Accepted by IEEE TRANSACTIONS ON CYBERNETICS, 2014

A Dynamic Multi-Armed 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 work, we propose a new high level strategy for the hyper-heuristic framework. The proposed high level strategy utilizes the dynamic multi-armed bandit-extreme value based rewards 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 various well-known acceptance criteria, state of the art of 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 (CHeSC) test suite. We again demonstrate the generality of our approach when we compare against other methods that have utilized the same six benchmarks datasets from this test suite.**

***Index Terms*—Gene Expression Programming, Hyper-heuristic, Timetabling, Vehicle Routing, Dynamic Optimization.**

Nasser R. Sabar and Masri Ayob are with Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent (CAIT), Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia. email: nasser@ftsm.ukm.my, masri@ftsm.ukm.my

Graham Kendall and Rong Qu are with ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK. email: Graham.Kendall@nottingham.ac.uk, Rong.Qu@nottingham.ac.uk.

Nasser R. Sabar and Graham Kendall are also with The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia. email: Nasser.Sabar@nottingham.edu.my, Graham.Kendall@nottingham.edu.my.

I. INTRODUCTION

Meta-heuristic research communities have acknowledged the fact that meta-heuristic configurations (operators and parameter settings) play a crucial role on the algorithm performance [1], [2], [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 to a wide variety of problem domains, rather than being tailored for each problem or even each problem instance encountered. Hyper-heuristics operate on the heuristic search spaces, 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 and providing a framework to exploit this. The most common hyper-heuristic framework has two levels; a high level strategy and a set of low level heuristics. The high level strategy manages which low level heuristic to call (heuristic selection mechanism) and then decides whether to accept the returned solution (the acceptance criterion). The low level heuristics contains 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 appropriate design of the *high* level strategy and it is not surprising that much work in the development of hyper-heuristics is focused on the *high* level strategy [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], [8], [9].

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

i) **Heuristic selection mechanism**: the proposed framework utilizes the dynamic multi-armed bandit-extreme value based rewards [10] as an on-line heuristic selection mechanism. The attractive feature of a dynamic multi-armed bandit is the integration of the Page-Hinkly statistical test to determine the most appropriate heuristic for the next iteration by detecting the changes in average rewards for the current best heuristics. In addition, the extreme value-based rewards credit assignment mechanism records the historical information of each heuristic to be used during the heuristic selection process.

ii) **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.

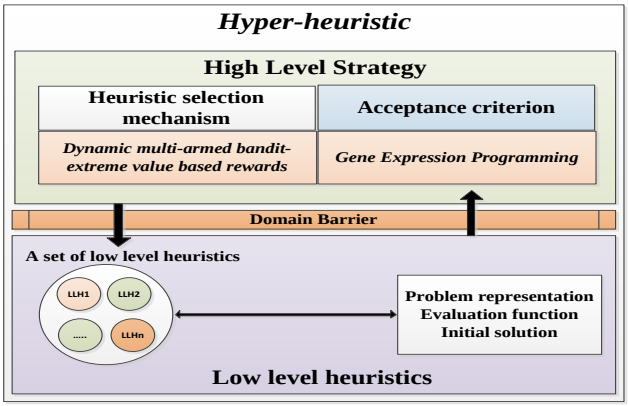

Fig. 1. The proposed gene expression programming based hyper-heuristic (GEP-HH) framework.

In the scientific literature, automatic program generation methods, such as genetic programming (GP), have been successfully utilized as a hyper-heuristics to generate heuristics for optimization problems such as 2D 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 be used on other domains. In other words, to solve a new problem domain we have to define a fundamentally different set of functions and terminals that are suitable to the problem at hand. Based on this current level of generality, in this work, 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 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, besides the fact that manual configurations may only represent a small fraction of the available search space, usually it requires a considerable amount of expertise and experience. Hence, exploring the search space using a suitable search methodology (i.e. GEP in this work) might yield a better performance compared to a manual configured search methodology. Our objectives are:

- To propose an on-line 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 low level heuristics.

- To propose an on-line 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 gene expression programming algorithm.

- 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 well-known acceptance criteria, 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 (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 strategies, which are problem independent and have no knowledge of the domain, control the selection or generation of heuristics to be called (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 *low* level heuristics 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) extra development effort. Recently, Burke et al. [4] classified hyper-heuristic frameworks based on the nature of the heuristic search spaces and the source of feedback during learning. The source of feedback can either be on-line, if the hyper-heuristic framework uses the feedback obtained during the problem solving procedure, or off-line, if the hyper-heuristic framework uses information gathered during a training phase in order to be used when solving 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 pertubative. Please note that our proposed hyper-heuristic framework can be classified as an on-line perturbative heuristic to *choose* heuristics framework.

A. Heuristic to choose heuristics

Most of the hyper-heuristic frameworks published so far have been heuristics to *choose* heuristics [4]. For a given problem instance, the role of the hyper-heuristic framework is to intelligently *choose* a low level heuristic from the low level set, so as to apply it at that decision point. 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, for example tabu search with reinforcement learning [16] and scatter search [17]. Many different acceptance criteria have also been used, including all moves [18], only improvements [18], improving and equal [18], Monte-Carlo [19], record to record travel [20], simulated annealing [21], [22], late acceptance [23], great deluge [24] and tabu search [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 low level heuristics are provided as part of the supplied framework [14]. The algorithm designer only needs to provide the higher level component (heuristic selection and acceptance criterion). Further details about the competition, including results, are available in [14].

B. Heuristic 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 genetic programming hyper-heuristics have been utilized to solve many combinatorial optimization problems including SAT [26], timetabling [27], vehicle routing [27] and bin packing [28]. 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 off-line heuristic/rule builder using a specific set of functions and terminals. Besides being computationally expensive due to the need of 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 that has 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 they will generalize to other domains?

The success of the above work, which has some resemblance to the proposed gene expression programming, is the main motivation for proposing an on-line gene expression programming 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. THE 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 low level heuristics that will be used in our framework.

A. A Perturbative based Hyper-heuristic Framework

Our on-line perturbative based hyper-heuristic framework comprises of a *high* level strategy and a set of *low* level heuristics. The *high* level strategy consists of two components, heuristic selection and acceptance criterion. The goal of the *high* level strategy is to select a low level heuristic to be applied at a given time. 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 low level heuristics, 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 low level heuristics.

The proposed hyper-heuristic iteratively explores the neighborhood of the incumbent solution, seeking for improvement. Given a set of low level heuristics (*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 low level heuristic 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 another iteration will start. This process is repeated for a certain number of

iterations. In what follows we discuss the components of the proposed high level strategy.

B. High Level Strategy

In this work, we propose a new high level strategy, as shown in Fig. 2. It has two components: heuristic selection mechanism (dynamic multi-armed bandit-extreme value-based rewards) and an acceptance criterion (gene expression programming).

1) The heuristic selection mechanism

The heuristic selection mechanism successively invokes two tasks, credit assignment (extreme value-based rewards [10]) and heuristic selector (dynamic multi-armed bandit mechanism [10]). The credit assignment mechanism maintains a value (reward) for each low level heuristic and these values are used by the heuristic selector mechanism to decide which heuristic to be executed at each decision point.

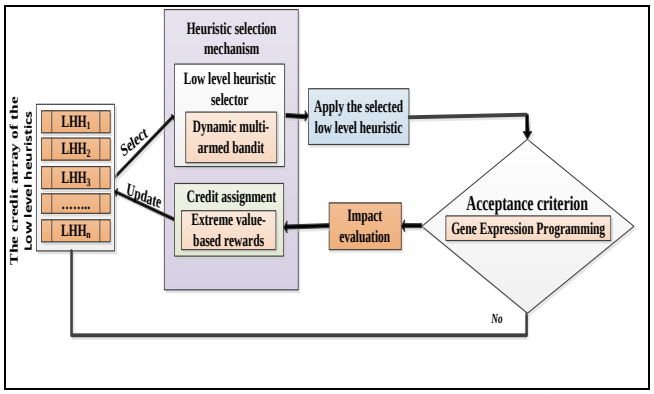

Fig. 2. The high level strategy within the proposed GEP-HH framework

a) *The credit assignment mechanism: Extreme value-based rewards*

The credit assignment mechanism maintains a value (reward) for each low level heuristic (*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 employed acceptance criterion. In this work, the extreme value-based reward [10] is employed as the credit assignment mechanism. Low level heuristics, 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. Low level heuristics 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 low level heuristic is selected, its improvement compared to the current solution is computed. The improvement (PI_{LLH}) of the applied low level heuristic 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 low level heuristic (*LLH*), $PI_{LLH} = (f_1\text{-}f_2 / f_1) * 100$. PI_{LLH} is then saved for that low level heuristic, at the corresponding iteration for a sliding time window of size *W*, using First in, First Out (*FIFO)*, i.e. improvements of the selected low level heuristic in the last *W* iterations are saved. The credit of any low level heuristic (*r*) is then set as the maximum value in its corresponding sliding window as follows:

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

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 multi-armed bandit mechanism*

Based on their previous performance (assigned credit value by the credit assignment mechanism) and the number of times that the low level heuristics have been applied, the heuristic selection mechanism selects low level heuristics to be applied to the current solution. In this work, we use the dynamic multi-armed bandit (DMAB) mechanism as an on-line heuristic selector mechanism [10]. The attractive feature of DMAB is the integration of the Page-Hinkly (PH) statistical test to detect the changes in average rewards of the current best low level heuristic. DMAB is based on an upper confidence bound strategy which deterministically selects a low level heuristic with maximum accumulative rewards. A low level heuristic is selected based on its empirical rewards and the number of times it has been applied up to current time step.

Formally, let *k* denote the number of available low level heuristics, each one having some unknown probability. DMAB selects the best low level heuristic that maximizes the accumulative rewards over time. Each low level heuristic is associated with its empirical rewards $q_{i,t}$ (2) (the average rewards r_i obtained by the *i*-th low level heuristic up to time *t*) and a confidence level $n_{i,t}$ (the number of times that the *i*-th low level heuristic has been applied up to time *t*).

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

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

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

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

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

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

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

2) The 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 low level heuristic has been applied [8]. In this work, we propose a gene expression programming framework to evolve the acceptance criterion for our hyper-heuristic. It is implemented as an on-line acceptance criterion generation method that takes the quality of the previous solution, current solution and 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 gene expression programming 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*

Gene expression programming (GEP) [11] is a form of genetic programming (GP) that uses the advantage of both genetic algorithms (GA) and genetic programming (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 a parse tree 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 genetic programming, 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 = {*, /, +, -} 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+ababbbabab 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, 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]:

- Scan the chromosome string one by one from left to right.
- 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.
- 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]:

- Crossover: exchanges elements between two randomly selected genes from the chosen parents. Both one-point and two-point crossover operators can be used. In this work, a one-point crossover operator is employed. In one-point crossover, first randomly select a point in both parents and then swap all data beyond the selected points between the parents [11].
- 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 terminal and function, whilst, the elements in the

tail part can be changed into terminals only. In this work, we use a point mutation operator. This mutation operator scans chromosome genes one by one and, based on the mutation probability, change the value of current gene in such a way that if the current gene is in the head part it can be changed into both terminal and function, whilst, if it is in the tail part it can be changed into terminals only.

- Inversion: reverses the sequence of elements within the head or tail. Based on the inversion probability rate, randomly select a point in either 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*

In this work, 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 terminal set (*T*), function set (*F*) and the fitness function.

In order to be able to use the proposed GEP framework across a variety of problems, we keep the definition of the terminal set (*T*), function set (*F*) and the fitness function 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 work, and can 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 1.

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 low level heuristic, which is applied to the current solution, and calculates the quality of the 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 is 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 (if the exponential value of the utilized acceptance criterion is greater than 0.5). In the literature, a value of 0.5 was suggested [28] but for different domains. In our work, the evolved programs in our hyper-heuristic framework are utilized as an acceptance criterion rather than as a constructive heuristic as in [28]. 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 GEP framework, for a pre-defined number of iterations (it stops after 10 consecutive non improvement iterations, determined by preliminary experimentation, see IV.A).

Algorithm 1: Pseudocode of GEP algorithm

Set number of generations, $population_{size}$, $Head_{length}$, $Tail_{length}$, $p_{crossover}$, $p_{muataion}$, $Inversion_{size}$
$population \leftarrow initializepopulation(population_{size}, Head_{length}, Tail_{length})$
foreach $sol_i \in$ population **do**
  // **translate the chromosome into expression tree**//
  $sol_{i\text{-}et} \leftarrow TranslateBreadthFirst(Sol_{i\text{-}genes})$
  // **execute the corresponding expression tree**//
  $sol_{i\text{-}cost} \leftarrow execute\ (sol_{i\text{-}et}\ )$
end
$sol_{best} \leftarrow SelectBestSolution(population_{size})$
while stopping condition not true **do**
  // **parent selection process** //
  $parent_i \leftarrow SelectParents(population_{size})$
  $parent_j \leftarrow SelectParents(population_{size})$
  // **crossover operator** //
  $child_1 \leftarrow Crossover(parent_i, parent_j, p_{crossover})$
  $child_2 \leftarrow Crossover(parent_i, parent_j, p_{crossover})$
  // **mutation operator** //
  $child_{1m} \leftarrow Mutation(child_1, p_{muataion})$
  $child_{2m} \leftarrow Mutation(child_2, p_{muataion})$
  // **inversion operator** //
  $child_{1\text{-}inversion} \leftarrow Inversion(child_{1m}, Inversion_{size})$
  $child_{2\text{-}inversion} \leftarrow Inversion(child_{2m}, Inversion_{size})$
  // **translated the chromosome into expression tree**//
  $child_{1\text{-}et} \leftarrow TranslateBreadthFirst(child_{1\text{-}inversion})$
  $child_{2\text{-}et} \leftarrow TranslateBreadthFirst(child_{2\text{-}inversion})$
  // **execute the corresponding expression tree**//
  $child_{1\text{-}cost} \leftarrow execute(child_{1\text{-}et})$
  $child_{2\text{-}cost} \leftarrow execute(child_{2\text{-}et}\ )$
  //**update the population** //
  $population \leftarrow populationUpdateRWS\ (child_{1\text{-}cost}, child_{2\text{-}cost})$
end
return the best solution

TABLE 1 THE 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*

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

In this work, we adapt the idea that was used to control the population size in an evolutionary algorithm [29] 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.

C. *Low Level Heuristics*

The low level of the proposed hyper-heuristic framework contains a pool of problem-specific heuristics. The aim of the low level heuristics 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-B-1-a and IV-B-2-a).

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*. Please note that, to save space, some tables and figures are presented in a supplementary file.

A. *GEP-HH Parameter Settings*

Finding the best parameter values is a tedious, and time consuming task that often requires considerable expertise and experience [30], [31]. In this work, the Relevance Estimation and Value Calibration (REVAC) [30] 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 [32]. 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 [30] for more details). Table 2 lists the parameter settings of *GEP-HH* that have been used across all problem domains.

TABLE 2 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
8	Crossover type	Two/multi/ one point	Sampling with Elitism 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

B. *Problem Descriptions*

Two well-known combinatorial optimization problems have been chosen as the test domains in this work (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 [33]. 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 [33]. The quality of a timetable is measured by how many soft constraints, possibly weighted, are violated. In this work, we test *GEP-HH* on the recently introduced exam timetabling instances from the 2007 International Timetabling Competition (ITC 2007) [12]. Tables 3 and 4 (see the supplementary file) present the hard and soft constraints, and Table 5 (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 [34]. The set of low level heuristics, which are commonly used in the scientific literature [33], 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 dynamic vehicle routing problem (DVRP) [13] is a variant of the classical, and static, VRP [35], 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 [35]: i) each vehicle starts, and terminates its route at the depot, ii) the total demand of each route does not exceed the vehicle capacity, iii) each customer is visited exactly once by exactly one vehicle, and iv) 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 [35] for more details).

In DVRP, the problem information can be changed over time [13], [36], i.e. new orders are revealed over time. 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 [36] are used as the benchmark to assess whether the proposed hyper-heuristic framework can perform well on dynamic problems (see Table 6 in the supplementary file).

In this work, we have used the same model presented in [37], [36], [38]. 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 7 (see the supplementary file), which is the same as in [37].

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 [38]. The low level heuristics that we employ in *GEP-HH* for the DVRP instances are the most common ones used to solve the capacitated vehicle routing problems in the literature [35]. 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 DISCUSSIONS

This section is divided into two subsections. The first section (V-A) is devoted to compare 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:

- For exam timetabling [12] and 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 minutes).
- For dynamic vehicle routing, the execution time is fixed as in [37] and [38] (i.e. 750 seconds).

To gain sufficient experimental data, for all experimental tests, we executed *GEP-HH* and the tested hyper-heuristic variants (implemented herein) 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) The 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:

- The ITC 2007 winners :$Witc_1$ [40], $Witc_2$ [41], $Witc_3$ [42], $Witc_4$ [43] and $Witc_5$ [44])
- The Post-ITC 2007 methods: hyper-heuristics ($HHitc_6$ [45], $HHitc_7$ [46] and $HHitc_8$ [47]) and bespoke methods ($Bitc_9$ [48], $Bitc_{10}$ [49] and $Bitc_{11}$ [49]).

The best and the instances ranking of *GEP-HH* results are presented and compared with the ITC 2007 winners and Post-ITC 2007 methods in Table 8 (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, $\Delta(\%)=((a-b)/b) * 100$, 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 minutes) 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].

As Table 8 shows, *GEP-HH* provides new best results for 4 out of 8 instances. From Table 8, 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 7 (except Dataset 1) out of 8 instances, *GEP-HH* produces better quality solutions compared to the ITC 2007 winners. Compared to the hyper-heuristic methods in Table 8, we can see that, across all instances, *GEP-HH* outperforms other hyper-heuristic methods ($HHitc_6$, $HHitc_7$ and $HHitc_8$). In Table 9 (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 9, 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 [39] 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 [39].

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 [39] for more details). Table 10 (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 11 (see the supplementary file) demonstrate that *GEP-HH* outperforms $Witc_5$, $Witc_3$ and $Witc_2$ (3 out of 6 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 $Witc_1$ (5 out of 6 methods) with a critical level of 0.10 (adjusted *p*-value < 0.10). However, the results in Table 11 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 11) suggest that *GEP-HH* is not better than $Bitc_9$, nevertheless, the results in Table 8 reveals that *GEP-HH* outperformed $Bitc_9$ on 7 out of 8 instances and the average result in Table 9 is much better across all instances. It 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.

TABLE 8 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.*

2) The 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*) [36], greedy randomize adaptive search procedure (*GRASP*) [36], genetic algorithms (*GA*) [38], tabu search (*TS*) [38] and genetic hyper-heuristic (*GA-HH*) [37]) 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 12 gives the computational results of *GEP-HH* (best, the relative error (Δ(%)) and instance ranking) along with best results obtained by other methods, while, Table 13 (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 12, 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 12, *GEP-HH* outperformed the bespoke methods on 13 problem instances, while it is inferior on 8 instances. Even though *GEP-HH* does not outperform bespoke methods on all problem instances, the average results of *GEP-HH* (Table 13, 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 12) 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 14 (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 15 (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, if not better (on some instances), to 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.

TABLE 12 THE 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.*

B. Discussion

The numerical results presented throughout this work 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 gene expression programming 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 such *IO*, *SA*, *GD* and *TS*.

2- The integration of the Page-Hinkly statistical test as well as the extreme value-based reward credit assignment mechanism in the heuristic selection mechanism provided good results. As shown, and analyzed, throughout the results section, the use of the Page-Hinkly statistical test and extreme value-based reward credit assignment mechanism with the heuristic selection mechanism has a positive impact and produced good results compared to other heuristic selection mechanisms. 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 low level heuristics during the instance solving process.

VI. CONCLUSIONS

The work presented in this paper has proposed a new improvement based hyper-heuristic framework, gene expression programming based hyper-heuristic (GEP-HH), for combinatorial optimization problems. *GEP-HH* has two levels, a high level strategy and a low level heuristic. 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 dynamic multi-armed bandit-extreme value based rewards is utilized at the higher level to perform the task of selecting a low level heuristic. Gene expression programming is used as an on-line method to generate the acceptance criterion in order to decide if the generated solution is accepted or not.

This work has shown that it is possible to use a heuristic selection mechanism that utilizes a statistical test in determining the most suitable low level heuristic 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, one dimensional 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 other well-known acceptance criteria (*IO*, *SA*, *GD* and *TS*) as well as state of the art of hyper-heuristics and bespoke methods. The main contributions of this work are:

- The development of the *GEP-HH* framework that utilizes an on-line heuristic selection mechanism which integrates a statistical test, demonstrating that this selection mechanism is capable of selecting the most appropriate low level heuristics using information gathered during the instance solving process.

- The development of a framework to generate an acceptance criterion that can be integrated with any hyper-heuristic or meta-heuristic method, using gene expression programming. 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.

- The development of a hyper-heuristic framework that is not customized to specific problems classes and can be applied to different problems without much development effort (i.e. the user only needs to replace the set of low level heuristics).

In this work, 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 low level heuristics 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 Transactions on Evolutionary Computation, ,* vol. 3, pp. 124-141, 1999.

[2] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: a comparative study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,* vol. 36, pp. 141-152, 2006.

[3] J. E. Smith, "Coevolving Memetic Algorithms: A Review and Progress Report," *IEEE Transactions on Systems, Man, and Cybernetics, Part B:,* vol. 37, pp. 6-17, 2007.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A Classification of Hyper-heuristic Approaches," in *Handbook of Metaheuristics*. vol. 146, M. Gendreau and J. Potvin, Eds., 2nd ed: Springer, 2010, pp. 449-468.

[5] E. Talbi, *Metaheuristics: From Design to Implementation*: Wiley online Library, 2009.

[6] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation, ,* vol. 8, pp. 99-110, 2004.

[7] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A Survey of the State of the Art," *Journal of the Operational Research Society, to appear,* 2013.

[8] E. Ozcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal.,* vol. 12, pp. 3-23, 2008.

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

[10] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, "Analyzing bandit-based adaptive operator selection mechanisms," *Annals of Mathematics and Artificial Intelligence,* pp. 1-40, 2010.

[11] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence (Studies in Computational Intelligence)*: Springer-Verlag New York, Inc., 2006.

[12] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting

the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, pp. 120-130, 2010.

[13] P. Kilby, P. Prosser, and P. Shaw, "Dynamic VRPs: A study of scenarios," Technical Report APES-06-1998, University of Strathclyde, U.K.1998.

[14] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. K. Burke, "HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search," in *Evolutionary Computation in Combinatorial Optimization*, 2012, pp. 136-147.

[15] G. Ochoa, R. Qu, and E. K. Burke, "Analyzing the landscape of a graph based hyper-heuristic for timetabling problems," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation GECCO '09*, 2009, pp. 341-348.

[16] E. K. Burke, G. Kendall, and E. Soubeiga, "A Tabu-Search Hyperheuristic for Timetabling and Rostering," *Journal of Heuristics*, vol. 9, pp. 451-470, 2003.

[17] N. R. Sabar and M. Ayob, "Examination timetabling using scatter search hyper-heuristic," in *Data Mining and Optimization, 2009. DMO '09. 2nd Conference on*, 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*. vol. 2079, E. Burke and W. Erben, Eds., ed: Springer Berlin Heidelberg, 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 *Proceedings of the International Conference on Intelligent Technologies, InTech*, 2003, pp. 132-141.

[20] G. Kendall and M. Mohamad, "Channel assignment optimisation using a hyper-heuristic," in *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, 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*, ed: 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," *European Journal of Operational Research*, vol. 179, pp. 759-774, 2007.

[23] E. Ozcan, Y. Bykov, M. Birben, and E. K. Burke, "Examination timetabling using late acceptance hyper-heuristics," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, 2009, pp. 997-1004.

[24] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," in *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, 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*, ed: 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 *Proceedings of the Evolution artificielle, 8th international conference on Artificial evolution*, Tours, France, 2008, pp. 37-49.

[27] N. R. Sabar, M. Ayob, G. Kendall, and Q. Rong, "Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems," *Evolutionary Computation, IEEE Transactions on*, vol. 17, pp. 840-861, 2013.

[28] E. K. 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, pp. 942-958, 2010.

[29] J. Arabas, Z. Michalewicz, and J. Mulawka, "GAVaPS-a genetic algorithm with varying population size," in *Proceedings of the 1st IEEE Conference on Evolutionary Computation, 1994*, 1994, pp. 73-78 vol. 1.

[30] V. Nannen and A. Eiben, "Efficient relevance estimation and value calibration of evolutionary algorithm parameters," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 103-110.

[31] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, pp. 19-31, 3// 2011.

[32] 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*. vol. 7491, C. C. Coello, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 306-315.

[33] 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," *Journal of Scheduling*, vol. 12, pp. 55-89, 2009.

[34] M. Ayob, A. Malik, S. Abdullah, A. Hamdan, G. Kendall, and R. Qu, "Solving a practical examination timetabling problem: a case study," *Computational Science and Its Applications–ICCSA 2007*, pp. 611-624, 2007.

[35] P. Toth and D. Vigo, *The vehicle routing problem* vol. 9: Society for Industrial Mathematics, 2002.

[36] R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati, "Ant colony system for a dynamic vehicle routing problem," *Journal of Combinatorial Optimization*, vol. 10, pp. 327-343, 2005.

[37] P. Garrido and M. C. Riff, "DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *Journal of Heuristics*, vol. 16, pp. 795-834, 2010.

[38] F. T. Hanshar and B. M. Ombuki-Berman, "Dynamic vehicle routing using genetic algorithms," *Applied Intelligence*, vol. 27, pp. 89-99, 2007.

[39] 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," *Information Sciences*, vol. 180, pp. 2044-2064, 2010.

[40] T. Müller, "ITC2007 solver description: a hybrid approach," *Annals of Operations Research*, vol. 172, pp. 429-446, 2009.

[41] C. Gogos, P. Alefragis, and E. Housos, "A multi-staged algorithmic process for the solution of the examination timetabling problem," *Practice and Theory of Automated Timetabling (PATAT 2008), Montreal*, pp. 19-22, 2008.

[42] M. Atsuta, K. Nonobe, and T. Ibaraki, "ITC2007 Track 2, an approach using general csp solver," *Practice and Theory of Automated Timetabling (PATAT 2008)*, pp. 19–22, 2008.

[43] G. De Smet, "Itc2007-examination track," *Practice and Theory of Automated Timetabling (PATAT 2008)*, pp. 19-22, 2008.

[44] A. Pillay, "Developmental Approach to the Examination timetabling Problem," *Practice and Theory of Automated Timetabling (PATAT 2008)*, pp. 19–22, 2008.

[45] E. K. Burke, R. Qu, and A. Soghier, "Adaptive selection of heuristics for improving constructed exam timetables," in *Practice and Theory of Automated Timetabling (PATAT 2010)*, 2010, pp. 136-151.

[46] N. Pillay, "Evolving Hyper-Heuristics for a Highly Constrained Examination Timetabling Problem," in *Practice and Theory of Automated Timetabling (PATAT 2010)*, 2010, pp. 336-346.

[47] N. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, pp. 1-11, 2012.

[48] C. Gogos, P. Alefragis, and E. Housos, "An improved multi-staged algorithmic process for the solution of the examination timetabling problem," *Annals of Operations Research*, pp. 1-19, 2010.

[49] B. McCollum, P. McMullan, A. Parkes, E. K. Burke, and S. Abdullah, "An Extended Great Deluge Approach to the Examination Timetabling Problem," in *4th Multidisciplinary International Scheduling Conference: Theory and Applications, MISTA 2009*, 2009, pp. 424-434.