# The Automatic Design of Hyper-heuristic Framework with Gene Expression Programming for Combinatorial Optimization problems

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

*Abstract*—**Hyper-heuristic approaches aim to automate heuristic design in order to solve multiple problems instead of designing tailor-made methodologies for individual problems. Hyper-heuristics accomplish this through a high level heuristic (heuristic selection mechanism and an acceptance criterion). This automates heuristic selection, deciding whether to accept or reject the returned solution. The fact that different problems or even instances, have different landscape structures and complexity, the design of efficient high level heuristics can have a dramatic impact on hyper-heuristic performance. In this work, instead of using human knowledge to design the high level heuristic, we propose a gene expression programming algorithm to automatically generate, during the instance solving process, the high level heuristic of the hyper-heuristic framework. The generated heuristic takes information (such as the quality of the generated solution and the improvement made) from the current problem state as input and decides which low level heuristic should be selected and the acceptance or rejection of the resultant solution. The benefit of this framework is the ability to generate, for each instance, different high level heuristics during the problem solving process. Furthermore, in order to maintain solution diversity, we utilize a memory mechanism which contains a population of both high quality and diverse solutions that is updated during the problem solving process. The generality of the proposed hyper-heuristic is validated against six well known combinatorial optimization problem, with very different landscapes, provided by the HyFlex software. Empirical results comparing the proposed hyper-heuristic with state of the art hyper-heuristics, conclude that the proposed hyper-heuristic generalizes well across all domains and achieves competitive, if not superior, results for several instances on all domains.**

*Index Terms*— **Hyper-heuristics, Gene Expression Programming, 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:naserdolayme@yahoo.com, masri@ftsm.ukm.my.
Graham Kendall and Rong Qu are with the ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK.email:gxk@cs.nott.ac.uk, rxq@cs.nott.ac.uk. Graham Kendall is also affiliated with the University of Nottingham Malaysia Campus, 43500 Semenyih Selangor, Malaysia. Email: Graham.Kendall@nottingham.edu.my.

## I. INTRODUCTION

The growth in the complexity and constraints of optimization problems that can be found in many real world applications makes them not only an ongoing challenge but also implies that they cannot be solved using exact methods within tractable (or acceptable) computational time [1], [2]. Alternatively, meta-heuristic approaches, which offer no guarantee of returning an optimal solution (or even near optimal solutions), becomes not only a suitable option but also the only available option, as they usually return reasonably good solutions within a reasonable time. Although the efficiency of meta-heuristic approaches has been demonstrated over several real world applications, their success is due to the use of domain-specific knowledge [3], [4], [5]. As a consequence, to solve a given problem by a meta-heuristic algorithm, practitioners usually have to face the problem of configuring the selected meta-heuristic such as selecting the appropriate problem specific structures, most suitable operators and fine tuning the parameters, which are non-trivial tasks [6], [7] .

Over the years, it has become evident that the decision of which problem specific structures, operators and parameter values to be included (or excluded) in a given meta-heuristic algorithm has an impact on algorithm performance [3], [8], [9], [10]. Thus, to obtain a good quality solution, meta-heuristic approaches have to be expertly crafted by incorporating problem-specific knowledge of the underlying problem instance [3], [11]. Customization of a meta-heuristic can be problem or even instance dependent and consequently will decrease its generality. Moreover, according to the No Free Lunch Theorem [12] no single algorithm with a unique configuration is able to perform well over all problem instances. As a consequence, when new problems are considered, meta-heuristics need to be (re)developed, which is usually not only time consuming but also requires a deep understanding of both algorithm behavior and the instance structure. Broadly speaking, at the expense of generality, researchers and practitioners have concentrated their effort on outperforming existing methods on one, or a few instances, by tailoring a given algorithm to the problem at hand.

Arguably, meta-heuristic configuration plays a crucial role on the algorithm performance [5], [6]. Furthermore, different problems require different configurations, and even for different instances of the same problem using a different configuration during the solving process could improve algorithm performance [7]. When a search becomes trapped in a local optima, adapting the algorithm, on the fly, could help the algorithm to escape. Therefore, one way to design an effective search methodology is to take advantage of several operators as well as different parameter values by combining them in one framework or adjusting them during the solving process [13]. Automated heuristic design has proven to be an efficient and effective way in enhancing the search methodology by adjusting algorithm operators or parameter values in on-line fashion [7], [13]. These methodologies should work well, not only across different instances of the same problem, but also across several problem domains. Hyper-heuristics [3], parameter tuning [7], reactive search [14], adaptive memetic algorithms [9] and multi-method [15] are some of examples of automated heuristic design. Recently proposed frameworks in the automatic heuristic design concern the self-adaptation of search methodologies by coadapting algorithm configuration through coevolutionary process such as coadapted memeplexes [16], a theoretic model of symbiotic evolution [17] and the coevolving memetic algorithms [18].

This work focuses on the hyper-heuristic framework. Hyper-heuristics are search methodologies that explore the search space of a given set of heuristics, or heuristic components in order to select the most appropriate heuristic. Hyper-heuristics can also be utilized to evolve new heuristic by combining basic component of existing heuristics. These features distinguish hyper-heuristics from meta-heuristic methods, as they operate directly on the solution space. The key motivation behind hyper-heuristics is to raise the level of generality and to combine the strength of several heuristics or heuristic components into one framework [3].

A traditional hyper-heuristic framework has two levels. The higher level heuristic manages which low level heuristic to call (heuristic selection mechanism) and then decides whether to accept the resultant solution (the acceptance criterion). The lower level contains a set of problem specific heuristics which are different for each problem domain. Since each instance has certain characteristics and landscape complexity, high level heuristic components have a dramatic impact on the hyper-heuristic performance and that is why there is considerable research interest in devolving either new heuristic selection mechanisms or different acceptance criteria [3], [4]. The design of a good high level heuristic would increase the ability of the hyper-heuristic in selecting the correct heuristic at any particular point, and a good acceptance criterion can guide the search process toward promising regions [19], [20].

Although the high level heuristic of a *heuristic to choose heuristic* hyper-heuristic framework, has been properly designed, one can argue that most of them have one (or a few) sensitive parameters and they have been manually designed by human experts [19]. In addition, a manually designed high level heuristic needs considerable expertise and experience, and they only represent a small fraction of the overall search space. Furthermore, as far as we are aware, previous hyper-heuristic frameworks that have been proposed in the scientific literature [4], [19] are single solution based method. Reliance on a single solution may restrict their ability in dealing with huge and heavily constrained search spaces [10].

Therefore, we address the challenges of designing the high level heuristic components and of using a population of solutions in a hyper-heuristic framework by proposing the following (see Fig. 1):

i) Instead of manually designing the high level heuristic of a perturbative heuristic to *choose* heuristics in a hyper-heuristic framework, we propose an automatic programming generation framework to automatically design the heuristic selection mechanism and the acceptance criteria by using gene expression programming [21] (denoted as GEP-HH). The proposed gene expression programming framework, see Fig. 1, is implemented as an on-line heuristic or rule generation method, which evolves a population of individuals. Each individual represents a set of rules that is decoded into a selection mechanism and acceptance criteria to be used by the hyper-heuristic framework. The quality of the generated rule is evaluated by inserting it into the hyper-heuristic framework and using it on a given problem instance for a certain number of iterations. We use the idea of controlling the population size in an evolutionary algorithm to measure the performance of the generated heuristic [22]. We utilize gene expression programming algorithm to automate the design of the high level heuristic of the hyper-heuristic framework instead of genetic programming, due to its ability in avoiding code bloat and the fact that it generates a solution that is syntactically correct [21].

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

ii) We utilize a memory mechanism, which contain a set of both high quality and diverse solutions, see Fig. 1, which is updated as the search progresses in order to enhance the ability of the perturbative heuristic to *choose* heuristics when dealing with heavily constrained problems in a huge search space, and also to diversify the search.

To our knowledge, the high level heuristic components of the currently existing hyper-heuristic frameworks are all manually designed and they are also single based solution methods. Hence, the proposed framework represents a paradigm shift in using an automatic program generation method in automating the design of hyper-heuristics or meta-heuristic components, as well as using a population of solutions instead of a single solution within the set of low level heuristics. This could reduce the human expertise required in manually customizing the high level heuristic of the hyper-heuristic framework and could also enhance the performance of the hyper-heuristic framework. Our research questions are:

*"Can we use a gene expression programming algorithm framework to generate high level heuristic components (heuristic selection mechanism and the acceptance criteria) of the hyper-heuristic framework? Does the use of a population of solutions, instead of a single solution, within the hyper-heuristic framework enhance the performance of the hyper-heuristics?* "

Thus, our objectives are:

- To propose an on-line gene expression programming (GEP-HH) framework to automatically generate the high level heuristic components (heuristic selection mechanism and the acceptance criteria) of the hyper-heuristic framework.

- To propose a population based hyper-heuristic framework by incorporating a memory mechanism which contains a set of solutions updated during problem solving progress in order to effectively diversify the search.

- To test the generality and the performance of the proposed hyper-heuristic framework over six different problem domains, of very different natures and compare the results with the state of the art hyper-heuristics.

We demonstrate the generality and the consistency of the proposed hyper-heuristic framework using the HyFlex (Hyper-heuristics Flexible Framework) software [23], which provides access to six problem domains with very different landscape structures and complexity. The domains are: boolean satisfiability (MAX-SAT), one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing. This work is among the first attempts to apply a hyper-heuristic framework to tackle all these challenging problems. Although it is entirely appropriate to have a bespoke method that can produce the best known results for one (perhaps more) instance, having a methodology which is generally applicable to more than one problems domain would be more beneficial. Our ultimate goal is not to propose a hyper-heuristic framework that can outperform the best known methods but rather propose a methodology that generalizes well over different problem domains. However, the results demonstrate that the proposed hyper-heuristic is able to update the best known results for some instances.

## II. The Motivation Behind Automated Heuristic Designing

As we have mentioned earlier, given an optimization problem and a solution method, researchers or practitioners have to address the problem of which problem specific structures, operators and parameter values to be used within the given solution method in order to achieve good quality results. Although algorithm configuration is intuitively appealing, usually it is very difficult, if not impossible, to manually search through all possible configurations such as adding or removing specific operators or adjusting the parameter values [24]. Therefore, exploring such an interactive and large search space using other search methods (i.e. GEP, GP or other meta-heuristic algorithms) might yield a better performance compared to manually designing an algorithm [6] and this is actually what the automated heuristic design usually does.

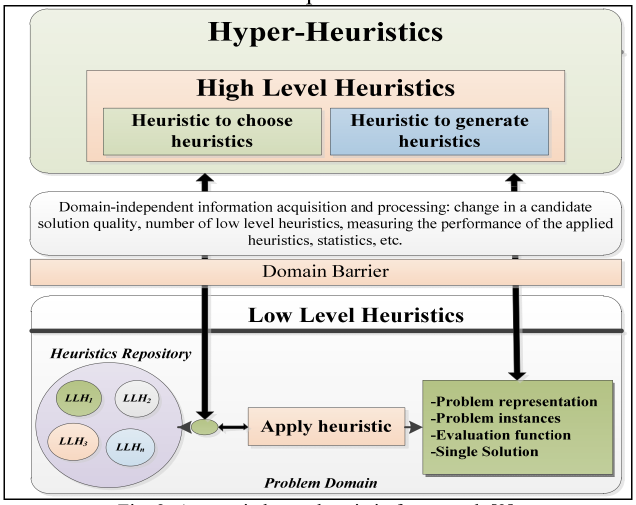
Recently, automatic program generation methods, such as genetic programming (GP), have paved the way for a paradigm of optimizing or evolving the components of search methodologies. For example, GP has been employed in [25] to evolve the cooling schedule in simulated annealing to solve quadratic assignment problems. Whilst, in [26] GP has been utilized to generate constructive heuristics for the hyper-heuristic framework. It is also used in [27] to evolve the equation that controls the movement of particles in particle optimization algorithms. In [28] GP has been used to evolve the pheromone updating strategy for an ant colony algorithm. Recently a grammatical evolution (GE) algorithm has been utilized in [29] to evolve low level heuristics for the bin packing problem. Whilst, GE is used in [30] to automatically combine the high level heuristic components of the hyper-heuristic framework. Please note that the main difference between the proposed gene expression framework and the framework introduced in [30] is that the framework proposed in this paper generates a set of rules to select the most suitable low level heuristic and then either accepts or rejects the generated solution, whilst the framework in [30] combines existing meta-heuristic acceptance criteria with neighborhood structures. Furthermore, the utilized terminal and function sets are fundamentally different.

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. That is, the function and terminal sets that have been defined for one domain cannot be used on other domains. In this work we propose an automatic program generation framework to automatically generate the high level heuristic of the hyper-heuristic framework. The novelty of our proposed framework is that it 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 also facilitate algorithm configurations.

Thus, if the automatic program generation methods can optimize meta-heuristic components [25], [28] and evolve the constructive heuristic of the hyper-heuristic framework [26], then using the automatic program generation method (GEP in this work) to automatically design the high level heuristic of the hyper-heuristic framework in an on-line manner may produce an effective hyper-heuristic framework.

## III. RELATED WORK

Hyper-heuristics are one of the automated heuristic design methodologies motivated by the fact that different heuristics impose different strength and weakness. Thus it makes sense to merge them into one framework. A recent definition of a hyper-heuristics framework is "*an automated methodology for selecting or generating heuristics to solve hard computational search problems*" [3]. Over the years, hyper-heuristic frameworks have demonstrated success in solving various classes of real world applications. A generic hyper-heuristic framework is composed of two levels known as *high* level and *low* level heuristics [3] (see Fig. 2). The *high* level heuristic is problem independent and has no domain knowledge. Its role is to manage the selection or generation of which heuristic are to be applied at each decision point. The *low* level heuristic corresponds to a pool of heuristics or heuristic components.


Hyper-Heuristics
High Level Heuristics
Heuristic to choose heuristics
Heuristic to generate heuristics
Domain-independent information acquisition and processing: change in a candidate solution quality, number of low level heuristics, measuring the performance of the applied heuristics, statistics, etc.
Domain Barrier
Low Level Heuristics
Heuristics Repository
LLH1
LLH2
LLH3
LLHn
Apply heuristic
-Problem representation
-Problem instances
-Evaluation function
-Single Solution
Problem Domain
Fig. 2. A generic hyper-heuristic framework [3]

Recently, hyper-heuristic frameworks have been classified [3] based on the nature of the heuristic search space and the source of feedback during learning (see Fig. 3). The source of feedback can be either on-line, if the hyper-heuristic framework uses the feedback obtained during the problem solving in decision making, or off-line, if the hyper-heuristic framework uses information gathered during the training phase in order to be used when solving other or 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. In either case, this is often further classified based on the employed low level heuristics into: constructive heuristics, which starts from scratch and keeps extending a partial solution step by step until a complete solution is generated, or perturbative heuristics, which starts with a complete solution and iteratively refines it to improve its quality.

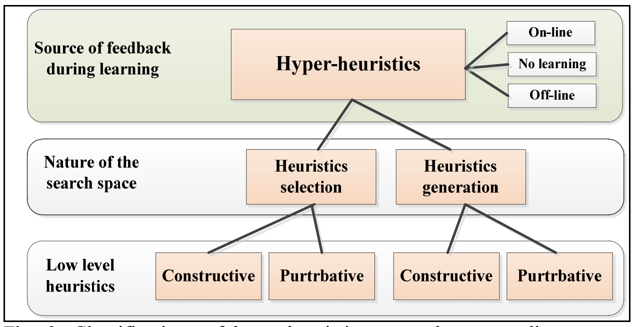


Fig. 3. Classifications of hyper-heuristic approaches, according to two dimensions: (i) the nature of the heuristic search space and (ii) the source of feedback during learning [3].

### A. *Heuristics to choose heuristics*

Most of hyper-heuristic frameworks published are heuristics to *choose* heuristics. These operate on a set of human designed heuristics called low level heuristics [19]. The set of low level heuristics can be either constructive or perturbative. The role of the hyper-heuristic framework is to intelligently select, from a given set of low level heuristics, which heuristic to apply at a given time. The motivation behind heuristics to choose heuristics is that the strength of several heuristics can be included in one framework. A traditionally perturbative heuristic based hyper-heuristic framework has two components, known as the heuristic selection mechanism and the acceptance criteria. The role of the selection mechanism is to select the low level heuristic from the given set, whilst, the acceptance criteria is to decide whether to accept or reject the resultant solution after applying the selected low level heuristic. Both components play an important role and have significant impact on hyper-heuristic performance [19], [20]. Examples of heuristic selection mechanisms are tabu search [31], genetic algorithm [32], iterated local search and variable neighborhood [33]. Examples of acceptance criteria that have been used within hyper-heuristics are simulated annealing, great deluge and tabu search [19]. More details of these hyper-heuristics can be found in recent surveys [19], [4].

The cross-domain heuristic search (CHeSC) competition has been recently 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 [23]. The algorithm designer only needs to provide the higher level component (heuristic selection and acceptance criterion). The adaptive hyper-heuristic (AdapHH) proposed in [34] was the competition winner. Their heuristic selection mechanism uses an adaptive dynamic heuristic set or relay hybridization and an adaptive acceptance criterion. Further details about the competition, including further results, are available in [23].

Recently, Chen [35] introduced an algorithm development environment (ADEP) to address meta-heuristic design and configuration problems through an integrated framework that allows both manual and automated configuration of a variety meta-heuristic approaches. The main difference between [35] and proposed GEP-HH framework is that the proposed GEP-HH framework generates meta-heuristic components instead of combining and/or configuring existing ones.

Although several types of heuristic selection mechanisms and acceptance criteria exist, no heuristic selection mechanisms or acceptance criteria so far presented are the best, or the most suitable, across all domains [19]. In practice, all of them face generalization issues. This is because the choice of which heuristic to apply does not depend only on the problem instances but also on the current stage of the solving process, since at each decision point the problem instance landscape is acquiescent to at least one low level heuristic. Most of the current heuristic selection mechanisms use simple rules to select the low level heuristic based on their past performance [19]. However, to quickly respond to instance landscape changes, a sophisticated heuristic selection mechanism may be needed. Furthermore, some low level heuristics perform well only at the beginning of the search process while others could be good at the end of solving process [19], [13]. For example, the application of a certain local search based low level heuristic would be unuseful if the solution is already trapped in a local optima. As a result, there is a need for a high level heuristic that is more general than those currently available, that can use the problem state in selecting the appropriate low level heuristic, and can cope with several problem domains or even different instances of the same problem.

In this work, we address this challenge by proposing a gene expression programming framework to generate, for each instance, the heuristic selection mechanism and the acceptance criteria for the perturbative heuristic to *choose* heuristic. What makes our proposed framework different from others is that, at every iteration, the generated selection mechanism and acceptance criteria favor different criteria or information in selecting the low level heuristic and the acceptance of the generated solution. For example, the heuristic selection mechanism generated at iteration *i* may favor the selection of the low level heuristic that has very good performance during the previous application, whilst, the heuristic selection mechanism generated at iteration *i+1* may favor the selection of the low level heuristic that has been more frequently applied than those of very good performance.

### B. Heuristics to generate heuristics

In contrast to the heuristics to *choose* heuristics hyper-heuristic, where the hyper-heuristic starts with a set of low level heuristics provided manually, in a heuristics to *generate* heuristics hyper-heuristic the aim is to fabricate new low level heuristics by combining existing heuristic components [3]. Genetic programming has been successfully used to evolve constructive heuristics for SAT [36], scheduling [37] and bin packing problems [26].

Despite the fact that genetic programming hyper-heuristics have achieved good results, one can argue that most of them are tailored to solve specific problems (e.g. SAT and the bin packing problems) using a restricted constructive heuristic component. Another limitation is that they have been used in an off-line manner which may restrict their generality because they will be tailor made to the training instances unless the testing instances have the same features and complexity which usually does not reflect many real world applications.

Motivated by the achievements of the above work, in this work, we propose a gene expression programming framework to automatically generate the high level heuristic for the perturbative heuristics to *choose* heuristics hyper-heuristic framework. The proposed gene expression framework can be classified as an on-line generational hyper-heuristic and thus the same as a genetic programming hyper-heuristic. The benefit of the proposed gene expression programming framework is its ability to use the current problem state to generate, for each instance, different high level heuristic in an on-line manner which could help the search in coping with the changes that might happen during the instance solving process.

## IV. The Proposed Framework

The proposed hyper-heuristic framework has two levels called *high* level and *low* level heuristics. The *high* level heuristic contains two components, a heuristic selection mechanism and an acceptance criterion. The low level heuristic contains a set of perturbative low level heuristics, the memory mechanism and the objective function. The proposed hyper-heuristic starts with an initial solution, randomly selected from the memory mechanism, and iteratively explores its neighborhood by applying a perturbative low level heuristic. Given a pool of perturbative low level heuristics, a complete solution (randomly selected from the memory mechanism) and the objective function, the proposed hyper-heuristic framework will successively invoke the following steps for a certain number of iterations (defined by the user):

i) Call the heuristic selection mechanism to select, from a given pool, one perturbative low level heuristic.
ii) Randomly selects one solution for the memory mechanism.
iii) Apply the selected perturbative low level heuristic to the given solution to generate a new solution.
iv) Call the objective function to evaluate the generated solution. If it is better than the incumbent solution, replace it with the incumbent solution and continue the search. If not, call the acceptance criterion to decide either to accept or reject the generated solution according to the acceptance criterion rules.

v) Update the memory mechanism, the parameters and start a new iteration.

## A. High level heuristic

In the high level heuristic, the role of the heuristic selection mechanism is to select, for a given instance, the suitable perturbative low level heuristic from those supplied. Usually, the choice of which perturbative low level heuristic to be applied is a crucial decision, as this would lead the search in order not to confine it to a localized region of the solution space. The aim of the acceptance criterion is to assist the search process in order to avoid local optima and at the same time explore different areas of the search space through the decision of accepting or rejecting the generated solution [4]. A good acceptance criteria mechanism should be able to strike a balance between accepting improving solutions and also worse solutions if the search is trapped in a local optima [20]. Therefore, this work proposes a program generation method using gene expression programming to adaptively select the suitable low level heuristic and to balance between accepting and rejecting the generated solution (see Fig. 4).

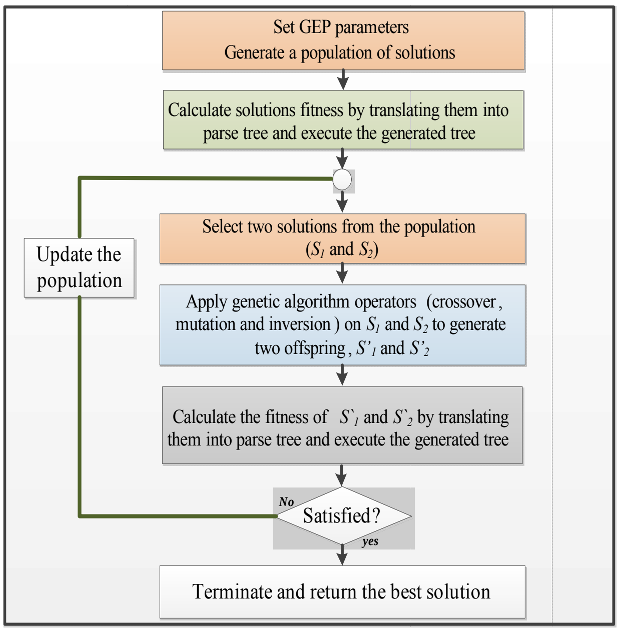
Fig. 4. The proposed high level heuristic

### 1) Basic gene expression programming algorithm

Gene expression programming (GEP) [21] is a program generation method that uses a linear representation instead of a tree representation that is often used in genetic programming (GP). Each individual in GEP comprises a set of strings with a fixed size, called genomes. The program in GEP is generated by converting the individual string into a parse tree utilizing breadth-first search. The parse tree is then executed against the given problem instance. To generate a new individual, GEP applies genetic algorithm operators (crossover, mutation, inversion and transposition) directly on the linear encoding instead of the parse tree. Thus, GEP merges the advantages of both a genetic algorithm and genetic programming in evolving a population of computer programs. This feature allows GEP to generate programs that are always syntactically correct while avoiding the problem of code bloat (a well-known problem in traditional GP). The evolutionary steps of GEP in generating population of individuals are shown in Fig 5.

First, GEP components are defined. These are the function set ($F$) (which manipulates the values returned by terminals, and they take one or more arguments), terminal set ($T$) (which represents a set of nodes that form the leaf nodes of the program tree; they take no arguments), fitness function, GEP parameters and stopping condition.

Next, we generate a population of individuals. An individual in GEP is composed of a set of symbols called genes. Each gene has two elements called *head* and *tail*. Head contains both terminals and functions and its length $h$ is fixed by users. The tail only contains terminals and its length $t$ is calculated by the formula $t=h*(n-1)+1$, where $n$ represents the maximum number of function arguments [21]. Thus, the individual length is equal to $h+t$. Assume a individual is comprised of a set of symbols of function $F = \{*, /, +, -\}$ and terminal $T = \{a, b\}$. In this example, $n = 2$ because the maximum arity of the function is two arguments. If we set the head length $h = 10$, then the tail length $t = 11$ and the length of the individual will be $h + t = 10+11 = 21$. An example of a randomly generated individual can be [21]: GEP_gene=+*ab-ab+aab+ababbbabaa and its corresponding expression tree is: GEP_expression= a+b*((a+b)-a). Each individual in the population employs the head-tail encoding method which ensures the validity of the generated individual.



Set GEP parameters
Generate a population of solutions
Calculate solutions fitness by translating them into parse tree and execute the generated tree
Select two solutions from the population ($S_1$ and $S_2$)
Update the population
Apply genetic algorithm operators (crossover, mutation and inversion) on $S_1$ and $S_2$ to generate two offspring, $S'_1$ and $S'_2$
Calculate the fitness of $S`_1$ and $S`_2$ by translating them into parse tree and execute the generated tree
No
Satisfied?
yes
Terminate and return the best solution

Fig. 5. Basic gene expression programming flowchart

Then, we calculate individual fitness as follows: following the breadth-first manner individuals are converted into expression trees. First, scan the individual string one by one from left to right. The first string will form the node of the tree and other strings are written in a left to right manner at each lower level. If the scanned string is a function ($F$) with $n$ ($n$>=1) arguments, then the next $n$ strings are attached below it as its $n$ children. Otherwise, it will form a leaf of the corresponding tree (terminal ($T$)). The scanning process is repeated until all leaves in the corresponding tree are terminals only. Next, the program trees are executed on the underlying problem and their fitness values are calculated.

Next, two individuals are selected by the selection mechanism (e.g. roulette wheel selection) according to their fitness values. The selected individuals will go through the following genetic operators:

i) Crossover: exchanges elements between two randomly selected genes from the chosen parents (e.g., one-point and two point crossover).
ii) Mutation: change any string in the generated individual while making sure that the string in the head part can be changed into both terminal and function and, string in the tail part can be changed into terminals only.
iii) Inversion: reveres small sequence of strings within the head or tail.
iv) Convert the created individuals (offsprings) to program trees and execute them on the underlying problem to calculate their fitness values.
v) Following roulette wheel (or other selection operators) sampling with elitism, the fittest individuals are always copied into the next generation.

This process is executed until the stopping condition is satisfied (e.g. a given number of generations).

*2) The proposed gene expression programming framework to generate the high level heuristic components*

In this work, we propose a gene expression programming framework to automatically generate the high level heuristic selection mechanism and the acceptance criteria, based on a given problem instance, for the perturbative heuristic to *choose* heuristic hyper-heuristic framework. This is an on-line heuristic generation method based hyper-heuristic which iteratively evolves a population of individuals through the evolution process. Each individual represents a set of rules which are decoded into a selection mechanism and acceptance criterion to be used by the hyper-heuristic framework. To simultaneously generate both selection mechanism and the acceptance criterion, each individual is divided into two parts of equal size to represent both components. For example, in a individual of *m* strings, strings 1 to *m*/2 will be used for the selection mechanism and strings *m*/2 to *m* will be used for the acceptance criterion. Each part has a *head* of a user defined length *h* (contains terminal and function) and a *tail* (contains terminal only) of length $t=h*(n-1)+1$, where *n* represent the maximum number of function arguments. Each part employs the head-tail encoding method which ensures the validly of the generated program which represents one expression tree for the selection mechanism and acceptance criterion, respectively.

Except crossover, genetic operators (mutation and inversion) can occur at any point as long as the gene rules are respected, i.e., a head element can be changed into terminal or function, whilst, a tail element can be changed into terminal only. Crossover operators will exchange elements between two randomly selected genes from the chosen parents within the same parts. For example, if the selected genes are from the first part of the first individual, these genes will be replaced with those in the first part of the second individual. This will ensure that the exchanged genes are the same types, i.e., either for the selection mechanism or the acceptance criterion.

To run the proposed gene expression programming framework, one needs to define the following components:

1- Terminal and function sets

A crucial issue in the design of the proposed framework is the definition of the terminal set (*T*) and the function set (*F*). The terminal set (*T*) represents a set of variables which will express the state of the underlying problems. The function set (*F*) represents a set of arithmetic or logical operators that will be used to connect or compose the terminal set (*T*). To use the proposed framework across various problems, we keep the definition of the terminal set (*T*) and function set (*F*) as general and simple as possible. By doing so, the proposed framework can be used across other problem domains, in addition to those considered in this work. Since the purpose of the heuristic selection mechanism is fundamentally different from the acceptance criterion, we use two terminal sets. The first set represents the selection mechanism, whilst, the second represents the acceptance criterion.

To cope with instance changes that might happen during the instance solving process, the proposed framework utilizes several evaluation criteria to represent the terminal sets in such a way that their combination will favor one criterion among others and these evaluation criteria will be updated during instance solving. Each evaluation criterion favors the selection of the low level heuristic from a different perspective. The rationale behind this is that some low level heuristics perform well only at the beginning of the search process while others could be better at the end of the process. Therefore, the heuristic selection mechanism should be able to quickly respond to instance landscape changes by selecting the appropriate low level heuristic. The function (*F*) and terminal (*T*) sets of the selection mechanism that have been used in this work are presented in Table 1. The utilized terminals for the heuristic selection are:

- Reward credit (*RC*): The main idea of this reward is that infrequently used low level heuristics which lead to a large improvement in the solution quality are preferred to be selected more than those that lead to a small improvement. Thus, as a result, the low level heuristic which brings frequent, but small improvements will get less reward and consequently has a lesser chance of being preferred [13]. This terminal is good in reducing the heuristic search space by only favoring certain low level heuristics.
- Update the best known solution counter ($C_{best}$): This terminal favors the low level heuristic that manage to update the best known results. This terminal is good in systematically improving the current local optima.
- Update the counter of accepting current solution ($C_{current}$): This terminal favors the low level heuristic that manages to update the current solution. This terminal is good in keeping the search focused around the current local solution.
- Update counter of accept solution ($C_{accept}$): This terminal favors the low level heuristic that produces a solution that is accepted by the acceptance

criterion. This terminal is good in helping the search to escape from a local optima.
- Update the average improvement counter ($C_{ava}$): This terminal favors the low level heuristic that has made a large improvement on average. This terminal is good at focusing the search on the current area in the search space.
- Update the first rank counter ($C_r$): This terminal favors the low level heuristic that has been selected first. This terminal is good for applying the current low level heuristic.

Please note that the terminal (*T*) set of the heuristic selection mechanism is used for the low level heuristic and their value together with function (*F*) set are used to rank the low level heuristics.

TABLE 1 THE TERMINAL AND FUNCTION SET OF THE SELECTION MECHANISM

| **Terminals set for the heuristics selection mechanism** | |
|---|---|
| ***terminal*** | ***description*** |
| *RC* | *The extreme value-based reward is used to calculate the credit (CA) for each low level heuristic. When the i-th low level heuristic is applied, its corresponding improvement to the current solution is computed. The improvement gained is then saved for the i-th low level heuristic in a sliding time window of size W, following the rule of FIFO. The credit of any low level heuristic is then set as the maximum value in its corresponding sliding window W. In this work, the improvement gained (PI) from the i-th low level heuristic is calculated as follows: PI(i) =(\|f1-f2/f1)*100 if f2<f1. Where f1 is the quality of the current solution and f2 is the quality of the resultant solution after applying the i-th low level heuristic.* |
| $C_{best}$ | *The number of times that the i-th low level heuristic has updated the best known solution.* |
| $C_{current}$ | *The number of times that the i-th low level heuristic has updated the current solution.* |
| $C_{accept}$ | *The number of times that the generated solution by the i-th low level heuristic has been accepted by the acceptance criterion.* |
| $C_{ava}$ | *The average of the previous improvement strength of the i-th low level over the search process.* |
| $C_r$ | *The number of times that the i-th low level heuristic has been ranked the first.* |

| **Function set for the heuristics selection mechanism** | |
|---|---|
| ***function*** | ***description*** |
| + | *Add two inputs.* |
| - | *Subtract the second input from the first one.* |
| * | *Multiply two inputs.* |
| % | *Protected divide function, i.e., change the division by zero into 0.001.* |

The function (*F*) and terminal (*T*) sets of the acceptance criteria that have been used in this work are presented in Table 2.

TABLE 2 THE TERMINAL AND FUNCTION SET OF THE ACCEPTANCE CRITERIA

| **Terminals set for the acceptance criteria mechanism** | |
|---|---|
| ***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 for the acceptance criteria mechanism** | |
|---|---|
| ***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.* |

2- Fitness function

The aim of the fitness function is to evaluate the performance of the generated high level heuristics (population individual). In this work, we use the idea in [22] that was used to control the population size in an evolutionary algorithm to evaluate the fitness of the generated high level heuristics. The probability of selecting each high level heuristic (an individual in the GEP framework) is updated according to the quality of the best solution returned, after the stopping condition is satisfied. The quality of the returned solution is usually either better or worse than the one that has been used as an input solution for the hyper-heuristic framework. Formally, let *Ah*[] represent the array of the probability of selecting the high level heuristics (individual), $f_i$ and $f_b$ represent the fitness of the initial and returned solutions, *NoH* represents the number of high level heuristics (individuals) or the population size of GEP. Then, if the application of the *i*-th high level heuristic leads to an improvement in the solution quality, then reward the *i*-th high level heuristic (individual) as follows: $Ah[i] = Ah[i]+\Delta$ where $\Delta = (f_i - f_b) / (f_i + f_b)$. Other high level heuristics, $\forall j \in \{1,\ldots, NoH\}$ and $j \neq i$, are penalized as $Ah[j] = Ah[j] - (\Delta/(NoH-1))$. Otherwise (if the solution cannot be improved), then penalize the *i*-th high level heuristic, $Ah[i]= Ah[i]-|(\Delta*\alpha)|$ where $\alpha$= *Current_Iteration* / *Total_Iteration* and reward other high level heuristics, $\forall j \in \{1,\ldots, NoH\}$ and $j \neq i$, $Ah[j] =Ah[j] + (|\Delta|*\alpha/(NoH-1))$. Please note that the main idea behind decreasing the probability of other high level heuristic is to decrease their chances of being selected. Initially, the probability of each high level heuristic (individual) is calculated by translating them into expression trees and executing the corresponding program.

3- The stopping condition

In this work, the maximum number of consecutive non improvement iterations is used as the stopping condition (see section V.A).

When all elements are defined, the proposed framework is carried out as follows (see Fig. 6):

i) Generate a population of individuals.

ii) Calculate the fitness of the population by inserting them into the hyper-heuristic framework and using it to solve a given instance for a certain number of iterations.
iii) Iteratively selects two parents, apply crossover and mutation operators to generate two offspring, evaluate the fitness of the generated offspring and update the population. This is executed for a certain number of generations.

The main role of GEP is to evolve a population of individuals, each encoding a high level heuristic (selection mechanism and acceptance criterion) which will be used by the hyper-heuristic framework. The hyper-heuristic framework will be called at every generation to evaluate the generated offspring. When the proposed hyper-heuristic is called the following steps will be carried out:

i) Decoded the current individual into a heuristic selection mechanism and an acceptance criterion, i.e., translate it into two expression trees for the selection mechanism and the acceptance criterion, respectively. Then, use the terminal (*T*) set value of each low level heuristic as the input for the selection mechanism expression tree.
ii) Execute the selection mechanism expression tree and rank the given set of low level heuristics from the highest to the lowest based on the value retuned from the expression tree.
iii) Randomly select one solution for the memory mechanism. Apply the highest ranked low level heuristic to the given solution and calculate the quality of the generated solution.
iv) If the generated solution is better than the current one, the current one is replaced. If not, the hyper-heuristic will call the acceptance criterion expression tree and execute the corresponding program. Then, the generated solution by the low level heuristic is accepted if the exponential of the value retuned by the acceptance criterion expression tree is less or equal to 0.5 (the *exp* function returns values between 0 and 1). In the literature, a value of 0.5 was suggested [26], but for different domains. The value 0.5 was also determined based on preliminary testing.
v) Repeatedly apply the *current* low level heuristic until no improvement is returned.
vi) If no improvement is returned, the hyper-heuristic framework will stop applying the current low level heuristic and restarts from the local optimum obtained by current low level heuristic, but with next low level heuristic in the ranked list.
vii) If the hyper-heuristic framework reaches the end of the low level heuristic ranked list, it executes the current heuristic selection mechanism expression tree again and rank the given set of low level heuristics and restart the search from the local optimum, but using the current highest ranked low level heuristic.
viii) The proposed hyper-heuristic framework will keep using the utilized high level heuristic components (selection mechanism and acceptance criterion), which is generated by the GEP framework, for a pre-defined number of iterations (see section V. A).

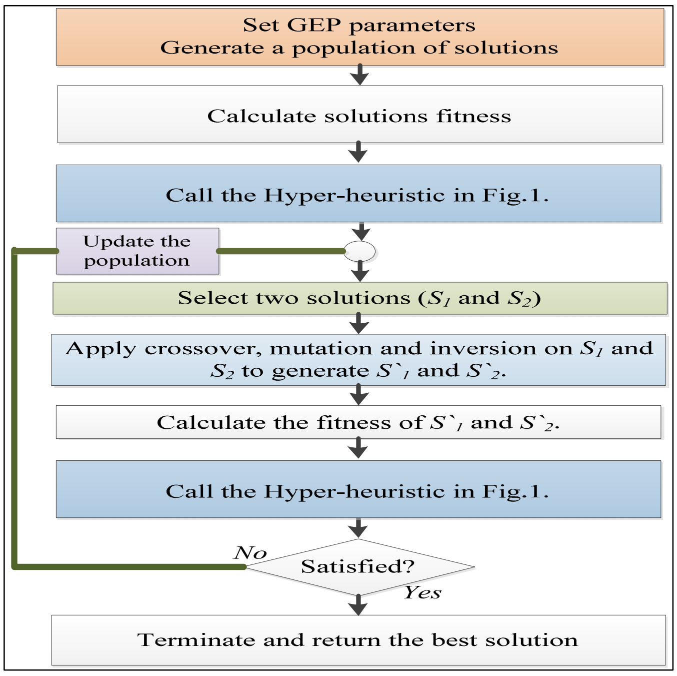

Set GEP parameters
Generate a population of solutions
Calculate solutions fitness
Call the Hyper-heuristic in Fig.1.
Update the population
Select two solutions ($S_1$ and $S_2$)
Apply crossover, mutation and inversion on $S_1$ and $S_2$ to generate $S`_1$ and $S`_2$.
Calculate the fitness of $S`_1$ and $S`_2$.
Call the Hyper-heuristic in Fig.1.
No
Satisfied?
Yes
Terminate and return the best solution

Fig. 6. The proposed hyper-heuristic

### *B. Low level heuristics*

The low level heuristic of the proposed hyper-heuristic framework has three components as follows:

#### *1) A set of perturbative low level heuristics*

In this work, a pool of problem-specific perturbative heuristics is used as low level heuristics. The aim of the low level heuristics is to explore the neighborhoods of the current solution by altering the current solution (perturbation). The generated neighborhood solution is accepted if it does not break the imposed hard constraints and also satisfies the acceptance criterion. Thus, the employed low level heuristic explores only the feasible search space. Details of these perturbative heuristics are presented in the problem description sections (see section V.C).

#### *2) Memory mechanism*

Most hyper-heuristic frameworks that have been proposed in the scientific literature operate on a single solution [4], [19]. Reliance on a single solution may restrict their ability in dealing with a large and heavily constrained search space, as it is widely known that single solution based methods are not well suited to cope with the large search spaces and heavily constrained problems [10]. In order to enhance the efficiency of the proposed hyper-heuristic framework and to diversify the search, we embed it with a memory mechanism as in [38] which contains a collection of both high quality and diverse solutions, updated as the algorithm progresses. The integrated memory mechanism interacts with the high level heuristic as follows: first initialize the memory mechanism by generating a set of diverse solutions (randomly or by using a heuristic method, see Section V). For each solution, associate a frequency matrix to measure solution diversity. The frequency matrix stores the frequency of an object assigned to the same location. At every iteration, the high level heuristic will randomly select one solution from the memory; apply the selected low level heuristic to this solution, update both the solution in memory and the solution frequency matrix.

The associated frequency matrix is represented by a two dimensional array where rows represent objects and columns represent locations. For example, in the bin packing problem, the frequency matrix stores how many times the item has been assigned to the same bin. Whilst, in the vehicle routing problem, it stores how many times a customer has been assigned to the same route. In this work, objects represent the items in the bin packing problem or customers in the vehicle routing problem, while locations represent bins in the bin packing problem and routes in the vehicle routing problems.

Fig. 7 shows an example of a solution and its corresponding frequency matrix. The frequency matrix is initialized to zero. We can see five objects (represented by rows, items or customers) and there are five available locations (represented by columns, bins or routes). The solution on the left side of Fig. 7 can be read as follows: object 1 is assigned to location 1, object 2 is assigned to location 3, etc. The frequency matrix on the right side of Fig. 7 can be read as follows: object 1 has been assigned to location 1 twice, to location 2 three times, to location 3 once, to location 4 four times and to location 5 once; and so on for the other objects.

| | *Location* | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *1* | *2* | *3* | *4* | *5* | | *1* | *2* | *3* | *4* | *5* |
| *Objects* *1* | *1* | *0* | *0* | *0* | *0* | *Objects* *1* | *2* | *3* | *1* | *4* | *1* |
| *2* | *0* | *0* | *1* | *0* | *0* | *2* | *1* | *1* | *1* | *2* | *2* |
| *3* | *0* | *0* | *0* | *0* | *1* | *3* | *2* | *2* | *2* | *2* | *1* |
| *4* | *0* | *0* | *0* | *1* | *0* | *4* | *2* | *1* | *3* | *1* | *1* |
| *5* | *0* | *1* | *0* | *0* | *0* | *5* | *2* | *1* | *2* | *1* | *3* |
| | *solution* | | | | | | *frequency matrix* | | | | |

Fig. 7. Solution and its corresponding frequency matrix.

If any solution is used by the hyper-heuristic framework, then we update the frequency matrix of this solution. Next we calculate the quality and the diversity of this solution. In this work, the quality represents the quality of the solution of a given instance (see section V). The diversity is measured using the entropy information theory (see Equations (1) and (2)) as follows [38]:

$$\varepsilon_i = \frac{\sum_{j=1}^{e} \frac{e_{ij}}{m} . \log \frac{e_{ij}}{m}}{-\log e} \qquad (1)$$

$$\varepsilon = \frac{\sum_{i=1}^{e} \varepsilon_i}{e} \qquad (2)$$

Where

- *$e_{ij}$ is the frequency of allocating object i to location j.*
- *m is the number of objects.*
- *$\varepsilon_i$ is the entropy for object i.*
- *$\varepsilon$ is the entropy for one solution ($0 \le \varepsilon_i \ge 1$).*

Next, add the new solution to the memory mechanism by considering the solution quality and diversity.

### 3) *Objective function*

The objective function is problem dependent and it measures the quality of the generated solution (see section V).

## V. Experimental Setup

In this section, we will discuss the parameter settings of GEP-HH, problem description and the perturbative low level heuristics of the considered problems.

### A. *GEP-HH Parameter Settings*

Fine tuning the algorithm parameters for optimal performance is usually a tedious task that needs considerable expertise and experience [6]. Therefore, the parameter values of the GEP-HH are obtained by using Relevance Estimation and Value Calibration method (REVAC) [39]. REVAC is a tool for parameter optimization, where a steady state genetic algorithm and entropy theory are used in defining algorithm parameter values. REVAC is utilized to find the generic values that can be used for all considered domains instead of finding the optimal one which is problem (if not instances) dependent.

Taking into consideration the solution quality and 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 [39] for more details). To do so, we tuned GEP-HH for each domain separately and then used the average of the minimum value for each parameter obtained by REVAC for all tested instances. Then the average values over all tested instances for all domains for each parameter are set as the generic values for GEP-HH. Table 3 lists the parameter settings of GEP-HH that have been used for all problem domains.

TABLE 3 GEP-HH PARAMETERS

| ***Parameters*** | ***Possible Range*** | ***Suggested Value by REVAC*** |
|---|---|---|
| Population size | 5-50 | 10 |
| Number of generations | 10-200 | 100 |
| One point crossover probability | 0.1-0.9 | 0.7 |
| Mutation probability | 0.1-0.9 | 0.1 |
| Inversion rate | 0.1-0.9 | 0.1 |
| Head length *h* | 2-40 | 5 |
| Selection mechanism | - | Roulette Wheel |
| Crossover type | Two/multi/ one point | One point |
| Consecutive non improvement | 1-1000 | 50 |
| The sliding window size *W* | 2-100 | 20 |
| Memory mechanism size | 2-40 | 8 |

## B. Problem Description

In this work, we used HyFlex (Hyper-heuristics Flexible Framework) to test the generality and the performance of GEP-HH. HyFlex is a java framework which provides six problem domains (boolean satisfiability (MAX-SAT), one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing), the initial solution generation method, and a set of perturbative low level heuristics [23]. HyFlex was used during the cross-domain heuristic search challenge competition (CHeSC) in order to compare the performance of hyper-heuristic methods and to support researchers in their efforts to develop generally applicable hyper-heuristics for various problem domains. In addition, we also report in the appendix, the results of testing GEP-HH on exam timetabling and dynamic vehicle routing problems (See the supplementary file).

### 1) Boolean Satisfiability (MAX-SAT) Problems

Boolean Satisfiability problems can be defined as follows [40]: given a formula of Boolean variables, determine the assignment of truth values to the variables that can make the formula true. MAX-SAT, which is an extension of Boolean Satisfiability, is an optimization problem where the aim is to determine the maximum number of true clauses of a given Boolean formula. In other words, the aim of the optimization process is to minimize the number of unsatisfied clauses in a given formula. The instances that were considered in this work are summarized in Table 4. The set of initial solutions are randomly generated by assigning either true or false value to each variable. The quality of the solution is measured based on how many `broken' clauses in a given formula i.e., those which evaluate to false. See [40] for more details.

TABLE 4 THE MAX-SAT INSTANCES

| *Instances* | *Name* | *Variables* | *Clauses* |
|---|---|---|---|
| Instance 1 | parity-games/instance-n3-i3-pp | 525 | 2276 |
| Instance 2 | parity-games/instance-n3-i4-pp-ci-ce | 696 | 3122 |
| Instance 3 | parity-games/instance-n3-i3-pp-ci-ce | 525 | 2336 |
| Instance 4 | jarvisalo/eq.atree.braun.8.unsat | 684 | 2300 |
| Instance 5 | highgirth/3SAT/HG-3SAT-V300-C1200-4 | 300 | 1200 |

### 2) One Dimensional Bin Packing Problems

The one dimensional bin packing is a well-known combinatorial optimization problem. Given a set of items of a fixed weight and a finite number of bins of fixed capacity, the goal is to pack all items into as few bins as possible [41]. The packing process should respect the following constraints: each item should be assigned to one bin only and the total weight of items in each bin should be less or equal to the bin capacity. The aim of the optimization process is to minimize the number of bins that are used. Table 5 shows the characteristic of the considered instances. The set of initial solutions are generated as follows: first, generate a random sequence of items and then pack them one by one into the first bin which they will fit, i.e. "first fit heuristic". The quality of solution is measured by *quality*= $1-\frac{1}{n}\sum_{i=1}^{n}\left(\frac{fl}{C}\right)^2$ where $n$ is the number of bins, $fl$ is the sum of the sizes of all the pieces in bin $i$, and $C$ the bin capacity. See [41] for more details.

TABLE 5 THE ONE DIMENSIONAL BIN PACKING INSTANCES

| *Instances* | *Name* | *Capacity* | *No. Pieces* |
|---|---|---|---|
| Instance 1 | triples2004/instance1 | 1000 | 2004 |
| Instance 2 | falkenauer/u1000-01 | 150 | 1000 |
| Instance 3 | test/testdual7/binpack0 | 100 | 5000 |
| Instance 4 | 50-90/instance1 | 150 | 2000 |
| Instance 5 | test/testdual10/binpack0 | 100 | 5000 |

### 3) Permutation Flow Shop Problems

The permutation flow shop problem is defined as, while respecting the imposed constraints, find the sequence for a set of jobs to be processed on a set of consecutive machines with the minimal completion time of the last job to exit the shop [42]. Each job requires a processing time on a particular machine. One machine can only process one job at a time. Jobs can be processed by only one machine at a time. The job ordering process should be respected and machines are not allowed to remain idle when a job is ready for processing. Table 6 shows the characteristic of the considered instances. The set of initial solutions are generated by using the NEH [42] algorithm which works as follows: first generate a random permutation of jobs and an empty schedule. Then, assign the first job in the permutation sequence into the schedule, second job into places 1 and 2; third job into places 1, 2 and 3, and so on. Each assignment should be fixed where the partial schedule has the smallest makespan time, i.e. completion time of the last job. The quality of solution represents the completion time of the last job in the schedule. See [42] for more details.

TABLE 6 THE PERMUTATION FLOW SHOP INSTANCES

| *Instances* | *Name* | *No. jobs* | *No. Machines* |
|---|---|---|---|
| Instance 1 | 100x20/2 | 100 | 20 |
| Instance 2 | 500x20/2 | 500 | 20 |
| Instance 3 | 100x20/4 | 100 | 20 |
| Instance 4 | 200x20/1 | 200 | 20 |
| Instance 5 | 500x20/3 | 500 | 20 |

### 4) Personnel Scheduling Problems

Personnel scheduling is a well-known NP-hard problem. Given a set of employees of specific categories, a set of pre-defined periods (shifts) on a working day, and a set of working days; the aim of the optimization process is to assign each employee to specific planning periods to meet the operational requirements and satisfying a range of preferences as much as possible [43]. Due to the variety of hard and soft constraints, which are different from one organization to another, the modeling and implementation is challenging. A unique general mathematical model to accommodate all related constraints does not exist. Table 7 gives the characteristics of the considered instances. The set of initial solutions are created by using a neighborhood operator which incrementally adds new shifts to the roster until all employees have been scheduled. The quality of the generated solutions is assessed based on how many soft constraints are satisfied. See [43] for more details.

TABLE 7 THE PERSONNEL SCHEDULING PROBLEMS INSTANCES

| *Instances* | *Name* | *Staff* | *Shift Types* | *Days* |
|---|---|---|---|---|
| Instance 1 | Ikegami-3Shift-DATA1.2 | 25 | 3 | 30 |
| Instance 2 | MER-A | 54 | 12 | 42 |
| Instance 3 | ERRVH-B | 51 | 8 | 42 |
| Instance 4 | BCV-A.12.1 | 12 | 5 | 31 |
| Instance 5 | ORTEC01 | 16 | 4 | 31 |

#### *5) Traveling Salesman Problems*

The traveling salesman problem is a very popular combinatorial optimization problem [44]. In its classic form, given a set of cities and their positions (pairwise distances), the aim is to find the shortest path where each city is visited only once and the path ends at the starting city. The aim of the optimization process is to minimize the traveling distance. Table 8 gives the characteristics of the considered instances. The set of initial solutions are created by randomly generating permutation sequences. The quality of solution is represented by the total distance of the route.

TABLE 8 THE TRAVELING SALESMAN INSTANCES

| *Instances* | *Name* | *No. Cities* |
|---|---|---|
| Instance 1 | pr299 | 299 |
| Instance 2 | usa13509 | 13509 |
| Instance 3 | rat575 | 575 |
| Instance 4 | u2152 | 2152 |
| Instance 5 | d1291 | 1291 |

#### *6) Vehicle Routing Problems*

The vehicle routing problem is a well-known challenging combinatorial optimization problem [45]. Given a set of customers associated with demand and serving time, and a fleet of vehicles with a maximum capacity, the aim is to design a least cost set of routes to serve all customers, where each vehicle starts and ends at the depot, the total demand of each route does not exceed the vehicle capacity, each customer is visited exactly once by exactly one vehicle during its time window(s). Table 9 shows the characteristics of the considered instances. The set of initial solutions are generated as follows: first create an empty route, then loop through all customers and add any one to the current route that does not violate any constraints. If no customer can be added to the current route, create a new route. The process is repeated until all customers have been assigned to a route. The quality of solution represents the total travel distance.

TABLE 9 THE VEHICLE ROUTING PROBLEMS INSTANCES

| *Instances* | *Name* | *No. Vehicles* | *Vehicle Capacity* |
|---|---|---|---|
| Instance 1 | Homberger/RC/RC2-10-1 | 250 | 1000 |
| Instance 2 | Solomon/RC/RC103 | 25 | 200 |
| Instance 3 | Homberger/C/C1-10-1 | 250 | 200 |
| Instance 4 | Solomon/R/R101 | 25 | 1000 |
| Instance 5 | Homberger/RC/RC1-10-5 | 250 | 200 |

### *C. The perturbative low level heuristics*

HyFlex provides, for each of the considered problems, a set of different perturbative low level heuristics. The set of perturbative low level heuristics are classified into four types as follows:

- Mutational or perturbation heuristics: generate a new solution by modifying the current solution by changing, removing, swapping, adding or deleting one solution component. Mutation intensity is controlled by $\alpha$, 0<= $\alpha$ <=1.
- Ruin-recreate (destruction-construction) heuristics: destroy part of the current solution and recreate it in a different way to generate a new solution. The difference between ruin-recreate and mutational heuristics is that the ruin-recreate can be seen as large neighborhood structures and they use problem specific construction heuristics to recreate the solutions.
- Hill-climbing or local search heuristics: iteratively perturb the current solution, only accepting improving solutions, until a local optimum is found or a stopping condition is satisfied. The difference between hill-climbing and mutational heuristics is that hill-climbing is an iterative improvement process, accepting only improving solutions. The depth of search is controlled by $\beta$, 0<= $\beta$ <=1.
- Crossover heuristics: take two solutions and produce a new one by combining them.

Table 10 shows the total number of each type of the perturbative low level heuristics for the six problem domains [23].

TABLE 10 HYFLEX LOW LEVEL HEURISTIC TYPES

| # | *Problem domains* | *M* | *R&R* | *HC* | *Xover* | *Total* |
|---|---|---|---|---|---|---|
| 1- | Boolean Satisfiability | 4 | 1 | 2 | 2 | 9 |
| 2- | One Dimensional Bin Packing | 3 | 2 | 2 | 1 | 8 |
| 3- | Permutation Flow Shop | 5 | 2 | 4 | 3 | 15 |
| 4- | Personnel Scheduling | 1 | 3 | 4 | 3 | 12 |
| 5- | Traveling Salesman | 5 | 1 | 6 | 3 | 15 |
| 6- | Vehicle Routing | 4 | 2 | 4 | 2 | 12 |

Note: M: mutation. R&R: Ruin-recreate. HC: Hill-climbing. Xover: Crossover

## VI. COMPUTATIONAL RESULTS AND DISCUSSION

This section is devoted to assess the performance of GEP-HH against other hyper-heuristic methods in the literature. Our aims are:

- To assess the benefit of integrating the memory mechanism within GEP-HH.
- To test the generality and consistency of GEP-HH over six different problem domains and compare it to the state of the art of hyper-heuristic methods.

In this work, we have carried out, for each problem domain, two sets of experiments:

i) The first one compares the performance of the GEP-HH with the memory mechanism (GEP-HH) against GEP-HH without the memory mechanism (denoted as GEP-HH*) using the same parameter values and computational resources.
ii) The second evaluates the performance of GEP-HH against the top five hyper-heuristics of the first cross-

domain heuristic search challenge (CHeSC) [23]. These are: AdapHH [34], VNS-TW [46], ML [47], PHUNTER [48] and EPH [49].

Following CHeSC, rules and in order to make the comparison as fair as possible, for both experimental tests, the execution time is used as the stopping condition. It is determined by using the benchmark software provided by the organizers to ensure fair comparisons between researchers using different platforms [23]. We have used this software to determine the allowed execution time using our computer resources (i.e. 10 minutes on the benchmark machine).

The *best*, *average, standard deviation* and *median* of GEP-HH and GEP-HH* over independent 31 runs (adhering to the CHeSC competition rules) are reported for each instance. In addition, the percentage deviation from the best known value found in the hyper-heuristic literature is also calculated for each instance as follows:

$$\Delta(\%) = \frac{best_{GEP-HH} - best^*}{best^*}\% \qquad (3)$$

where $best_{GEP-HH}$ is the best result returned over 31 independent runs by GEP-HH and *best** is the best result obtained by other hyper-heuristic methods.

To demonstrate the generality, consistency and the effectiveness of GEP-HH across all tested problem domains, we have compared the performance of GEP-HH against GEP-HH* and existing hyper-heuristic methods based on generality, consistency, efficiency, statistical test and formula one (see [30] for more details).

## *A. The computational results of GEP-HH compared to GEP-HH**

The first set of experiments presents the comparison between GEP-HH and GEP-HH* across all of the six considered problems. Each problem domain contains 5 instances and the total number of tested instances is 30. The computational results of GEP-HH and GEP-HH* over 31 independent runs for the six problems are summarized in Table 11.

Observing the results reported in Table 11, we can make the following observations: in terms of solution quality, GEP-HH outperformed GEP-HH* on 18, tieing with GEP-HH* on 11 and being inferior on 1 (MAX-SAT Instances 4) out of 30 instances of the considered problem domains. From the average results perspective, it is clear that, across all instances of the considered problems domains, GEP-HH is the overall best.

In addition to the solution quality and the average results, it is natural to ask how consistent GEP-HH is, i.e., how likely GEP-HH would perform well over multiple runs on each instance compared to GEP-HH*. This question can be answered by analyzing the *standard deviation* and the *median* over 31 runs as well as the *box-plots* of solution distributions. In general, the standard deviation produced by GEP-HH is smaller than those from GEP-HH* for all instances of the considered problem domains (except PS 2 in Table 11). From the median perspective, we can draw the following conclusion: GEP-HH obtained better median results for 19, tieing on 3 and being slightly worse than GEP-HH* on 8 out of 30 instances of the considered problem domains. To save space, the *box-plot* figures (Figs. 8 to 13) are presented in the supplementary file. Figs. 8 to 13 show the *box-plot* of results distribution of GEP-HH and GEP-HH* for all instances of the considered problem domains, where one can clearly see that, for most instances, GEP-HH is more consistent than GEP-HH*. This indicates that GEP-HH is more consistent than GEP-HH* across all tested problem domains.

In addition to the above results, it is worth drawing some statistical significant conclusions regarding the performance of GEP-HH and GEP-HH*. Therefore, the Wilcoxon test (pairwise comparisons) with significant level of 0.05 is performed. The *p*-value of the Wilcoxon test of GEP-HH versus GEP-HH* are presented in the last column of Table 11. Where "S+" indicate GEP-HH is statistically better than GEP-HH* (*p*-value <0.05), "S-" indicate GEP-HH outperformed by GEP-HH* (*p*-value >0.05) and "~" indicate both algorithms have the same performance (*p*-value =0.05). The results in Table 11 (last column) show that GEP-HH is statistically better than GEP-HH* on 23 instances, not statistically better than GEP-HH* on 5 and perform the same as GEP-HH* on 2 instances out of 30 tested instances of the considered problem domains.

To summarize, the results demonstrate that GEP-HH is better than GEP-HH* in term of consistency, efficiency and generality (with regards to the tested instances of the considered problem domains). This is mainly due to the use of memory mechanism within GEP-HH which has a positive effect on the ability of GEP-HH in producing good quality and consistent results compared to GEP-HH*.

TABLE 11 THE RESULT OF GEP-HH COMPARING TO GEP-HH* FOR ALL PROBLEM DOMAINS

| | ***Instances*** | ***GEP-HH*** | | | | ***GEP-HH**** | | | | ***GEP-HH vs. GEP-HH**** |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ***Best*** | ***Average*** | ***Std*** | ***Median*** | ***Best*** | ***Average*** | ***Std*** | ***Median*** | ***p-value*** |
| **MAX-SAT** | SAT 1 | **1** | **4.4** | **1.70** | **3** | **1** | 5.0 | 1.74 | **3** | **S+** |
| | SAT 2 | **1** | **13.0** | **11.01** | **3** | 5 | 20.4 | 13.73 | 5 | **S+** |
| | SAT 3 | **1** | **3.8** | **2.44** | **2** | **1** | 4.7 | 3.26 | 3 | **S+** |
| | SAT 4 | 4 | **8.0** | **4.60** | **4** | **1** | 12.3 | 6.78 | 8 | S- |
| | SAT 5 | **7** | **7.8** | **0.88** | **7** | **7** | 10.3 | 3.20 | 8 | **S+** |
| **Bin Packing** | BP 1 | **0.0131** | **0.029** | **0.013** | 0.0192 | 0.034 | 0.053 | 0.021 | **0.0168** | **S+** |
| | BP 2 | **0.0029** | **0.005** | **0.003** | **0.0032** | 0.0067 | 0.011 | 0.006 | 0.0036 | **S+** |
| | BP 3 | **0.0011** | **0.003** | **0.002** | 0.0039 | 0.0035 | 0.014 | 0.004 | **0.0038** | **S+** |
| | BP 4 | **0.1083** | **0.108** | **0.001** | **0.1083** | **0.1083** | 0.115 | 0.024 | 0.1085 | S- |
| | BP 5 | **0.0031** | **0.015** | **0.010** | **0.0066** | **0.0031** | 0.027 | 0.016 | **0.0066** | **S+** |
| **F** | FS 1 | **6212** | **6243.29** | **10.39** | **6245** | **6212** | 6283.12 | 89.57 | 6248 | **S+** |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | FS 2 | **26721** | **26821** | **83.79** | 26898 | 26744 | 26887.9 | 85.12 | **26804** | **S+** |
| | FS 3 | **6285** | **6325.83** | **11.87** | 6326 | 6295 | 6335.83 | 16.25 | **6323** | **S+** |
| | FS 4 | **11320** | **11376.7** | **24.56** | 11377 | 11327 | 11377 | 27.12 | **11359** | **S+** |
| | FS 5 | **26530** | **26616** | **40.70** | 26634 | 26531 | 26638 | 50.37 | **26604** | **S+** |
| **Personnel Scheduling** | PS 1 | **11** | **18.64** | **3.91** | **21** | 14 | 33.80 | 24.10 | 22 | **S+** |
| | PS 2 | **9345** | **10830.4** | 1660.5 | **9628** | **9345** | 11077.0 | **1403.8** | 9630 | **S-** |
| | PS 3 | **3123** | **3312.16** | **83.10** | 3351 | 3124 | 3369.38 | 104.79 | **3231** | ~ |
| | PS 4 | **1364** | **1541.48** | **86.52** | **1555** | 1378 | 1619.54 | 133.14 | 1590 | **S+** |
| | PS 5 | **280** | **306.54** | **14.25** | **315** | 290 | 322.45 | 27.03 | 320 | **S+** |
| **Traveling Salesman** | TSP 1 | **48194.9** | **48222.72** | **38.41** | **48194.9** | **48194.9** | 48519.82 | 450.35 | **48194.9** | S- |
| | TSP 2 | **20754969** | **21227727** | **255264** | **21268571** | 20910693 | 21536045 | 760071.4 | 21270792 | ~ |
| | TSP 3 | **6796** | **6828.34** | **13.80** | **6810.5** | **6796.0** | 6868.6 | 54.98 | 6816.2 | **S+** |
| | TSP 4 | **65952.1** | **67118.9** | **493.71** | **67105.2** | 66448.2 | 67360.89 | 624.27 | 66898.2 | **S+** |
| | TSP 5 | **52050** | **54393.66** | **1015.18** | **54755.3** | 52052.7 | 55547.7 | 1879.55 | 54896.8 | **S+** |
| **Vehicle Routing** | VRP 1 | **58052.1** | **60046.3** | **1444.7** | **60720.0** | 67012.9 | 82505.9 | 5722.2 | 83094.9 | **S+** |
| | VRP 2 | **12261.0** | **12814.52** | **519.7** | **12337.9** | 12263.0 | 13639.4 | 907.4 | 13341.0 | **S+** |
| | VRP 3 | **142479.1** | **145294.4** | **1622.3** | 145418.9 | 142562.5 | 145664.7 | 1857.9 | **145329.9** | S- |
| | VRP 4 | **20650.8** | **20653.6** | **1.3** | **20653.8** | **20650.8** | 20684.2 | 6.3 | 20683.5 | **S+** |
| | VRP 5 | **144258.1** | **148943.6** | **1365.3** | **149007.9** | **144258.1** | 149326.4 | 2488.1 | 149107.9 | **S+** |

## B. *The computational results of GEP-HH compared to other hyper-heuristic methods*

We now assess the performance GEP-HH versus the top five hyper-heuristic methods from the CHeSC competition [23] (AdapHH, VNS-TW, ML, PHUNTER and EPH) from the *best* and *median* results perspective. In addition, we have also included the results of GEP-HH* (without memory) in the comparison to assess its ability in producing good quality solutions compared to the top five hyper-heuristic methods from the CHeSC competition. Table 12 present the best, percentage deviation and instances ranking results for the six problems obtained by GEP-HH along with a comparison with respect to the best result of top five hyper-heuristic methods from the CHeSC competition. Please note that all the compared methods (GEP-HH, GEP-HH* and the top five hyper-heuristics) used the 10 minute execution time as the stopping condition which is determined by the benchmark software provided by the CHeSC organizers.

The results in Table 12 suggest that, out of 30 instances, GEP-HH outperformed the top five hyper-heuristic methods on 12 instances, match the best results on 12 instances and is inferior on 6 instances. We can also remark that GEP-HH without memory mechanism (GEP-HH*) manages to produce new best results for 6 instances and tieing on 12 out of 30 instances compared to the top five hyper-heuristic methods.

In Table 13, we provide the median, percentage deviation and instances ranking results achieved by GEP-HH in comparison with the median results obtained by the top five hyper-heuristic methods from the CHeSC competition as well as GEP-HH* median results. It is clear from Table 13 that, GEP-HH obtained better median results for 4 instances and tie with other hyper-heuristic methods on 8 out of 30 instances. Table 13 also show that GEP-HH without memory mechanism (GEP-HH*) obtained better median results for 1 instance and matched the best in 6 out of 30 instances of the considered problem domains.

To summarize, even though GEP-HH did not manage to obtain the best results for all instances, the percentage deviation of these instances is, however, relatively small and GEP-HH achieved the second best and third best results for other instances. One can clearly see that both GEP-HH and GEP-HH* have generalized well across all tested domains and produced good quality results compared to the top five hyper-heuristic methods in the existing literature.

TABLE 12 THE BEST RESULT OF GEP-HH and GEP-HH* COMPARING TO THE TOP FIVE HYPER-HEURISTICS

| | | ***GEP-HH*** | | | ***GEP-HH**** | ***The top five hyper-heuristic framework from CHeSC competition*** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***Instances*** | ***Best*** | **Δ(%)** | ***Rank*** | ***Best*** | ***AdapHH*** | ***VNS-TW*** | ***ML*** | ***PHUNTER*** | ***EPH*** |
| **MAX-SAT** | SAT 1 | **1** | **0.0** | **1** | 1 | **1** | **1** | **1** | **1** | 4 |
| | SAT 2 | **1** | **0.0** | **1** | 5 | 3 | **1** | 3 | 5 | 5 |
| | SAT 3 | **1** | **0.0** | **1** | 1 | **1** | **1** | **1** | 2 | 2 |
| | SAT 4 | 4 | 300 | 2 | 1 | **1** | **1** | 4 | 4 | 5 |
| | SAT 5 | **7** | **0.0** | **1** | 7 | 9 | **7** | **7** | **7** | **7** |
| **Bin Packing** | BP 1 | **0.0131** | **0** | **1** | 0.034 | **0.0131** | 0.0298 | 0.0323 | 0.0397 | 0.0430 |
| | BP 2 | 0.0029 | 3.5 | 2 | 0.0067 | **0.0028** | 0.0036 | 0.0067 | 0.0034 | 0.0034 |
| | BP 3 | 0.0011 | 175 | 2 | 0.0035 | **0.0004** | 0.0136 | 0.0124 | 0.0178 | 0.0080 |
| | BP 4 | **0.1083** | **0** | **1** | 0.1083 | **0.1083** | 0.1087 | 0.1084 | 0.1088 | **0.1083** |
| | BP 5 | **0.0031** | **0** | **1** | 0.0031 | **0.0031** | 0.0238 | 0.0178 | 0.0318 | 0.0136 |
| **Flow Shop** | FS 1 | **6212** | **-0.03** | **1** | 6212 | 6214 | 6230 | 6226 | 6221 | 6232 |
| | FS 2 | **26721** | **-0.06** | **1** | 26744 | 26757 | 26765 | 26744 | 26786 | 26738 |
| | FS 3 | **6285** | **-0.2** | **1** | 6295 | 6303 | 6303 | 6304 | 6303 | 6309 |
| | FS 4 | 11320 | 0.01 | 2 | 11327 | **11318** | 11333 | 11338 | 11336 | 11328 |
| | FS 5 | **26530** | **-0.01** | **1** | 26531 | 26541 | 26535 | 26559 | 26600 | 26569 |
| **Sc** | PS 1 | **11** | **0.00** | **1** | 14 | 17 | 13 | **11** | 13 | 16 |
| | PS 2 | **9345** | **-0.02** | **1** | 9345 | 9435 | 9347 | 9436 | 9624 | 9747 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | PS 3 | **3123** | **-0.03** | **1** | 3124 | 3142 | 3124 | 3138 | 3142 | 3142 |
| | PS 4 | 1364 | 1.03 | 2 | 1378 | 1448 | 1370 | 1384 | **1350** | 1469 |
| | PS 5 | **280** | **-3.44** | **1** | 290 | 295 | 290 | 300 | 290 | 310 |
| **Traveling Salesman** | TSP 1 | **48194.9** | **0.00** | **1** | 48194.9 | **48194.9** | **48194.9** | **48194.9** | **48194.9** | **48194.9** |
| | TSP 2 | 20754969 | 0.01 | 3 | 20910693 | **20752853.8** | 2084855.6 | 20793219.8 | 20754199.8 | 20941645.1 |
| | TSP 3 | **6796** | **0.00** | **1** | 6796.0 | 6797.5 | **6796.0** | 6805.3 | **6796.0** | 6799.2 |
| | TSP 4 | **65952.1** | **-0.009** | **1** | 66448.2 | 66277.1 | 66830.2 | 66428.2 | 66641.4 | **65958.6** |
| | TSP 5 | **52050** | **-0.006** | **1** | 52052.7 | 52383.8 | 52896.5 | 52626.7 | 52172.0 | 52053.4 |
| **Vehicle Routing** | VRP 1 | **58052.1** | **0.0** | **1** | 67012.9 | **58052.1** | 68340.4 | 67622.1 | 61139.3 | 63932.2 |
| | VRP 2 | **12261.0** | **-0.016** | **1** | 12263.0 | 13304.9 | 13298.1 | 13298.4 | 12263.0 | 13284.0 |
| | VRP 3 | **142479.1** | **-0.02** | **1** | 142562.5 | 145481.5 | 144012.6 | 142517.0 | 143663.9 | 143510.8 |
| | VRP 4 | **20650.8** | **0.0** | **1** | 20650.8 | 20652.3 | 20651.1 | 20651.1 | **20650.8** | **20650.8** |
| | VRP 5 | **144258.1** | **-1.17** | **1** | 144258.1 | 146154.0 | 146513.6 | 146200.8 | 146472.9 | 145976.5 |

TABLE 13 THE MEDIAN RESULT OF GEP-HH and GEP-HH* COMPARING TO THE TOP FIVE HYPER-HEURISTICS

| | | ***GEP-HH*** | | | ***GEP-HH**** | ***The top five hyper-heuristic framework from CHeSC competition*** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***Instances*** | ***Median*** | ***Δ(%)*** | ***Rank*** | ***Median*** | ***AdapHH*** | ***VNS-TW*** | ***ML*** | ***PHUNTER*** | ***EPH*** |
| **MAX-SAT** | SAT 1 | **3** | **0.0** | **1** | 3 | **3** | **3** | 5 | 5 | 7 |
| | SAT 2 | **3** | **0.0** | **1** | 5 | 5 | **3** | 10 | 11 | 11 |
| | SAT 3 | **2** | **0.0** | **1** | 3 | **2** | **2** | 3 | 4 | 6 |
| | SAT 4 | 4 | 33.3 | 2 | 8 | **3** | **3** | 9 | 9 | 15 |
| | SAT 5 | **7** | **-12.5** | **1** | 8 | 8 | 10 | 8 | 8 | 13 |
| **Bin Packing** | BP 1 | 0.0192 | 19.2 | 2 | 0.0168 | **0.0161** | 0.0370 | 0.0421 | 0.0479 | 0.0504 |
| | BP 2 | **0.0032** | **-11.1** | **1** | 0.0036 | 0.0036 | 0.0072 | 0.0075 | 0.0036 | 0.0036 |
| | BP 3 | 0.0039 | 8.3 | 2 | 0.0038 | **0.0036** | 0.0167 | 0.0146 | 0.0201 | 0.0113 |
| | BP 4 | **0.1083** | **0** | **1** | 0.1085 | **0.1083** | 0.1088 | 0.1085 | 0.1091 | 0.1087 |
| | BP 5 | 0.0066 | 88.5 | 2 | 0.0066 | **0.0035** | 0.0278 | 0.0218 | 0.0395 | 0.0224 |
| **Flow Shop** | FS 1 | 6245 | 0.08 | 2 | 6248 | **6240** | 6251 | 6245 | 6253 | 6250 |
| | FS 2 | 26898 | 0.36 | 6 | 26804 | 26814 | 26803 | **26800** | 26858 | 26816 |
| | FS 3 | 6326 | 0.04 | 2 | 6323 | 6326 | 6328 | **6323** | 6350 | 6347 |
| | FS 4 | 11377 | 0.15 | 3 | 11359 | **11359** | 11376 | 11384 | 11388 | 11397 |
| | FS 5 | 26634 | 0.12 | 3 | 26604 | 26643 | **26602** | 26610 | 26677 | 26640 |
| **Personnel Scheduling** | PS 1 | 21 | 16.6 | 2 | 22 | 24 | 19 | **18** | 25 | 22 |
| | PS 2 | **9628** | **0.0** | **1** | 9630 | 9667 | **9628** | 9812 | 10136 | 10074 |
| | PS 3 | 3351 | 3.9 | 6 | 3231 | 3289 | **3223** | 3228 | 3255 | 3232 |
| | PS 4 | **1555** | **-2.2** | **1** | 1590 | 1765 | 1590 | 1605 | 1595 | 1615 |
| | PS 5 | **315** | **0.0** | **1** | 320 | 325 | 320 | **315** | 320 | 345 |
| **Traveling Salesman** | TSP 1 | **48194.9** | **0.0** | **1** | 48194.9 | **48194.9** | **48194.9** | **48194.9** | **48194.9** | **48194.9** |
| | TSP 2 | 21041571 | 0.01 | 2 | 21270792 | **20822145.7** | 21042675.8 | 21093828.3 | 21246427.7 | 21064606.3 |
| | TSP 3 | **6810.5** | **0.0** | **1** | 6816.2 | **6810.5** | 6819.1 | 6820.6 | 6813.6 | 6811.9 |
| | TSP 4 | 67105.2 | 0.5 | 4 | 66898.2 | 66879.8 | 67378.0 | 66894.0 | 67136.8 | **66756.2** |
| | TSP 5 | 54755.3 | 3.4 | 5 | 54896.8 | 53099.8 | 54028.6 | 54368.4 | 52934.4 | **52925.3** |
| **Vehicle Routing** | VRP 1 | **60720.0** | **-0.20** | **1** | 83094.9 | 60900.6 | 76147.1 | 80671.3 | 64717.8 | 74715.8 |
| | VRP 2 | 12337.9 | 0.3 | 2 | 13341.0 | 13347.6 | 13367.9 | 13329.8 | **12290.0** | 13335.6 |
| | VRP 3 | 145418.9 | 0.05 | 2 | 145329.9 | 148516.8 | 148206.2 | **145333.5** | 146944.4 | 162188.5 |
| | VRP 4 | 20653.8 | 0.01 | 2 | 20683.5 | 20656.6 | 21642.9 | 20654.1 | **20650.8** | **20650.8** |
| | VRP 5 | 149007.9 | 0.23 | 4 | 149107.9 | 148689.2 | 149132.4 | 148975.1 | **148659.0** | 155224.7 |

## C. *DISCUSSION*

The numerical results presented throughout this work demonstrate that, across six very different combinatorial optimization problems, GEP-HH achieved favorable results compared to the top five hyper-heuristic methods from the CHeSC competition. More importantly, out of the 30 instances GEP-HH matched the best results for 12 instances and manages to obtain new best results for 12 instances. In all domains, the *standard deviation* and the *percentage deviation* of GEP-HH reveal that GEP-HH results are stable and very close to the best results obtained by other hyper-heuristic methods. These results are also supported by statistical tests and box-plots of solution distribution. In order to compare the performance of GEP-HH against the top five hyper-heuristic methods from the CHeSC competition (AdapHH, VNS-TW, ML, PHUNTER and EPH) more accurately, we have conducted the following comparison:

i) In the first comparison we used Formula one that was used in the CHeSC competition [23] to calculate the score of GEP-HH and the top five hyper-heuristic methods. Table 14 shows the overall rankings of GEP-HH and the top five hyper-heuristic methods (the higher the better). We also included GEP-HH* in the comparisons. It is interesting to note that GEP-HH obtained the first rank, whilst, GEP-HH* obtained the third rank compared to the top five hyper-heuristic methods.

TABLE 14 THE RANKING OF GEP-HH AND THE TOP FIVE HYPER-HEURISTICS

| # | *Hyper-heuristics* | *Score* |
|---|---|---|
| 1- | **GEP-HH** | **167.03** |

| | | |
|---|---|---|
| 2- | **AdapHH** | 155.7 |
| 3 | **GEP-HH*** | 130.43 |
| 4- | **VNS-TW** | 110.2 |
| 5- | **ML** | 101.33 |
| 6- | **PHUNTER** | 63.83 |
| 7- | **EPH** | 75.25 |

ii) In the second comparison, we conducted a multiple comparison statistical tests between GEP-HH and the top five hyper-heuristic 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. The p-value of Friedman (p-value = 0.000) and Iman-Davenport (p-value =0.000) are less than the critical level 0.05, which implies that there is a significant difference between the compared methods. As a result, we conducted a Friedman test to calculate the average ranking of each method. Table 15 summarizes the average ranking (the lower the better) produced by the Friedman test for each method. It is obvious that, GEP-HH ranked the first, followed by AdapHH, GEP-HH*, ML, VNS-TW, PHUNTER and EPH.

TABLE 15 THE AVERAGE RANK OBTIANED BY FRIEDMAN TEST

| *#* | *Hyper-heuristics* | *Ranking* |
|---|---|---|
| 1- | **GEP-HH** | **3.2333** |
| 2- | **AdapHH** | 3.2667 |
| 3- | **GEP-HH*** | 3.6833 |
| 4- | **ML** | 3.8667 |
| 5- | **VNS-TW** | 4.05 |
| 6- | **PHUNTER** | 4.9333 |
| 7- | **EPH** | 4.9667 |

Overall, the advantages of the proposed framework are the ability to utilize the information about the current state during instance solving to automatically generate the heuristic selection mechanism and an acceptance criterion. Results demonstrate that it provides a general mechanism regardless of the nature and complexity of the instances and can be applied to other domains without many changes (i.e. the user only needs to change the low level heuristics). Applying a methodology to other problem domains or even different instances of the same problem usually requires a considerable amount of modification (e.g. change algorithm parameters or structures). Our GEP-HH provides automated heuristic method that can cope with not only different instances of the same problem, but we have demonstrated its generality across six different problem domains. We would hope that the proposed methodology would also generalize to other domains.

## VII. Conclusions

In this work, we have proposed a new hyper-heuristic framework for combinatorial optimization problems. At the higher level, we have introduced a gene expression programming framework to automatically generate the high level heuristic of the hyper-heuristic framework. The proposed gene expression programming framework evolves a population of individuals and each one is decoded into a heuristic selection mechanism and an acceptance criterion. The evolved heuristic selection mechanism takes the current state as input (pervious performance) and decides which low level heuristic is to be applied. Then, the generated solution is accepted if it satisfies the evolved acceptance criterion. At the lower level, we employed a set of human designed perturbative low level heuristics to perturb the solution of a given instance. To diversify the search, we have embedded the proposed hyper-heuristic with a memory mechanism, which contains a set of high quality and diverse solutions, which are updated during the search.

We have shown that gene expression programming algorithm can be effectively used to automatically generate the high level heuristics of the perturbative hyper-heuristic framework. The efficiency, consistency and the generality of GEP-HH is demonstrated across six challenging problems using HyFlex software. The experimental results demonstrate that GEP-HH achieves highly competitive results, if not superior, and generalizes well over six problem domains (MAX-SAT, one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing problems) when compared to GEP-HH without a memory mechanism as well as the top five hyper-heuristic methods from the CHeSC competition. The main contributions of this work are:

1- The development of a GEP-HH hyper-heuristic framework that automatically generates, during instance solving process, the high level heuristic (heuristic selection mechanism and the acceptance criteria) of the improvement based hyper-heuristic framework.

2- The development of a population based hyper-heuristic framework that uses a memory mechanism of a set of solutions, which is updated during the solving process to effectively diversify the search.

3- The development of a hyper-heuristic framework that is not customized to specific problems class and can be applied to different problems without much development effort.

In our future work, we intend to investigate the effectiveness of the GEP-HH across other combinatorial optimization problems.

## REFERENCES

[1] E. G. Talbi, *Metaheuristics: From design to implementation*: Wiley Online Library, 2009.

[2] T. Weise, M. Zapf, R. Chiong, and A. Nebro, "Why is optimization difficult?," *Nature-Inspired Algorithms for Optimisation,* pp. 1-50, 2009.

[3] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," *Handbook of Metaheuristics,* pp. 449-468, 2010.

[4] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," *Adaptive and multilevel metaheuristics,* pp. 3-29, 2008.

[5] Y. Hamadi, E. Monfroy, and F. Saubion, "What is Autonomous Search?," *Hybrid Optimization,* pp. 357-391, 2011.

[6] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: an automatic algorithm configuration framework,"

*Journal of Artificial Intelligence Research,* vol. 36, pp. 267-306, 2009.

[7] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation,* vol. 3, pp. 124-141, 1999.

[8] X. Chen, Y. S. Ong, M. H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Transactions on Evolutionary Computation,* vol. 15, pp. 591-607, 2011.

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

[10] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Applied Soft Computing,* 2011.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Tradeoffs in the empirical evaluation of competing algorithm designs," *Annals of Mathematics and Artificial Intelligence,* vol. 60, pp. 65-89, 2010.

[12] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation,* vol. 1, pp. 67-82, 1997.

[13] Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, "Analyzing bandit-based adaptive operator selection mechanisms," *Annals of Mathematics and Artificial Intelligence,* vol. 60, pp. 25-64, 2010.

[14] M. Brunato and R. Battiti, "R-EVO: A Reactive Evolutionary Algorithm for the Maximum Clique Problem," *IEEE Transactions on Evolutionary Computation,* pp. 1-13, 2010.

[15] J. A. Vrugt, B. A. Robinson, and J. M. Hyman, "Self-adaptive multimethod search for global optimization in real-parameter spaces," *IEEE Transactions on Evolutionary Computation,* vol. 13, pp. 243-259, 2009.

[16] X. Chen and Y.-S. Ong, "A Conceptual Modeling of Meme Complexes in Stochastic Search," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* vol. 42, pp. 612-625, 2012.

[17] M. N. Le, Y.-S. Ong, Y. Jin, and B. Sendhoff, "A Unified Framework for Symbiosis of Evolutionary Mechanisms with Application to Water Clusters Potential Model Design," *IEEE Computational Intelligence Magazine,* vol. 7, pp. 20-35, 2012.

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

[19] 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, in press,* 2013.

[20] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis,* vol. 12, pp. 3-23, 2008.

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

[22] 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, pp. 73-78 vol. 1.

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

[24] F. G. Lobo, C. F. Lima, and Z. Michalewicz, *Parameter setting in evolutionary algorithms* vol. 54: Springer Verlag, 2007.

[25] A. Bölte and U. W. Thonemann, "Optimizing simulated annealing schedules with genetic programming," *European Journal of Operational Research,* vol. 92, pp. 402-416, 1996.

[26] 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.

[27] W. Langdon and R. Poli, "Evolving problems to learn about particle swarm optimizers and other search algorithms," *IEEE Transactions on Evolutionary Computation,* vol. 11, pp. 561-578, 2007.

[28] J. Tavares and F. B. Pereira, "Towards the development of self-ant systems," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO)*, 2011, pp. 1947-1954.

[29] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical Evolution of Local Search Heuristics," *IEEE Transactions on Evolutionary Computation,* pp. 1-1, 2012.

[30] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Grammatical Evolution Hyper-heuristic for Combinatorial Optimization problems," *IEEE Transactions on Evolutionary Computation, to appear* 2013.

[31] 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.

[32] 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.

[33] R. Qu and E. K. Burke, "Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems," *Journal of the Operational Research Society,* vol. 60, pp. 1273-1285, 2008.

[34] M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe, "An intelligent hyper-heuristic framework for chesc 2011," in *The 6th Learning and Intelligent Optimization Conference (LION12).* Paris, France, 2012.

[35] X. Chen, "An algorithm development environment for problem-solving: software review," *Memetic Computing,* vol. 4, pp. 149-161, 2012/06/01 2012.

[36] M. Bader-El-Den and R. Poli, "Generating SAT local-search heuristics using a GP hyper-heuristic framework," in *Artificial Evolution*, 2008, pp. 37-49.

[37] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering,* vol. 54, pp. 453-473, 2008.

[38] E. G. Talbi and V. Bachelet, "Cosearch: A parallel cooperative metaheuristic," *Journal of Mathematical Modelling and Algorithms,* vol. 5, pp. 5-22, 2006.

[39] 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.

[40] M. Hyde, G. Ochoa, J. A. Vázquez-Rodríguez, and T. Curtois, "A HyFlex Module for the MAX-SAT Problem," University of Nottingham, Tech. Rep.2011.

[41] M. Hyde, G. Ochoa, T. Curtois, and J. Vázquez-Rodríguez, "A hyflex module for the one dimensional bin-packing problem," School of Computer Science, University of Nottingham, Tech. Rep 2010.

[42] J. A. Vázquez-Rodrıguez, G. Ochoa, T. Curtois, and M. Hyde, "A hyflex module for the permutation flow shop problem," School of Computer Science, University of Nottingham, Tech. Rep 2010.

[43] T. Curtois, G. Ochoa, M. Hyde, and J. A. Vázquez-Rodríguez, "A hyflex module for the personnel scheduling problem," School of Computer Science, University of Nottingham, Tech. Rep 2010.

[44] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: a survey," *Operations Research,* pp. 538-558, 1968.

[45] P. Toth and D. Vigo, "The vehicle routing problem, Society for industrial and applied mathematics," *SIAM Monographs on Discrete Mathematics and Applications,* 2002.

[46] P. C. Hsiao, T. C. Chiang, and L. C. Fu, "A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in CHeSC 2011 competition " CHeSC 2011 competition 2011.

[47] M. Larose, "A Hyper-heuristic for the CHeSC 2011," CHeSC 2011 competition 2011.

[48] F. Xue, C. Chan, W. Ip, and C. Cheung, "Pearl Hunter: A Hyper-heuristic that Compiles Iterated Local Search Algorithms," CHeSC 2011 competition 2011.

[49] D. Meignan, "An Evolutionary Programming Hyper-heuristic with Co-evolution for CHeSC'11," CHeSC 2011 competition 2011.