.09583	**-0.09583**	-0.09583	0.00E+00	2.75E-16†
g09	680.63008	680.63016	680.63021	4.80E-05	680.63006	**680.63006**	680.63006	0.00E+00	1.08E-18†
g10	7052.68216	7056.31318	7055.92677	8.37E+00	7049.35056	7050.00206	7050.05808	8.59E-01	3.09E-08†
g11	0.75000	0.75106	0.75093	1.60E-03	0.75000	0.75048	0.75030	7.07E-04	2.62E-06†
g12	-1.00000	-0.9999995	-0.9999993	6.96E-07	-1.00000	**-1.00000**	-1.00000	0.00E+00	5.25E-08†
g13	0.05395	**0.05395**	0.05395	0.00E+00	0.05395	**0.05395**	0.05395	0.00E+00	1

[1] 'best' denotes the best of the objective function values obtained among all runs, 'median' and 'mean' respectively denote the a median and average of the best objective function value over the 60 runs, 'std' denotes the standard deviation, and 'p-value' denotes the result of paired t-test between the two method. The value in boldface indicates that the algorithm reaches the best known solution of the problem. In the last column, † indicates that the PPA-HH has statistically significant improvement over the mCDE (at 95% confidence level).

TABLE III. THE EXPERIMENTAL RESULTS OF THE hypDE WITH AND WITHOUT PPA-HH.

	hypDE				hypDE with PPA-HH				
Problem	best	median	mean	std	best	median	mean	std	p-value
g01	-15.00000	**-15.00000**	-15.00000	0.00E+00	-15.00000	**-15.00000**	-15.00000	0.00E+00	1
g02	-0.80148	-0.80146	-0.80144	4.67E-03	-0.80360	-0.80338	-0.80303	5.22E-05	8.09E-12†
g03	-0.9999999	-0.9999997	-0.9999997	7.05E-07	-0.99999999	-0.99999999	-0.99999998	1.02E-08	1.03E-20†
g04	-30665.53900	**30665.53900**	-30665.53900	0.00E+00	-30665.53900	**-30665.53900**	-30665.53900	0.00E+00	1
g05	5126.49810	5151.32067	5171.09695	1.42E+02	5126.49810	**5126.49810**	5126.49810	0.00E+00	3.08E-03†
g06	-6961.81400	**-6961.81400**	-6961.81400	0.00E+00	-6961.81400	**-6961.81400**	-6961.81400	0.00E+00	1
g07	24.30627	24.30630	24.30649	7.54E-04	24.30622	24.30623	24.30626	3.10E-05	7.15E-05†
g08	-0.09583	**-0.09583**	-0.09583	0.00E+00	-0.09583	**-0.09583**	-0.09583	0.00E+00	1
g09	680.63006	**680.63006**	680.63006	0.00E+00	680.63006	**680.63006**	680.63006	0.00E+00	1
g10	7049.55589	7085.07333	7103.02420	8.82E+01	7049.33880	7049.35006	7049.36901	1.25E-02	1.33E-03†
g11	0.75000	0.82663	0.89979	1.18E-01	0.75000	**0.75000**	0.75000	0.00E+00	2.82E-12†
g12	-1.00000	-0.9999995	-0.9999993	6.96E-07	-1.00000	**-1.00000**	-1.00000	0.00E+00	1
g13	0.31329	0.36922	0.43886	2.13E-01	0.05395	**0.05395**	0.05395	0.00E+00	5.82E-22†

[2] 'best' denotes the best of the objective function values obtained among all runs, 'median' and 'mean' respectively denote the a median and average of the best objective function value over the 60 runs, 'std' denotes the standard deviation, and 'p-value' denotes the result of paired t-test between the two method. The value in boldface indicates that the algorithm reaches the best known solution of the problem. In the last column, † indicates that the hypDE with PPA-HH has statistically significant improvement over the hypDE without PPA-HH (at 95% confidence level).

[9] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 2, pp. 1695–1724, 2013.

[10] A. Cuesta-Cañada, L. Garrido, and H. Terashima-Marín, "Building hyper-heuristics through ant colony optimization for the 2d bin packing problem," in *Knowledge-Based Intelligent Information and Engineering Systems*, ser. Lecture Notes in Computer Science, R. Khosla, R. Howlett, and L. Jain, Eds. Springer Berlin Heidelberg, 2005, vol. 3684, pp. 654–660.

[11] R. Poli, J. Woodward, and E. Burke, "A histogram-matching approach to the evolution of bin-packing strategies," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 3500–3507.

[12] E. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[13] N. B. Ho and J. C. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *IEEE Congress on Evolutionary Computation*, vol. 3, 2005, pp. 2848–2855.

[14] J. A. Vázquez-Rodríguez and S. Petrovic, "A new dispatching rule based genetic algorithm for the multi-objective job shop problem," *Journal of Heuristics*, vol. 16, no. 6, pp. 771–793, 2010.

[15] K. Anagnostopoulos and G. Koulinas, "A genetic hyperheuristic algorithm for the resource constrained project scheduling problem," in *2010 IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–6.

[16] G. Koulinas, L. Kotsikas, and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, 2014, online first.

[17] N. Pillay and W. Banzhaf, "A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling

problem," *European Journal of Operational Research*, vol. 197, no. 2, pp. 482–491, 2009.

[18] R. Qu and E. K. Burke, "Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems," *Journal of the Operational Research Society*, vol. 60, no. 9, pp. 1273–1285, 2009.

[19] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón, "Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 571–578.

[20] S. A. Bittle and M. S. Fox, "Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, ser. GECCO '09. New York, NY, USA: ACM, 2009, pp. 2209–2212.

[21] J. Ortiz-Bayliss, H. Terashima-Marin, and S. Conant-Pablos, "Using learning classifier systems to design selective hyper-heuristics for constraint satisfaction problems," in *2013 IEEE Congress on Evolutionary Computation*, June 2013, pp. 2618–2625.

[22] J. C. Villela Tinoco and C. A. Coello Coello, "*hyp*DE: A hyper-heuristic based on differential evolution for solving constrained optimization problems," in *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, ser. Advances in Intelligent Systems and Computing, O. Schütze, C. A. Coello Coello, A.-A. Tantar, E. Tantar, P. Bouvry, P. Del Moral, and P. Legrand, Eds. Springer Berlin Heidelberg, 2013, vol. 175, pp. 267–282.

[23] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[24] Z. Ren, H. Jiang, J. Xuan, and Z. Luo, "Hyper-heuristics with low level parameter adaptation," *Evolutionary Computation*, vol. 20, no. 2, pp. 189–227, 2012.

[25] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," in *Proceedings of the 4th Annual Conference on Evolutionary Programming*, J. McDonnell, R. Reynolds, and D. Fogel, Eds., 1995, pp. 135–155.

[26] J. J. Liang, A. K. Qin, P. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, 2006.

[27] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[28] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *IEEE Congress on Evolutionary Computation*, 1998, pp. 69–73.

[29] J. Kennedy, "Bare bones particle swarms," in *IEEE Swarm Intelligence Symposium*, 2003, pp. 80–87.

[30] S. Koziel and Z. Michalewicz, "Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization," *Evolutionary Computation*, vol. 7, no. 1, pp. 19–44, Mar. 1999.

[31] M. Azad and M. Fernandes, "Modified constrained differential evolution for solving nonlinear global optimization problems," in *Computational Intelligence*, ser. Studies in Computational Intelligence, K. Madani, A. Dourado, A. Rosa, and J. Filipe, Eds. Springer Berlin Heidelberg, 2013, vol. 465, pp. 85–100.

[32] T. Runarsson and X. Yao, "Constrained evolutionary optimization - the penalty function approach," in *Evolutionary Optimization*, ser. International Series in Operations Research and Management Science, e. a. Sarker, Ed. New York: Springer, 2003, vol. 48, pp. 87–113.

[33] F. Hutter and H. H. Hoos, "Automatic algorithm configuration based on local search," in *Proceedings of the 22nd National Conference on Artificial Intelligence*, 2007, p. 1152C1157.