A Dynamic Multiarmed Bandit-Gene Expression Programming Hyper-Heuristic for Combinatorial Optimization Problems

Nasser R. Sabar, Masri Ayob, Graham Kendall, *Senior Member, IEEE*, and Rong Qu, *Senior Member, IEEE*

***Abstract*—Hyper-heuristics are search methodologies that aim to provide high-quality solutions across a wide variety of problem domains, rather than developing tailor-made methodologies for each problem instance/domain. A traditional hyper-heuristic framework has two levels, namely, the high level strategy (heuristic selection mechanism and the acceptance criterion) and low level heuristics (a set of problem specific heuristics). Due to the different landscape structures of different problem instances, the high level strategy plays an important role in the design of a hyper-heuristic framework. In this paper, we propose a new high level strategy for a hyper-heuristic framework. The proposed high-level strategy utilizes a dynamic multiarmed bandit-extreme value-based reward as an online heuristic selection mechanism to select the appropriate heuristic to be applied at each iteration. In addition, we propose a gene expression programming framework to automatically generate the acceptance criterion for each problem instance, instead of using human-designed criteria. Two well-known, and very different, combinatorial optimization problems, one static (exam timetabling) and one dynamic (dynamic vehicle routing) are used to demonstrate the generality of the proposed framework. Compared with state-of-the-art hyper-heuristics and other bespoke methods, empirical results demonstrate that the proposed framework is able to generalize well across both domains. We obtain competitive, if not better results, when compared to the best known results obtained from other methods that have been presented in the scientific literature. We also compare our approach against the recently released hyper-heuristic competition test suite. We again demonstrate the generality of our approach when we compare against other methods that have utilized the same six benchmark datasets from this test suite.**

***Index Terms*—CHeSC, dynamic optimization, gene expression programming, hyper-heuristic, timetabling, vehicle routing.**

Manuscript received January 31, 2013; revised July 30, 2013, March 8, 2014, and April 22, 2014; accepted May 11, 2014. Date of publication June 2, 2014; date of current version January 13, 2015. This paper was recommended by Associate Editor Q. Zhang.

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. This includes a PDF file with additional tables. This material is 0.060 MB in size.

N. R. Sabar is with Data Mining and Optimization Research Group, Centre for Artificial Intelligent, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia, and also with University of Nottingham Malaysia Campus, Semenyih 43500, Malaysia (e-mail: nasser.sabar@nottingham.edu.my).

M. Ayob is with Data Mining and Optimization Research Group, Centre for Artificial Intelligent, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia (e-mail: masri@ftsm.ukm.my).

G. Kendall is with ASAP Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K., and also with University of Nottingham Malaysia Campus, Semenyih 43500, Malaysia (e-mail: graham.kendall@nottingham.ac.uk).

R. Qu is with ASAP Research Group, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, U.K. (e-mail: rong.qu@nottingham.ac.uk).

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.2014.2323936

I. Introduction

META-HEURISTIC research communities have acknowledged the fact that meta-heuristic configurations (operators and parameter settings) play a crucial role in algorithmic performance [1]–[3]. Indeed, it has been shown that different meta-heuristic configurations work well for particular problem instances or only at particular stages of the solving process [4], [5]. Within this context, automated heuristic design methods have emerged as a new research trend [4]. The ultimate goal of these methods is to automate the algorithm design process as far as possible, to enable them to work effectively across a diverse set of problem domains [1], [6]. Hyper-heuristics [4] represent one of these methodologies. They are a search methodology that is able to provide solutions across a wide variety of problem domains, rather than being tailored for each problem or even each problem instance. Hyper-heuristics operate on the heuristic search space, rather than operating directly on the solution space, which is usually the case with meta-heuristic algorithms [7]. The key motivation behind hyper-heuristics is to raise the level of generality, by drawing on the strengths, and recognizing the weaknesses, of different heuristics. The most common hyper-heuristic framework has two levels; a high level strategy and a set of low level heuristics (LLHs). The high level strategy manages which LLH to call (heuristic selection mechanism) and then decides whether to accept the returned solution (the acceptance criterion). The LLHs contain a set of problem specific heuristics which are different for each problem domain.

The success of a hyper-heuristic framework is usually due to the design of the high level strategy and it is not surprising that much research has been focused in this area [8]. The variety of landscape structures and the difficulty of the problem domains, or even problem instances, usually require an efficient heuristic selection mechanism and acceptance criteria to achieve good performance [4]. Both components are crucial and many works have shown that different combinations and configurations usually yield different performance [7]–[9].

Therefore, in this paper, we address these challenges by proposing a new high level strategy for the hyper-heuristic framework with the following two components (see Fig. 1).

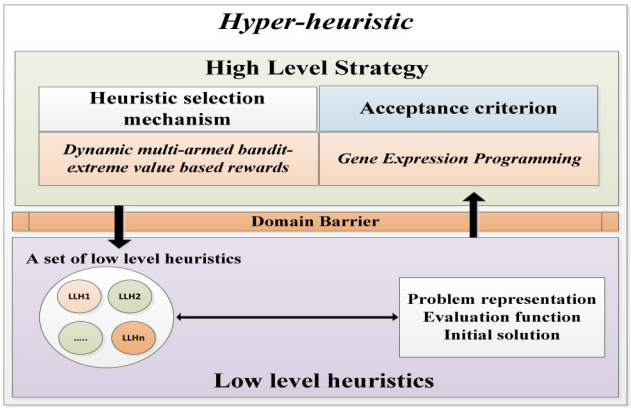

Fig. 1. Proposed gene expression programming-based hyper-heuristic (GEP-HH) framework.

A. Heuristic Selection Mechanism

The proposed framework utilizes the dynamic multiarmed bandit (DMAB)-extreme value-based reward [10] as an online heuristic selection mechanism. The attractive feature of a DMAB is the integration of the Page–Hinkly (PH) statistical test to determine the most appropriate heuristic for the next iteration by detecting the changes in average reward for the current best heuristics. In addition, the extreme value-based reward credit assignment mechanism records the historical information of each heuristic to be used during the heuristic selection process.

B. Acceptance Criterion

Instead of using an acceptance criterion manually designed by human experts (which usually requires ongoing tuning), we propose an automatic framework to automatically generate, during the solving process, different acceptance criteria for different instances or problem domains as well as to consider the current problem state by using gene expression programming (GEP) [11]. We choose GEP to automatically generate the acceptance criteria due to its ability to always generate solutions that are syntactically correct and to avoid the problem of code bloat.

In the scientific literature, automatic program generation methods, such as genetic programming (GP), have been successfully utilized as hyper-heuristics to generate heuristics for optimization problems such as 2-D packing and MAX-SAT problems [7]. However, despite the success of GP-based hyper-heuristics, the same hyper-heuristic cannot be used to generate heuristics for other domains such as exam timetabling or vehicle routing problems. This is because existing GP-based hyper-heuristics generate heuristics for only one domain. Hence, the function and terminal sets that have been defined for one domain cannot usually be used for other domains. In other words, to solve a new problem domain we have to define a different set of functions and terminals that are suitable to the problem at hand. In this paper, we propose an automatic program generation framework to automatically generate the high level strategy competent of the hyper-heuristic framework. The novelty of the proposed framework is that it is being used at the higher level of abstraction and can tackle many optimization problems using the same set of functions and terminals. This feature distinguishes our framework from existing GP-based hyper-heuristics. In practice, evolving or optimizing algorithm components will not only alleviate user intervention in finding the most effective configuration, but can also facilitate algorithm configurations. In addition, manual configurations usually only represent a small fraction of the available search space and they often require a considerable amount of expertise and experience. Hence, exploring the search space using a suitable search methodology (i.e., GEP in this paper) might yield a better performance compared to a manually configured search methodology.

Our objectives are as follows.

1) To propose an online hyper-heuristic framework that can adapt itself to the current problem state using a heuristic selection mechanism which integrates a statistical test to select the most appropriate LLH.
2) To propose an online framework to automatically generate, for each instance, an acceptance criterion that uses the current problem state, and that is able to cope with changes that might occur during the search process. This will be achieved using a GEP algorithm.
3) To test the generality and consistency of the proposed hyper-heuristic framework on two different problem domains (both static and dynamic) and compare its performance against the state-of-the-art of hyper-heuristics and the best known bespoke methods in the scientific literature.

Two well-known combinatorial optimization problems, but with very different search space characteristics, are used as our benchmarks. The problems are: the exam timetabling problem (ITC 2007 instances [12]) and the dynamic vehicle routing problem (DVRP) (Kilby instances [13]). We also further test the generality of the approach by comparing against the recently introduced hyper-heuristic test suite of problems [hyper-heuristic competition (CHeSC 2011)] [14]. To the best of our knowledge, this is the first work in the hyper-heuristic literature which has considered both dynamic and static problems.

II. Hyper-Heuristics and Related Works

Burke *et al*. [4] defined a hyper-heuristic as "an automated methodology for selecting or generating heuristics to solve hard computational search problems". Hyper-heuristics have been widely used, with much success, to solve various classes of problems. A traditional hyper-heuristic framework has two levels, a high and a low level.

The high level strategy, which are problem-independent and have no knowledge of the domain, control the selection or generation of heuristics (without knowing what specific function it performs) at each decision point in the search process. In contrast to meta-heuristics, hyper-heuristics search the space of heuristics instead of directly searching the solution space. The LLHs represent a set of problem-dependent heuristics which operate directly on the solution space.

Generally, the high level abstraction of the hyper-heuristic framework means that it can be applied to multiple problem domains with little (or no) additional development effort. Recently, Burke *et al*. [4] classified hyper-heuristic frameworks

based on the nature of the heuristic search space and the source of feedback during learning. The source of feedback can either be online, if the hyper-heuristic framework uses feedback obtained during the problem-solving procedure, or offline, if the hyper-heuristic framework uses information gathered during a training phase in order to be used when solving future unseen instances. The nature of the heuristic search space is also classified into two subclasses known as heuristics to choose heuristics and heuristics to generate heuristics. The classification specifies whether heuristics (either chosen or generated) are constructive or perturbative. Our proposed hyper-heuristic framework can be classified as an online perturbative heuristic to choose heuristics.

A. Heuristic to Choose Heuristics

Most hyper-heuristic frameworks that have been published to date have been heuristics to choose heuristics [4]. For a given problem instance, the role of the hyper-heuristic framework is to intelligently choose a LLH from the provided set, and apply it. The idea is to combine the strengths of several heuristics into one framework. Meta-heuristics and machine-learning methodologies have been used as heuristic selection mechanisms [15], for example tabu search (TS) with reinforcement learning [16], and scatter search [17]. Many different acceptance criteria have also been used, including all moves [18], only improving [18], improving and equal [18], Monte Carlo [19], record-to-record travel [20], simulated annealing [21], [22], late acceptance [23], great deluge [24], and TS [25]. More details are available in [7].

Recently, the cross-domain heuristic search (CHeSC) competition was introduced, which provides a common software interface for investigating different (high level) hyper-heuristics and provides access to six problem domains where the LLHs are provided as part of the supplied framework [14]. The algorithm designer only needs to provide the high level components (heuristic selection and acceptance criterion). Further details about the competition, including results, are available in [14].

B. Heuristics to Generate Heuristics

Heuristics that generate heuristics focus on designing new heuristics by combining existing heuristic components and then applying them to the current solution. Generative GP hyper-heuristics have been utilized to solve many combinatorial optimization problems including SAT [26], [27], timetabling [27], [28], vehicle routing [28], job shop scheduling [29], and bin packing [30]. A recent review on hyper-heuristic is available in [7] which provides more details about this area.

Although promising results have been achieved, GP has been utilized as an offline heuristic/rule builder using a specific set of functions and terminals. Besides being computationally expensive due to the need for training and testing, they do not guarantee to deliver the same performance across different domains or even different instances of the same domain. This is because the generated heuristics/rules are suited to only the instance(s) that have been used in the training phase. Furthermore, they are tailored to solve specific problems and were only applied to a single (static) domain, which raises the question: to what extent will they generalize to other domains?

The success of the above work, which has some resemblance to the proposed GEP, is the main motivation for proposing an online GEP framework to generate the acceptance criteria for the hyper-heuristic framework. The benefit of this framework is the ability to generate different acceptance criteria for different instances based on the problem state and thus enabling it to cope with changes that might occur during the solving process.

III. Proposed Framework

We start by describing the proposed perturbative-based hyper-heuristic framework, followed by the components of the high level strategy, i.e., the heuristic selection and acceptance criterion mechanisms. Finally, we describe the LLHs that will be used in our framework.

A. Perturbative-Based Hyper-Heuristic Framework

Our online perturbative-based hyper-heuristic framework comprises a high level strategy and a set of LLHs. The high level strategy consists of two components, heuristic selection and acceptance criterion. The goal of the high level strategy is to select a LLH to be applied at a given decision point. The low level contains a set of perturbative heuristics that are applied to the problem instance, when called by the high level strategy. Therefore, based on the utilized LLHs, the proposed hyper-heuristic framework is an improvement-based method as the hyper-heuristic starts with an initial solution and iteratively improves it using a set of perturbative LLHs.

The proposed hyper-heuristic iteratively explores the neighborhood of the incumbent solution, seeking for improvement. Given a set of LLHs, a complete initial solution, S, (generated either randomly or via a constructive heuristic) and the objective function, F, the aim of the hyper-heuristic framework is to select an appropriate LLH and apply it to the current solution ($S' = LLH(S)$). Next, the objective function is called to evaluate the quality of the resultant solution ($F(S')$), followed by the acceptance criterion which decides whether to accept or reject S'. If S' is accepted, it will replace S. Otherwise, S' will be rejected. The hyper-heuristic will update the algorithmic parameters and start another iteration. This process is repeated for a given number of iterations. Next, we discuss the components of the proposed high level strategy.

B. High-Level Strategy

We propose a new high level strategy, as shown in Fig. 2. It has two components: a heuristic selection mechanism (DMAB-extreme value-based reward) and an acceptance criterion (GEP).

1) Heuristic Selection Mechanism: The heuristic selection mechanism successively invokes two tasks, credit assignment (extreme value-based reward [10]) and heuristic selector (DMAB mechanism [10]). The credit assignment mechanism maintains a value (reward) for each LLH and these values are used by the heuristic selector mechanism to decide which heuristic will be executed at each decision point.

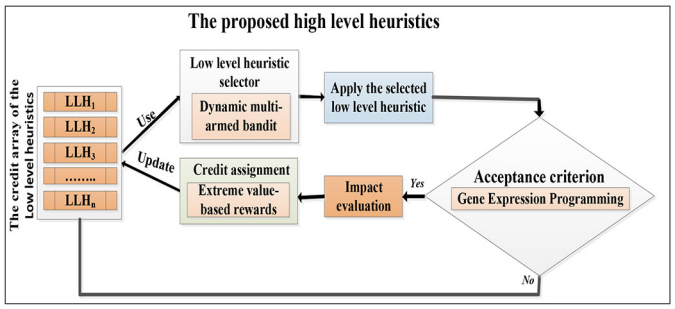

Fig. 2. High level strategy within the proposed GEP-HH framework.

a) Credit assignment mechanism: Extreme value-based reward: The credit assignment mechanism maintains a value (reward) for each LLH, indicating how well it has performed recently. The LLH will be rewarded during the search process if it finds a better solution or the solution is accepted by the acceptance criterion. In this paper, the extreme value-based reward [10] is employed as the credit assignment mechanism. LLHs, which are called infrequently, but lead to large improvements in solution quality, are preferred over those that only lead to small improvements, over a longer time frame. LLHs which bring frequent, small improvements will be rewarded less and consequently have less chance of being selected [10].

The extreme value-based reward mechanism works as follows. When a LLH is selected, its improvement, compared to the current solution, is computed. The improvement (PI_{LLH}) of the applied LLH to a current solution is calculated as follows: assume f_1 is the quality of the current solution and f_2 is the quality of the resultant solution after applying the LLH, $PI_{\text{LLH}} = (f_1 - f_2 / f_1) \times 100$. PI_{LLH} is then saved for that LLH, at the corresponding iteration for a sliding time window of size W, using first in, first out (FIFO), i.e., improvements of the selected LLH in the last W iterations are saved. The credit of any LLH (r) is then set as the maximum value in its corresponding sliding window as follows:

$$r = \max_{i=1\ldots W} \{(PI_{\text{LLH}}, i)\} \tag{1}$$

where W is the size of the sliding window. The size of the sliding window, W, controls the size of the gathered information. Based on our preliminary testing (see Section IV-A), it was found that when $W = 20$ the hyper-heuristic performs well. We have thus fixed $W = 20$ in all our experiments.

b) Heuristic selector: Dynamic multiarmed bandit mechanism: Based on their previous performance (assigned credit value by the credit assignment mechanism) and the number of times that a LLH has been applied, the heuristic selection mechanism selects which LLH to be applied next. We use the DMAB mechanism as an online heuristic selector mechanism [10]. The attractive feature of DMAB is the integration of the PH statistical test to detect the changes in average reward of the current best LLH. DMAB is based on an upper confidence bound strategy which deterministically selects a LLH with maximum accumulative reward. A LLH is selected based on its empirical reward and the number of times it has been applied up to the current time step.

Formally, let k denote the number of available LLHs. DMAB selects the best LLH that maximizes the accumulative reward over time. Each LLH is associated with its empirical reward $q_{i,t}$ (2) (the average reward r_i obtained by the i-th LLH up to time t) and a confidence level $n_{i,t}$ (the number of times that the i-th LLH has been applied up to time t)

$$q_{i,t+1} = \left(\frac{(n_{i,t-1}) * q_{i,t} + r_{i,t}}{n_{i,t}} \right) \tag{2}$$

where $r_{i,t}$ is the credit value (based on the credit assignment mechanism) of the i-th LLH up to time t. At each decision point, the LLH with the best confidence interval (maximum accumulative reward) is selected [using (3)] to be applied to the current solution

$$\text{select} \max_{i=1\ldots k} \left(q_{i,t} + c \sqrt{\frac{2 \log \sum_{j=1}^{k} n_{j,t}}{n_{i,t}}} \right) \tag{3}$$

where c is a scaling factor which controls the trade-off between the LLH that has the best reward [the left term of (3)] and the heuristic that has been infrequently applied [the right term of (3)].

DMAB uses the PH statistical test to detect the changes in average reward of the current best LLHs. If the current best LLH is no longer the best one, DMAB is restarted from scratch. The underlying idea is to quickly, and dynamically, identify a new best LLH, avoiding irrelevant information. Let α_t represent the average reward of a LLH over the last t time steps, r represents the reward of the i-th LLH at time j, the PH test [10] uses (4) to detect the changes in average reward

$$\alpha_t = \frac{1}{t} \sum_{j=1}^{t} r_j, \quad e_t = (r_t - \alpha_t + \delta), \quad m_t = \sum_{j=1}^{t} e_j \tag{4}$$

where e_t is the difference between the current reward (r_t) and the average reward (α_t) of the i-th LLH plus a tolerance parameter δ which is fixed to 0.15 (see Section IV-A). m_t is a variable that accumulates the differences e_t up to step t. PH recognizes a change in reward when the difference $(\max_{j=1\ldots t}\{|m_i|\} - |m_t|)$ is greater than a predefined threshold γ ($\max_{j=1\ldots t}\{|m_j|\} - |m_t| > \gamma$).

2) Acceptance Criterion Mechanism: The role of the acceptance criterion is to decide whether to accept or reject the solution that is generated once the selected LLH has been applied [8]. In this paper, we propose a GEP framework to evolve the acceptance criterion for our hyper-heuristic. It is implemented as an online acceptance criterion generation method that takes the quality of the previous solution, current solution, and the current state of the search process as input and returns the decision as to whether to accept or reject the solution. This removes the need for manual customization and/or tuning. It also contributes to the literature on the automatic generation of heuristic components that are able to avoid being trapped in confined areas of the search space and are able to work well across different problem domains/instances.

In the following subsections, we present the basic GEP algorithm followed by the proposed framework to generate the acceptance criteria of the proposed high level strategy of the hyper-heuristic.

a) Basic gene expression programming algorithm: GEP [11] is a form of GP that uses the advantage of both genetic algorithms (GA) and GP in evolving a population of computer programs. The attractive features of GEP, compared to GP, are its ability to always generate syntactically correct programs and avoid the problem of code bloat (a recognized problem in traditional GP) [11]. GEP uses a linear representation of a fixed size of strings called genomes (or chromosomes) which are translated into parse trees of various sizes in a breadth-first search form. The parse tree is then executed to generate a program that will be used to solve the given problem. Instead of applying operators directly to the trees, as in GP, GEP applies genetic operators (crossover, mutation, inversion, and transposition) directly to the linear encoding.

The genomes in GEP represent a set of symbol strings called genes. Each one consists of two parts; head which contains both terminal and function symbols and tail which only contains terminal symbols [11]. Usually, the head length h is set by the user, whilst, the tail length tt is calculated by the formula $tt = h*(n - 1) + 1$, where n represents the maximum number of arguments of the functions.

Consider a chromosome comprising of a set of symbols of function $F = \{\times, /, +, -\}$ and terminals $T = \{a, b\}$. In this example, $n = 2$ because the maximum arity of the function is two arguments. If we set the head length $h = 10$, then $tt = 11$, and the chromosome length will be $h + tt = 10 + 11 = 21$. Assume the solution (chromosome) is randomly generated, one possible example is as follows [11]: GEP_gene = $+*ab - ab + aab + ababbbababb$ and its corresponding expression tree is: GEP_expression = $a + b * ((a + b) - a)$.

There are five components in GEP, namely the function set (F), terminal set (T), fitness function (FF), GEP parameters, and stopping condition [11]. To evaluate the fitness of individuals, chromosomes are firstly translated into expression trees following the breadth-first form as follows [11].

1) Scan the chromosome string one-by-one from left to right.
2) The first element represents the node of the corresponding tree and other strings are written in a left to right manner on each lower level.
3) If the scanned element is a function (F) with n ($n >= 1$) arguments, then the next n elements are attached below it as its n children. If the scanned element is a terminal (T), then it will form a leaf of the corresponding tree.

This process is repeated until all leaves in the tree are from the terminal set (T) only. Next, the corresponding programs are executed on the user-defined problem, and their fitness values are calculated. Algorithm 1 presents the pseudocode of GEP. As in standard GA, GEP starts with a population of solutions (randomly generated). Each individual (chromosome) in the population employs the head–tail encoding method which ensures the validity of the generated solution. All chromosomes are then translated into expression trees, and executed to obtain their fitness values. Based on their fitness values, some individuals are selected by the selection mechanism (e.g., roulette wheel selection) to form the new generation by using the following genetic operators [11].

1) *Crossover:* Exchanges elements between two randomly selected genes from the chosen parents. In this paper, we employ one-point crossover operator. This operator randomly selects a point in both parents and then swaps all data beyond the selected points between the parents [11].
2) *Mutation:* Occurs at any position in the generated chromosome as long as it respects the gene rules such that the elements in the head part can be changed into both terminals and functions, whilst, the elements in the tail part can be changed into terminals only. In this paper, we use a point mutation operator. This mutation operator scans chromosome genes one-by-one and, based on the mutation probability, changes the value of the current gene in such a way that if the current gene is in the head part it can be changed into both terminals and functions, whilst, if it is in the tail part it can be changed into terminals only.
3) *Inversion:* Reverses the sequence of elements within the head or tail. Based on the inversion probability rate, randomly selects a point in either the head or the tail of a given chromosome and reverses the sequence beyond the selected point.

The newly generated chromosomes are then evaluated to calculate their fitness values, and added into the next generation. Following roulette wheel selection the fittest individuals are always copied into the next generation (i.e., elitism is employed). This process is executed until a stopping condition is satisfied.

b) Gene expression programming algorithm for evolving acceptance criterion: We propose a GEP framework to automatically generate the acceptance criterion which is specific to a given problem instance within the hyper-heuristic framework. A key decision in the design of the proposed GEP framework is the definition of the T, F, and the FF.

In order to be able to use the proposed GEP framework across a variety of problems, we keep the definition of the T, F, and the FF as general, and simple, as possible. This ensures that the proposed framework can be used to solve different classes of problems rather than just those considered in this paper. It can also be easily integrated into other meta-heuristic algorithms. The function (F) and terminal (T) sets that have been used in the proposed GEP framework are presented in Table I.

The main role of GEP is to evolve a population of individuals, each encoding an acceptance criterion. To assess the performance of an acceptance criterion, the hyper-heuristic framework is run on the given problem instance with the evolved acceptance criterion. Specifically, the proposed hyper-heuristic invokes the following steps: it calls the heuristic selection mechanism to select a LLH, which is applied to the current solution, and then calculates the quality of the

TABLE I
TERMINAL AND FUNCTION SETS OF GEP-HH

Terminal set	
terminal	***description***
delta	*The change in the solution quality*
PF	*The quality of the previous solution*
CF	*The quality of the current solution*
CI	*Current iteration*
TI	*Total number of iterations*
Function set	
function	***description***
+	*Add two inputs*
-	*Subtract the second input from the first one*
*	*Multiply two inputs*
e^x	*The result of the child node is raised to its power (Euler's number)*
%	*Protected divide function, i.e., change the division by zero into 0.001*

generated solution. If the generated solution is better than the current one, the current one is replaced (accepted). If not, the hyper-heuristic will call the acceptance criterion that has been generated by the GEP framework and execute the corresponding program. Then, the generated solution is accepted if the exponential value of the utilized acceptance criterion returns a value less or equal to 0.5 (the *exp* function returns values between 0 and 1). Otherwise, the solution will be rejected. In the literature, a value of 0.5 was suggested [30] but for different domains. In our paper, the evolved programs in our hyper-heuristic framework are utilized as an acceptance criterion rather than as a constructive heuristic as in [30]. The value 0.5 was also determined based on preliminary testing. The proposed hyper-heuristic framework will keep using the utilized acceptance criterion, which is generated by the GEP framework, for a predefined number of iterations (it stops after 10 consecutive nonimprovement iterations, determined by preliminary experimentation, see Section IV-A).

When the stopping condition is satisfied, the performance of the utilized acceptance criterion is assessed by calculating its FF. The FF, which is problem independent, is used to assess the performance of the current acceptance criterion.

In this paper, we adapt the idea that was used to control the population size in an evolutionary algorithm [31] to evaluate the fitness of the current acceptance criterion. The probability of each acceptance criterion is updated with respect to the quality of the best solution returned after the stopping condition is satisfied.

Let $Ac[]$ be the array of the fitness value of selecting the acceptance criterion, f_i and f_b represents the quality of the initial and returned solutions, *NoAc* represents the number of acceptance criteria, or the population size of GEP, respectively. Then, if the application of the i-th acceptance criterion leads to an improvement in the solution quality, the fitness of the i-th acceptance criterion is updated as follows: $Ac[i] = Ac[i] + \Delta$ where $\Delta = (f_i - f_b)/(f_i + f_b)$, $\forall j \in \{1, \ldots, NoAc\}$ and $j \neq i$, $Ac[j] = Ac[j] - (\Delta/(NoAc - 1))$. Otherwise (if the solution cannot be improved), $Ac[i] = Ac[i] - |(\Delta * \alpha)|$ where $\alpha = Current\_Iteration/Total\_Iteration$, $\forall j \in \{1, \ldots, NoAc\}$ and $j \neq i, Ac[j] = Ac[j] + (|\Delta| * \alpha/(NoAc - 1))$. We decrease the fitness value of the other acceptance criteria (individuals) in order to decrease their chances of being selected. Initially, the fitness of each acceptance criterion is calculated by executing their corresponding program.

Algorithm 1: Pseudocode of GEP Algorithm

```
Set number of generations, population_size, Head_length,
Tail_length, p_crossover,
p_muataion, Inversion_size
population ← initializepopulation(population_size,
Head_length, Tail_length)
for each sol_i ∈ population do
    // translate the chromosome into expression tree//
    sol_i−et ←TranslateBreadthFirst(Sol_i−genes)
    // execute the corresponding expression tree//
    sol_i−cost ←execute (sol_i−et)
end
sol_best ← SelectBestSolution(population_size)
while stopping condition not true do
    // parent selection process //
    parent_i ← SelectParents(population_size)
    parent_j ← SelectParents(population_size)
    // crossover operator //
     child_1 ← Crossover(parent_i, parent_j,p_crossover)
    child_2 ← Crossover(parent_i, parent_j, p_crossover)
     // mutation operator //
    child_1m ← Mutation(child_1, p_muataion)
    child_2m ← Mutation(child_2, p_muataion)
     // inversion operator //
    child_1−inversion ← Inversion(child_1m, Inversion_size)
    child_2−inversion ← Inversion(child_2m, Inversion_size)
     // translated the chromosome into expression tree//
    child_1−et ← TranslateBreadthFirst(child_1−inversion)
    child_2−et ← TranslateBreadthFirst(child_2−inversion)
     // execute the corresponding expression tree//
    child_1−cost ← execute(child_1−et)
    child_2−cost ← execute(child_2−et)
    //update the population //
    population ← populationUpdateRWS (child_1−cost,
    child_2−cost)
end
return the best solution
```

C. Low-Level Heuristics

The low level of the proposed hyper-heuristic framework contains a pool of problem-specific heuristics. The aim of the LLHs is to explore the neighborhoods of the current solution by altering the current solution (perturbation). Details of these heuristics are presented in the problem description sections (Sections IV-B1a and IV-B2a).

IV. EXPERIMENTAL SETUP

In this section, we discuss the parameter settings of GEP-HH and briefly describe the combinatorial optimization problems that we used to evaluate GEP-HH. Some tables and figures are presented in a supplementary file.

TABLE II
GEP-HH PARAMETERS

#	*Parameters*	*Possible Range*	*Suggested Value by REVAC*
1	Population size	5-50	10
2	Number of generations	10-200	100
3	Crossover probability	0.1-0.9	0.7
4	Mutation probability	0.1-0.9	0.1
5	Inversion rate	0.1-0.9	0.1
6	Head length h	2-40	5
7	Selection mechanism	-	Roulette Wheel Sampling with Elitism
8	Crossover type	Two/multi/ one point	One point
9	No. of consecutive non improvement iterations	0-100	10
10	γ in the PH test	1-50	14
11	The scaling factor C	1-100	7
12	The sliding window size W	2-100	20
13	The tolerance parameter δ	0.1-1.00	0.15

A. GEP-HH Parameter Settings

Finding the best parameter values is a tedious, and time consuming task that often requires considerable expertise and experience [32], [33]. In this paper, the relevance estimation and value calibration (REVAC) [32] is used. REVAC is a tool for parameter optimization that takes all parameters, and their possible values, and suggests the appropriate value for each parameter [34], taking into consideration the solution quality as well as the computational time needed to achieve good quality solutions. The running time for each instance is fixed to 20 seconds and the number of iterations performed by REVAC is fixed at 100 iterations (see [32] for more details). Table II lists the parameter settings of GEP-HH that have been used across all problem domains.

B. Problem Descriptions

Two well-known combinatorial optimization problems have been chosen as the test domains in this paper (exam timetabling and dynamic vehicle routing). In addition, the generality of the proposed hyper-heuristic is also verified using the CHeSC competition dataset [14], which provides access to six problem domains (see the supplementary file).

1) Application I: Exam Timetabling: The exam timetabling problem involves allocating a set of exams into a limited number of timeslots and rooms [35]. The allocation process is subject to a set of hard and soft constraints. The aim of the optimization process is to minimize soft constraint violations as much as possible and satisfy the hard constraints [35]. The quality of a timetable is measured by how many soft constraints, possibly weighted, are violated. In this paper, we test GEP-HH on the recently introduced exam timetabling instances from the 2007 International Timetabling Competition (ITC 2007) [12]. Tables A1 and A2 (see the supplementary file) present the hard and soft constraints, and Table A3 (see the supplementary file) shows the main characteristics of these instances. The proximity cost [12], which represents the soft constraint violations, is used to calculate the penalty cost (objective function value) of the generated solution.

a) Exam timetabling: Initial solution and the low level heuristics: As mentioned in Section III-A, GEP-HH starts with a complete initial solution and iteratively improves it. The initial solution is generated by hybridizing three graph coloring heuristics proposed in [36]. The set of LLHs, which are commonly used in the scientific literature [35], are as follows.

- Nbe_1: *Select one exam at random and move it to any feasible timeslot/room.*
- Nbe_2: *Select two exams at random and swap their timeslots (if feasible).*
- Nbe_3: *Select two timeslots at random and swap all their exams.*
- Nbe_4: *Select three exams at random and exchange their timeslots randomly (if feasible).*
- Nbe_5: *Move the exam leading to the highest soft constraint violation to any feasible timeslot.*
- Nbe_6: *Select two exams at random and move them to any feasible timeslots.*
- Nbe_7: *Select one exam at random, then randomly select another timeslot and apply the Kempe chain neighborhood operator.*
- Nbe_8: *Select one exam at random and move it to a randomly selected room (if feasible).*
- Nbe_9: *Select two exams at random and swap their rooms (if feasible).*

2) Application II: Dynamic Vehicle Routing Problems: The DVRP [13] is a variant of the classical, and static, VRP [37], where the aim in both versions is to minimize the cost of routes to serve a set of customers. In contrast to the static VRP, where the problem information is known in advance, in DVRP not all information is known at the start, and changes might occur at any time. DVRP can be modeled as a VRP with the difference that new orders from customers might appear during the optimization process.

The goal is to find a feasible set of routes that do not violate any hard constraints and minimizes the travel distance as far as possible. The hard constraints that must be satisfied are [37]: 1) each vehicle starts, and terminates its route at the depot; 2) the total demand of each route does not exceed the vehicle capacity; 3) each customer is visited exactly once by exactly one vehicle; and 4) the duration of each route does not exceed a global upper bound. The quality of the generated solution is represented as the total traveling distance (see [37] for more details).

In DVRP, the problem information can be changed over time [13], [38], i.e., new orders arrive once deliveries have started. Such changes need to be included in the current schedule as follows: when new orders appear, they should be integrated into a current route or a new route is created for them. As a result, some customers in the current solution may be rescheduled in order to accommodate these changes. The 21 DVRP instances that were originally introduced in [13] and further refined in [38] are used as the benchmark to assess whether the proposed hyper-heuristic framework can perform well on dynamic problems (see Table A4 in the supplementary file).

TABLE III
RESULTS OF GEP-HH ON THE ITC 2007 EXAM TIMETABLING DATASETS COMPARED TO ITC 2007 WINNERS AND POST-ITC 2007 METHODS

GEP-HH				*ITC 2007 Winners*					*Hyper-heuristics*			*Bespoke methods*		
Instances	*Best*	Δ (%)	*Rank*	$Witc_1$	$Witc_2$	$Witc_3$	$Witc_4$	$Witc_5$	$HHitc_6$	$HHitc_7$	$HHitc_8$	$Bitc_9$	$Bitc_{10}$	$Bitc_{11}$
Dataset 1	4371	0.02	2	**4370**	5905	8006	6670	12035	6235	8559	6234	4775	**4370**	4633
Dataset 2	**380**	*	1	400	1008	3470	623	3074	2974	830	395	385	385	405
Dataset 3	**8965**	*	1	10049	13862	18622	-	15917	15832	11576	13002	8996	9378	9064
Dataset 4	15381	0.08	2	18141	18674	22559	-	23582	35106	21901	17940	16204	**15368**	15663
Dataset 5	**2909**	*	1	2988	4139	4714	3847	6860	4873	3969	3900	2929	2988	3042
Dataset 6	25750	0.03	2	26950	27640	29155	27815	32250	31756	28340	27000	**25740**	26365	25880
Dataset 7	**4037**	*	1	4213	6683	10473	5420	17666	11562	8167	6214	4087	4138	4037
Dataset 8	7468	0.09	2	7861	10521	14317	-	16184	20994	12658	8552	7777	7516	**7461**

Note: Best results are shown in bold. Δ (%) represents the relative error in percentage from the best result. "" means GEP-HH result is better than other methods. "-" indicates no feasible solution has been found.*

In this paper, we have used the same model presented in [38]–[40]. In this model, the DVRP is decomposed into a (partial) sequence of static VRPs and then they are successively solved by the proposed GEP-HH. The model parameters are presented in Table A5 (see the supplementary file), which is the same as in [39].

a) DVRP: Initial solution and the low level heuristics: The initial feasible solution is constructed by generating a random permutation of orders which missed the service from the previous working day [40]. The LLHs that we employ in GEP-HH for the DVRP instances are the most common ones used to solve the capacitated vehicle routing problems in [37]. They are described as follows.

- Nbv_1: *Select one customer randomly and move it to any feasible route.*
- *Nbv2: Select two customers at random and swap their routes.*
- *Nbv3: Select one route at random and reverse a part of a tour between two selected customers.*
- *Nbv4: Select and exchange routes of three customers at random.*
- *Nbv5: Select one route at random and perform the 2-opt procedure.*
- *Nbv6: Perform the 2-opt procedure on all routes.*
- *Nbv7: Select two distinct routes at random and swap a portion of the first route with the first portion of the second route.*
- *Nbv8: Select two distinct routes at random and from each route select one customer. Swap the adjacent customer of the selected one for both routes.*
- *Nbv9: Select two distinct routes at random and swap the first portion with the last portion.*
- *Nbv10: Select one customer at random and move it to another position in the same route.*

V. COMPUTATIONAL RESULTS AND DISCUSSION

This section is divided into two subsections. The first Section V-A compares the results of GEP-HH with the state-of-the-art of hyper-heuristic and bespoke methods. The second Section V-B discusses the performance of the GEP-HH across all the problem domains. In order to make the comparison as fair as possible, for all experimental tests, the execution time is fixed, with the stopping condition, determined as follows.

1) For exam timetabling [12] and the HyFlex problem domains [14] the execution time is determined by using the benchmark software provided by the organizers to ensure fair comparisons between researchers using different platforms. We have used this software to determine the allowed execution time using our computer resources (i.e., 10 min).
2) For dynamic vehicle routing, the execution time is fixed as in [39] and [40] (i.e., 750 seconds).

To gain sufficient experimental data, for all experimental tests, we executed GEP-HH for 51 independent runs with different random seeds for exam timetabling and DVRP problems and, 31 runs for the HyFlex domains (adhering to the competition rules [14]).

A. Comparing GEP-HH Results With the State-of-the-Art

This section presents the performance comparison between GEP-HH and the state-of-the-art of hyper-heuristics as well as other bespoke methods that have been tested on ITC 2007 and DVRP. The results of HyFlex problem domains (adhering to all CHeSC rules) are presented in the supplementary file.

1) Comparison of GEP-HH Results With the State-of-the-Art Methods for ITC 2007: In this section, we assess the computational results of GEP-HH against the best known results in the scientific literature. The considered methods are as follows.

1) The ITC 2007 winners: $Witc_1$ [41], $Witc_2$ [42], $Witc_3$ [43], $Witc_4$ [44], and $Witc_5$ [45].
2) The Post-ITC 2007 methods: hyper-heuristics ($HHitc_6$ [46], $HHitc_7$ [47], and $HHitc_8$ [48]) and bespoke methods ($Bitc_9$ [49], $Bitc_{10}$ [50], and $Bitc_{11}$ [50]).

The best and the instance rankings of GEP-HH results are presented and compared with the ITC 2007 winners and Post-ITC 2007 methods in Table III (best results are shown in bold). In addition, for each instance, the relative error in percentage [$\Delta(\%)$] from the best known value found in the literature is also calculated, $\Delta(\%) = ((a-b)/b) \times 100$, where a is the best result returned over 51 independent runs by GEP-HH and b is the best known value found in the literature. It should be noted that the execution time (i.e., 10 min) of all the compared methods (GEP-HH, ITC 2007 winners and post ITC 2007 methods) are determined by the benchmark software provided by the ITC 2007 organizers [12].

TABLE IV
BEST RESULTS OF GEP-HH ON DVRP INSTANCES COMPARED TO THE LITERATURE

	GEP-HH			*ANT*	*GRASP*	*GA*	*TS*	*GA-HH*
Instances	*Best*	Δ (%)	*Rank*	*Best*	*Best*	*Best*	*Best*	*Best*
c100	**957.157**	*	1	973.26	1080.33	961.1	997.15	975.17
c100b	890.11	0.92	2	944.23	978.39	**881.92**	891.42	956.67
c120	**1237.61**	*	1	1416.45	1546.5	1303.59	1331.8	1245.94
c150	1322.13	0.29	2	1345.73	1468.36	1348.88	**1318.22**	1342.91
c199	**1642.1**	*	1	1771.04	1774.33	1654.51	1750.09	1689.52
c50	581.05	1.77	2	631.3	696.92	**570.89**	603.57	597.74
c75	**956.17**	*	1	1009.38	1066.59	981.57	981.51	979.25
f134	**14563.4**	*	1	15135.51	15433.84	15528.81	15717.9	14801.55
f71	281.62	0.49	2	311.18	359.16	301.79	**280.23**	288
tai100a	**2180.24**	*	1	2375.92	2427.07	2232.71	2208.85	2227.51
tai100b	**2058.21**	*	1	2283.97	2302.95	2147.7	2219.28	2183.35
tai100c	1525.31	0.67	2	1562.3	1599.19	1541.28	**1515.1**	1656.92
tai100d	1865.78	1.69	2	2008.13	1973.03	**1834.6**	1881.91	1834.4
tai150a	**3290.12**	*	1	3644.78	3787.53	3328.85	3488.02	3346.08
tai150b	**2864.96**	*	1	3166.88	3313.03	2933.4	3109.23	2874.83
tai150c	**2510.38**	*	1	2811.48	3110.1	2612.68	2666.28	2583.04
tai150d	**2901.61**	*	1	3058.87	3159.21	2950.61	2950.83	3084.52
tai75a	**1764.45**	*	1	1843.08	1911.48	1782.91	1778.52	1769.67
tai75b	1451.31	0.05	2	1535.43	1582.24	1464.56	1461.37	**1450.44**
tai75c	1453.28	3.34	3	1574.98	1596.17	1440.54	**1406.27**	1685.15
tai75d	1432.88	2.36	4	1472.35	1545.21	**1399.83**	1430.83	1432.87

Note: Bold fonts indicate the best results. Δ (%): represents the relative error in percentage from the best result. "" means GEP-HH result is better than other methods.*

As Table III shows, GEP-HH provides new best results for four out of eight instances. From Table III, we infer that, although GEP-HH does not obtain the best results for all instances (Datasets 1, 4, 6, and 8), overall, the quality of solutions with regard to relative error is between 0.02 and 0.09. In addition, GEP-HH obtained the second rank for these instances (Datasets 1, 4, 6, and 8). If we compare GEP-HH with the ITC 2007 winners, on seven (except Dataset 1) out of eight instances, GEP-HH produces better quality solutions compared to the ITC 2007 winners. Compared to the hyper-heuristic methods in Table III, we can see that, across all instances, GEP-HH outperforms other hyper-heuristic methods ($HHitc_6$, $HHitc_7$, and $HHitc_8$). In Table A6 (see the supplementary file), we present the average results of GEP-HH and the compared methods. Please note that only those that reported the average results are considered in the comparison. As shown in Table A6, the average results of GEP-HH are better than other methods. Thus, we can conclude that the relative error and instance ranking reveal that GEP-HH generalizes well and obtains good results (with regard to ITC 2007 instances).

To validate the performance of GEP-HH more accurately, we have also performed a multiple comparison statistical test [51] with regard to other methods (ITC 2007 winners and Post-ITC 2007 methods). To do so, we performed Friedman and Iman–Davenport tests with a critical level of 0.05 to detect whether there are statistical differences between the results of these methods [51].

The p-value of Friedman (p-value = 0.000) and Iman–Davenport (p-value = 0.000) are less than the critical level 0.05. This implies that there is a significant difference between the compared methods (GEP-HH, ITC 2007 winners and Post-ITC 2007 methods). As a result, a *post-hoc* statistical test (Holm and Hochberg statistical tests) is used to detect the correct difference between the methods (see [51] for more details). Table A7 (see the supplementary file) summarizes the average ranking (the lower the better) produced by the Friedman test for each method. GEP-HH is ranked first with $Bitc_9$, $Witc_1$, $HHitc_8$, $Witc_2$, $Witc_3$, and $Witc_5$ ranking the 2, 3, 4, 5, 6, and 7, respectively. The adjusted p-values of Holm and Hochberg statistical tests for the GEP-HH (the control method) and others in Table A8 (see the supplementary file) demonstrate that GEP-HH outperforms $Witc_5$, $Witc_3$, and $Witc_2$ (three out of six methods) with a critical level of 0.05 (adjusted p-value < 0.05) and better than $Witc_5$, $Witc_3$, $Witc_2$, $HHitc_8$, and $Witc_1$ (five out of six methods) with a critical level of 0.10 (adjusted p-value < 0.10). However, the results in Table A8 indicate that, GEP-HH does not outperform $Bitc_9$ (adjusted p-value > 0.10).

To summarize, although the results of Holm and Hochberg statistical tests (Table A8) suggest that GEP-HH is not better than $Bitc_9$, nevertheless, the results in Table III reveals that GEP-HH outperformed $Bitc_9$ on seven out of eight instances and the average result in Table A6 is much better across all instances. It is worth noting that all of the compared methods are tailor-made to obtain the best results for one or few instances only, whilst one can easily see that GEP-HH generalizes well across all instances.

2) Comparison of GEP-HH Results With the State-of-the-Art Methods for DVRP: In this section, we evaluate the performance of GEP-HH against the best available results in the scientific literature [ant colony (ANT) [38], greedy randomize adaptive search procedure (GRASP) [38], GA [40], TS [40], and genetic hyper-heuristic (GA-HH) [39]] that have been tested on DVRP. To our knowledge, only one hyper-heuristic method (GA-HH) has been tested on DVRP. The computational time of the compared methods is as follows: GEP-HH, GA, TS, and GA-HH is 750 seconds, whilst ANT

and GRASP is 1500 seconds. Table IV gives the computational results of GEP-HH [best, the relative error ($\Delta(\%)$) and instance ranking] along with best results obtained by other methods, while, Table A9 (see the supplementary file) shows the average results obtained by GEP-HH as well as the compared methods (best results are shown in bold).

Considering the best results in Table IV, we can see that GEP-HH achieved better quality results for 20 (except tai75b) out of 21 instances compared to GA-HH. Observing the best results of the bespoke methods (ANT, GRASP, GA, and TS) reported in Table IV, GEP-HH outperformed the bespoke methods on 13 problem instances, while it is inferior on eight instances. Even though GEP-HH does not outperform bespoke methods on all problem instances, the average results of GEP-HH (Table A9, see the supplementary file) are, however, much better than the bespoke methods across all instances, except instance tai75d where the average results achieved by GA are slightly better than GEP-HH. In addition, the relative error from the best known results (Table IV) of GEP-HH for instance c100b, c150, c50, f71, tai100c, tai100d, tai75b, tai75c, and tai75d which are 0.92, 0.29, 1.77, 0.49, 0.67, 1.69, 0.05, 3.34, and 2.36, respectively, are relatively small.

In addition to the above results, it is worth drawing some statistical significant conclusions regarding the performance of GEP-HH as well as the bespoke methods (ANT, GRASP, GA, TS, and GA-HH). Therefore, multiple comparison statistical tests Friedman and Iman–Davenport with a critical level of 0.05 are carried out, followed by a *post-hoc* statistical (Holm and Hochberg statistical tests) in case that the results of Friedman and Iman–Davenport are less than 0.05. Thus, since the p-value of both tests is less than the critical level 0.05, we further analyze the result to detect the correct difference among the considered methods.

Table A10 (see the supplementary file) shows the average ranking of GEP-HH as well as ANT, GRASP, GA, TS, and GA-HH produced by Friedman test (the lower the better). From this table one can observe that, GEP-HH achieved the first rank out of the six compared methods followed by GA, GA-HH, TS, ANT, and GRASP, respectively.

Table A11 (see the supplementary file) gives the adjusted p-values of Holm and Hochberg statistical tests for each comparison between GEP-HH (the controlling method) and ANT, GRASP, GA, TS, and GA-HH. The results of the adjusted p-values reveal the following: GEP-HH is statistically better than all of the bespoke methods (ANT, GRASP, GA, and TS) as well as the hyper-heuristic method (GA-HH) with a critical level of 0.05. That is, no comparison of GEP-HH, with any method obtained an adjusted p-value equal to or greater than 0.05.

The above result implies that GEP-HH outperforms the GA-HH hyper-heuristic and is competitive to, if not better (on some instances), some bespoke methods (ANT, GRASP, GA, and TS). Also, it is worth noting that the compared methods are specifically designed to produce the best results for one or, a few instances only. All of the above observations are evidence that GEP-HH is able to produce good quality results and generalize well over all instances, instead of producing good quality results for just a few instances.

B. Discussion

The numerical results presented throughout this paper demonstrate that, across different combinatorial optimization problems with fundamentally different search spaces (static and dynamic), GEP-HH achieved favorable results compared to the best available methods in the literature. The results establish that, on some instances, GEP-HH has better performance than the best available methods in the literature. Hence, a fundamental question naturally arises: why GEP-HH obtains such good results? We hypothesis that the capability of GEP-HH in dealing with different problem domains and achieving such results is due to the following two factors.

1) The ability of the proposed GEP algorithm to generate, for each instance, different acceptance criterion during the optimization process. Due to the fact that some instances of the considered problem domains have a large search space, or the search spaces are rugged and contain many local optima because of the imposed constraints, it might be that feasible regions are isolated by infeasible ones. Therefore, by generating for each instance different acceptance criterion during the instance solving process, the hyper-heuristic is capable of escaping from the local optima as well as effectively exploring the entire search space. Generating algorithm components can reduce the user intervention in finding the most effective configuration and the facilitate algorithm configurations. The success of GEP-HH on all problem domains validated our hypothesis in using GEP-HH to automatically evolve the hyper-heuristic acceptance criteria instead of using human-designed ones.
2) The integration of the PH statistical test as well as the extreme value-based reward credit assignment mechanism in the heuristic selection mechanism provided good results. Therefore, the good results obtained on all the considered problem domains validated our hypothesis that these two components help the heuristic selection to quickly select the suitable LLHs during the instance solving process.

VI. Conclusion

The work presented in this paper has proposed a new improvement-based hyper-heuristic framework, GEP-HH, for combinatorial optimization problems. GEP-HH has two levels, a high level strategy and a LLH. The latter consists of a set of human-designed low level heuristics that are used to perturb the solution of a given instance. The former has two components, the heuristic selection mechanism and the acceptance criterion. The DMAB-extreme value-based reward is utilized at the higher level to perform the task of selecting a LLH. GEP is used as an online method to generate the acceptance criterion in order to decide if the generated solution is accepted or not.

This paper has shown that it is possible to use a heuristic selection mechanism that utilizes a statistical test in determining the most suitable LLH as well as generating a different acceptance criterion for each problem instance. The efficiency,

consistency, and the generality of GEP-HH has been demonstrated across eight challenging problems, a static problem (exam timetabling), a dynamic problem (dynamic vehicle routing problems), and the HyFlex problem domains (Boolean satisfiability, 1-D bin packing, permutation flow shop, personnel scheduling, traveling salesman, and vehicle routing), which have very different search spaces. The experimental results show that GEP-HH achieves highly competitive results, if not superior to other methods, and that it generalizes well over all domains when compared to the state-of-the-art of hyper-heuristics and bespoke methods. The main contributions of this paper are as follows.

1) The development of the GEP-HH framework that utilizes an online heuristic selection mechanism which integrates a statistical test, demonstrating that this selection mechanism is capable of selecting the most appropriate LLHs using information gathered during the instance solving process.
2) The development of a framework to generate an acceptance criterion that can be integrated with any hyper-heuristic or meta-heuristic method, using GEP. This framework generates, for each instance, a different acceptance criterion during instance solving and obtains consistent, competitive results that generalize well across eight different problem domains.
3) The development of a hyper-heuristic framework that is not customized to specific problem classes and can be applied to different problems without much development effort (i.e., the user only needs to replace the set of LLHs).

In this paper, we have proposed an automatic programing generation method to generate the high level strategy component. In future work, we would also like to investigate generating the LLHs and, perhaps, placing them in competition with one another. If this were successful, we will have a complete framework that is able to tackle any problem, with very little human intervention.

REFERENCES

[1] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[2] 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.

[3] J. E. Smith, "Coevolving memetic algorithms: A review and progress report," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 6–17, Feb. 2007.

[4] E. K. Burke *et al*., "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*, M. Gendreau and J. Potvin, Eds., 2nd ed. Berlin, Germany: Springer, 2010, pp. 449–468.

[5] E. Talbi, *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: Wiley, 2009.

[6] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 99–110, Apr. 2004.

[7] E. K. Burke *et al*., "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, pp. 1695–1724, Jul. 2013.

[8] E. Ozcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal*., vol. 12, no. 1, pp. 3–23, Jan. 2008.

[9] K. Chakhlevitch and P. Cowling, "Hyperheuristics: Recent developments," in *Adaptive and Multilevel Metaheuristics*, 2008, pp. 3–29.

[10] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, "Analyzing bandit-based adaptive operator selection mechanisms," *Ann. Math. Artif. Intell.*, vol. 60, nos. 1–2, pp. 1–40, 2010.

[11] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence.* New York, NY, USA: Springer-Verlag, 2006.

[12] B. McCollum *et al*., "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS J. Comput.*, vol. 22, no. 1, pp. 120–130, 2010.

[13] P. Kilby, P. Prosser, and P. Shaw, "Dynamic VRPs: A study of scenarios," Univ. Strathclyde, Glasgow, U.K., Tech. Rep. APES-06-1998, 1998.

[14] G. Ochoa *et al*., "HyFlex: A benchmark framework for cross-domain heuristic search," in *Proc. EvoCOP*, 2012, pp. 136–147.

[15] Z. Ren, H. Jiang, J. Xuan, Y. Hu, and Z. Luo, "New insights into diversification of hyper-heuristics," *IEEE Trans. Cybern.*, Dec. 2013, to be published.

[16] E. K. Burke, G. Kendall, and E. Soubeiga, "A Tabu-search hyperheuristic for timetabling and rostering," *J. Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[17] N. R. Sabar and M. Ayob, "Examination timetabling using scatter search hyper-heuristic," in *Proc. 2nd DMO*, Kajand, Hungary, 2009, pp. 127–131.

[18] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Practice and Theory of Automated Timetabling III*, E. Burke and W. Erben, Eds. Berlin, Germany: Springer, 2001, pp. 176–190.

[19] M. Ayob and G. Kendall, "A Monte Carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine," in *Proc. InTech*, 2003, pp. 132–141.

[20] G. Kendall and M. Mohamad, "Channel assignment optimisation using a hyper-heuristic," in *Proc. IEEE CIS*, Singapore, 2004, pp. 791–796.

[21] R. Bai and G. Kendall, "An investigation of automated planograms using a simulated annealing based hyper-heuristic," in *Metaheuristics: Progress as Real Problem Solvers*. Berlin, Germany: Springer, 2005, pp. 87–108.

[22] K. A. Dowsland, E. Soubeiga, and E. Burke, "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation," *Eur. J. Oper. Res.*, vol. 179, no. 3, pp. 759–774, 2007.

[23] E. Ozcan, Y. Bykov, M. Birben, and E. K. Burke, "Examination timetabling using late acceptance hyper-heuristics," in *Proc. IEEE CEC*, Trondheim, Norway, 2009, pp. 997–1004.

[24] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," in *Proc. 12th IEEE ICON*, 2004, pp. 769–773.

[25] K. Chakhlevitch and P. Cowling, "Choosing the fittest subset of low level heuristics in a hyperheuristic framework," in *Evolutionary Computation in Combinatorial Optimization*. Berlin, Germany: Springer, 2005, pp. 23–33.

[26] M. Bader-El-Den and R. Poli, "Generating SAT local-search heuristics using a GP hyper-heuristic framework," in *Proc. 8th EA*, Tours, France, 2008, pp. 37–49.

[27] N. Sabar, M. Ayob, G. Kendall, and R. Qu, "The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, Apr. 2014, to be published.

[28] N. R. Sabar, M. Ayob, G. Kendall, and Q. Rong, "Grammatical evolution hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 840–861, Dec. 2013.

[29] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Trans. Cybern.*, Apr. 2014, to be published.

[30] 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.

[31] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS-a genetic algorithm with varying population size," in *Proc. 1st IEEE CEC*, Orlando, FL, USA, 1994, pp. 73–78.

[32] V. Nannen and A. Eiben, "Efficient relevance estimation and value calibration of evolutionary algorithm parameters," in *Proc. IEEE CEC*, Singapore, 2007, pp. 103–110.

[33] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 19–31, 2011.

[34] E. Montero, M.-C. Riff, L. Pérez-Caceres, and C. Coello Coello, "Are state-of-the-art fine-tuning algorithms able to detect a dummy parameter?" in *Parallel Problem Solving from Nature—PPSN XII*. Berlin, Germany: Springer, 2012, pp. 306–315.

[35] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, "A survey of search methodologies and automated system development for examination timetabling," *J. Sched.*, vol. 12, no. 1, pp. 55–89, 2009.
[36] M. Ayob *et al*., "Solving a practical examination timetabling problem: A case study," in *Proc*. *ICCSA*, Kuala Lumpur, Malaysia, 2007, pp. 611–624.
[37] P. Toth and D. Vigo, *The Vehicle Routing Problem*. Philadelphia, PA, USA: Society for Industrial Mathematics, 2002.
[38] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "Ant colony system for a dynamic vehicle routing problem," *J. Comb. Optim.*, vol. 10, no. 4, pp. 327–343, 2005.
[39] P. Garrido and M. C. Riff, "DVRP: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *J. Heuristics*, vol. 16, no. 6, pp. 795–834, 2010.
[40] F. T. Hanshar and B. M. Ombuki-Berman, "Dynamic vehicle routing using genetic algorithms," *Appl. Intell.*, vol. 27, no. 1, pp. 89–99, 2007.
[41] T. Müller, "ITC2007 solver description: A hybrid approach," *Ann. Oper. Res.*, vol. 172, no. 1, pp. 429–446, 2009.
[42] C. Gogos, P. Alefragis, and E. Housos, "A multi-staged algorithmic process for the solution of the examination timetabling problem," in *Proc. PATAT*, Montreal, QC, Canada, 2008, pp. 19–22.
[43] M. Atsuta, K. Nonobe, and T. Ibaraki, "ITC2007 Track 2, an approach using general CSP solver," in *Proc. PATAT*, 2008, pp. 19–22.
[44] G. De Smet, "ITC2007-examination track," in *Proc. PATAT*, 2008, pp. 19–22.
[45] A. Pillay, "Developmental approach to the examination timetabling problem," in *Proc. PATAT*, 2008, pp. 19–22.
[46] E. K. Burke, R. Qu, and A. Soghier, "Adaptive selection of heuristics for improving constructed exam timetables," in *Proc. PATAT*, 2010, pp. 136–151.
[47] N. Pillay, "Evolving hyper-heuristics for a highly constrained examination timetabling problem," in *Proc. PATAT*, 2010, pp. 336–346.
[48] N. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Appl. Intell.*, vol. 37, no. 1, pp. 1–11, 2012.
[49] C. Gogos, P. Alefragis, and E. Housos, "An improved multi-staged algorithmic process for the solution of the examination timetabling problem," *Ann. Oper. Res.*, vol. 194, no. 1, pp. 1–19, 2010.
[50] B. McCollum, P. McMullan, A. Parkes, E. K. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," in *Proc. 4th MISTA*, 2009, pp. 424–434.
[51] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, 2010.

Authors' photographs and biographies not available at the time of publication.