Grammatical Evolution Hyper-heuristic for Combinatorial Optimization problems

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

***Abstract*— Designing generic problem solvers that perform well across a diverse set of problems is a challenging task. In this work, we propose a hyper-heuristic framework to automatically generate an effective and generic solution method by utilizing grammatical evolution. In the proposed framework, grammatical evolution is used as an online solver builder, which takes several heuristic components (e.g. different acceptance criteria and different neighborhood structures) as inputs and evolves templates of perturbation heuristics. The evolved templates are improvement heuristics which represent a complete search method to solve the problem at hand. To test the generality and the performance of the proposed method, we consider two well-known combinatorial optimization problems; exam timetabling (Carter and ITC 2007 instances) and the capacitated vehicle routing problem (Christofides and Golden instances). We demonstrate that the proposed method is competitive, if not superior, when compared to state of the art hyper-heuristics, as well as bespoke methods for these different problem domains. In order to further improve the performance of the proposed framework we utilize an adaptive memory mechanism which contains a collection of both high quality and diverse solutions and is updated during the problem solving process. Experimental results show that the grammatical evolution hyper-heuristic, with an adaptive memory, performs better than the grammatical evolution hyper-heuristic without a memory. The improved framework also outperforms some bespoke methodologies which have reported best known results for some instances in both problem domains.**

***Index Terms*—Grammatical Evolution, Hyper-heuristics, Timetabling, Vehicle Routing**

I. Introduction

Combinatorial optimization can be defined as the problem of finding the best solution(s) among all those available for a given problem [1]. These problems are encountered in many real world applications such as scheduling, production planning, routing, economic systems and management [1]. Many real world optimization problems are complex and very difficult to solve. This is due to the large, and often heavily constrained, search spaces which make their modeling (let alone solving) a very complex task [2]. Usually, heuristic methods are used to solve these problems, as exact methods often fail to obtain an optimal solution in reasonable times. The main aim of heuristic methods, which provide no guarantee of returning an optimal solution (or even near optimal solution), is to find a reasonably good solution within a realistic amount of time [3, 4]. Meta-heuristic algorithms provide some high level control strategy in order to provide effective navigation of the search space. A vast number of meta-heuristic algorithms, and their hybridizations, have been presented to solve optimization problems. Examples of meta-heuristic algorithms include scatter search, tabu search, genetic algorithms, genetic programming, memetic algorithms, variable neighborhood search, guided local search, GRASP, ant colony optimization, simulated annealing, iterated local search, multi-start methods and parallel strategies [3],[4].

Given a problem, an interesting question that comes to mind is:

> *Which algorithm is the most suitable for the problem at hand and what are the optimal structures and parameter values?*

The most straightforward answer to the above question might be to employ trial-and-error to find the most suitable meta-heuristic from the large variety of those available, and then employ trial-and-error to determine the appropriate structures and parameter values. While these answers seem reasonable, in terms of the computational time involved, it is impractical in many real world applications. Many bespoke meta-heuristic algorithms that have been proposed over the years are manually designed and tuned, focusing on producing good results for specific problem instances. The manually designed algorithms (customized by the user and not changed during problem solving) that have been developed over the years are problem specific, i.e. they are able to obtain high quality results for just a few problem instances, but usually fail on other instances even of the same problem and cannot be directly applied to other optimization problems. Of course, the No Free Lunch Theorem [5] states that a general search method does not exist, but it does not mean that we cannot investigate *more general* search algorithms to explore the limits of such an algorithm [6-8].

Numerous attempts have been made to develop automated search methodologies that are able to produce good results

Nasser R. Sabar and Masri Ayob are with Data Mining and Optimisation 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 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

across several problem domains and/or instances. Hyper-heuristics [6], meta-learning [9], parameter tuning [10], reactive search [11], adaptive memetic algorithms [12] and multi-method [13], are just some examples. The performance of any search method critically depends on its structures and parameter values [6]. Furthermore, different search methodologies, coupled with different structures and parameter settings may be needed to cope with problem instances or different problem domains [9],[10]. A search may even benefit from adapting as it attempts to solve a given instance. Therefore, the performance of any search method may be enhanced by automatically adjusting their structures or parameter values during the problem solving process. Thus, the ultimate goal of automated heuristic design is to develop search methodologies that are able to adjust their structures or parameter values during the problem solving process and work well, not only across different instances of the same problem, but also across a diverse set of problem domains [6], [9], [10].

Motivated by these aspects, particularly the hyper-heuristic framework [6], in this work, we propose a grammatical evolution hyper-heuristic framework (GE-HH) to generate local search templates during the problem instance solving process, as depicted in Fig 1.

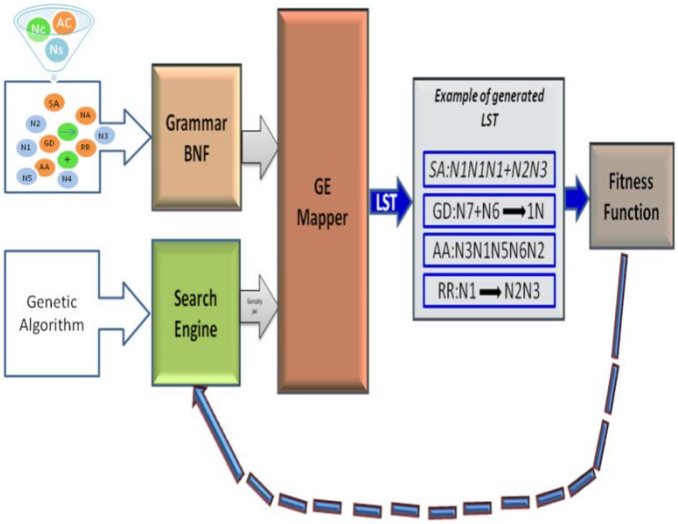


Fig.1.The GE-HH framework

The evolved templates represent a complete local search method which contains the acceptance criteria of the local search algorithm (to determine away of escaping from local optima), the local search structures (neighborhoods), and their combination. The GE-HH operates on the search space of heuristic components, instead of the solution space. In addition, GE-HH also maintains a set of diverse solutions, utilizing an adaptive memory mechanism which updates the solution quality and diversity as the search progresses. We choose grammatical evolution to search the space of heuristic components due to its ability to represent heuristic components and it being able to avoid the problem of code bloat that is often encountered in traditional genetic programming. Our objectives are:

- To design an automatic algorithm that works well across different instances of the same problem and also across two different problem domains.
- To merge the strengths of different search algorithms in one framework.
- To test the generality and consistency of the proposed method on two different problem domains.

The performance and generality of the GE-HH is assessed using two well-known NP-hard combinatorial optimization problems; examination timetabling (Carter [14] and ITC 2007 [15] instances) and the capacitated vehicle routing problem (Christofides [16] and Golden [17] instances). Although both domains have been extensively studied by the research community, the reasons of choosing them are twofold. Firstly, they represent real world applications and the state of the art results, we believe, can still be improved. Currently, a variety of algorithms have achieved very good results for *some* instances. However, most methodologies fail on generality and consistency. Secondly, these two domains have been widely studied in the scientific literature and we would like to evaluate our algorithm across two different domains that other researchers have studied. Although our intention is not to present an algorithm that can beat the state of the art, but rather can work well across different domains, our results demonstrate that GE-HH is able to update the best known results for some instances.

The remainder of the paper is organized as follows: the generic hyper-heuristic framework and its classification are presented in Section II. The grammatical evolution algorithm is presented in Section III, followed by our proposed GE-HH framework in Section IV. The experimental results and result comparisons are presented in Section V and VI, respectively. Finally discussions and concluding remarks are presented in Sections VII and VIII.

II. HYPER-HEURISTICS

Meta-heuristics are generic search methods that can be applied to solve combinatorial optimization problems. However, to find high quality solutions, meta-heuristics often need to be designed and tuned (as do many classes of algorithms, including those in this paper) and they are also often limited to one problem domain or even just a single problem instance. The objective for a solution methodology that is independent of the problem domain, serves as one of the main motivations for designing hyper-heuristic approaches [6],[18].

Recently, significant research attention has been focused on hyper-heuristics. Burke et al. [6] defined hyper-heuristics as

> *An automated methodology for selecting or generating heuristics to solve hard computational search problems.*

One possible hyper-heuristic framework is composed of two levels, known as *high* and *low* level heuristics (see Fig.2).

The *high* level heuristic is problem independent. It has no knowledge of the domain, only the number of heuristics that are available and (non-domain) statistical information that is allowed to pass through the domain barrier. Only the lower part of the framework has access to the objective function, the problem representation and the low level heuristics that have

been provided for the problem. During the problem solving process, the *high* level strategy decides which heuristic is called (without knowing what specific function it performs) at each decision point in the search process. Unlike meta-heuristics, hyper-heuristics operate over a search space of heuristics, rather than directly searching the solution space.

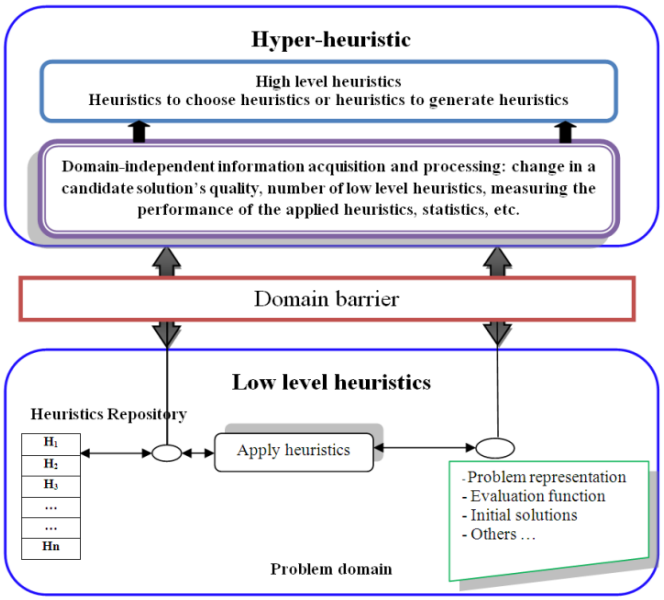


Fig.2. A generic hyper-heuristic framework

The *low* level heuristics correspond to a pool of candidates of problem dependent human-designed heuristics or components of existing heuristics which operate directly on the solution space for a given problem instance. Based on their past performance, heuristics compete with each other through learning, selection or generating mechanisms at a particular point to construct or improve a solution for a given problem instance.

The fact that the high level strategy is problem independent means that it can be applied to different problem domains with little development effort. Indeed, one of the goals of hyper-heuristics is to raise the level of generality of search methodologies and to build systems which are more generic than other methods [6].

Burke et al. [6] classified hyper-heuristics into two dimensions, based on the nature of the heuristic search space and the source of feedback during learning (see Fig.3).The nature of the heuristic search space can either be *heuristics to choose heuristics* or *heuristics to generate heuristics*. Heuristics can be called from a given pool of heuristics. For example, Burke et al. [19] used tabu search with reinforcement learning as a heuristic selection mechanism to solve nurse rostering and timetabling problems. Heuristics can also be generated by combining existing heuristic components. For example, Burke et al. [20],[21] employed genetic programming to evolve new low level heuristics to solve the bin packing problem.

The nature of the heuristic search space can be further classified depending on the type of low level heuristics as either constructive or perturbative. Constructive based hyper-heuristics start with an empty solution, and select low level heuristics to build a solution step by step. Perturbation based hyper-heuristics start with an initial solution and, at each decision point, select an appropriate improvement low level heuristic to perturb the solution. Based on the employed learning methods, two subclasses are distinguished: on-line or off-line.

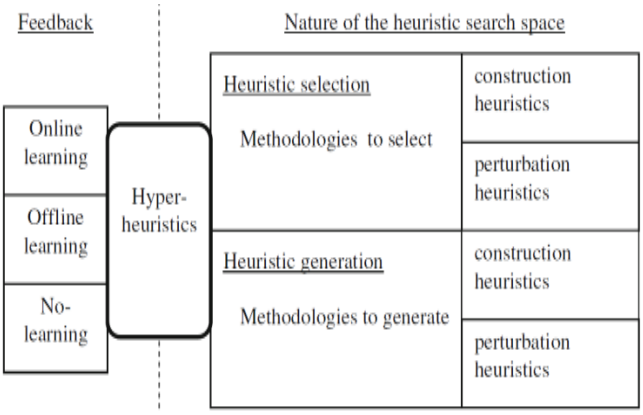


Fig.3. A classifications of hyper-heuristic approaches, according to two dimensions: (i) the nature of the heuristic search space and (ii) the source of feedback during learning [6].

In on-line hyper-heuristics, the learning takes place during the problem solving. Examples of online approaches include those based on genetic algorithms [22], tabu search[19], and local based search [23]. In off-line hyper-heuristics, learning occurs during the training phase before solving other problem instances, examples include those based on genetic programming [20] and learning classifier systems [24]. Recently, GE was utilized in [21] as an off-line heuristic builder to solve the bin packing problem. Our work differs from [21], where we use GE as an online solver builder, and is a much more general methodology that is able to address two problem domains, and produce best known results. In addition, the GE in [21] has been specifically designed and tested on the bin packing problem only (i.e. the grammar has been specifically designed for the bin packing problem only).

Our proposed GE-HH framework can be classified as an on-line generational hyper-heuristic. In this respect, it is the same as a genetic programming hyper-heuristic which generates heuristics. Genetic programming hyper-heuristics have been utilized to solve many combinatorial optimization problems including SAT [25],[26], scheduling [27] and bin packing [20],[28]. A recent, and comprehensive, review on hyper-heuristics is available in [29].

Most of the proposed genetic programming based hyper-heuristic approaches, however, are constructive heuristics. Their general limitation is that they are tailored to solve specific problems (e.g. SAT, bin packing, and TSP) using a restricted constructive heuristic component. This limitation restricts their applicability to cope with different problem domains without any redevelopment (e.g. redefine the terminals and functions). In addition, previous genetic programming based hyper-heuristics were only applied to one single domain, which raises the question to what extent they will generalize to other domains.

Motivated by the above, this work proposes an improvement based hyper-heuristic using grammatical

evolution. The proposed framework takes several heuristic components (e.g. acceptance criteria and neighborhood structures) as input and automatically generates a local search template by selecting the appropriate combination of these heuristic components. The differences between our approach and the previous genetic programming based hyper-heuristics in the literature are:

1. The proposed framework generates a perturbation local search template rather than constructive heuristics.
2. The proposed framework is not tailored to a particular problem domain e.g. it can be applied to several domains (the user only needs to change the neighborhood structures when applying it to another problem domain).
3. The proposed framework utilizes an adaptive memory mechanism to maintain solution diversity.

III. Grammatical Evolution

Grammatical evolution (GE) [30] is a variant of genetic programming (GP) [31]. It is a grammar based GP that can evolve a variable-length program in an arbitrary language. Unlike GP, GE uses a linear genome representation rather than a tree. The clear distinction between the genotype and phenotype in GE allows the evolutionary process (e.g. crossover) to be performed on the search space (variable length linear genotypic) without needing to tailor the diversity-generating operator to the nature of the phenotype[30],[31]. The mapping process of genotype and phenotype to generate a program is governed by a grammar which contains domain knowledge [30]. The grammar is represented by Backus Naur Form (BNF). The program is generated by using a binary string (genome) to determine which production rule in the BNF definition will be used. The GE general framework is composed of three procedures: grammar, search engine and a mapper procedure (see Fig.4).

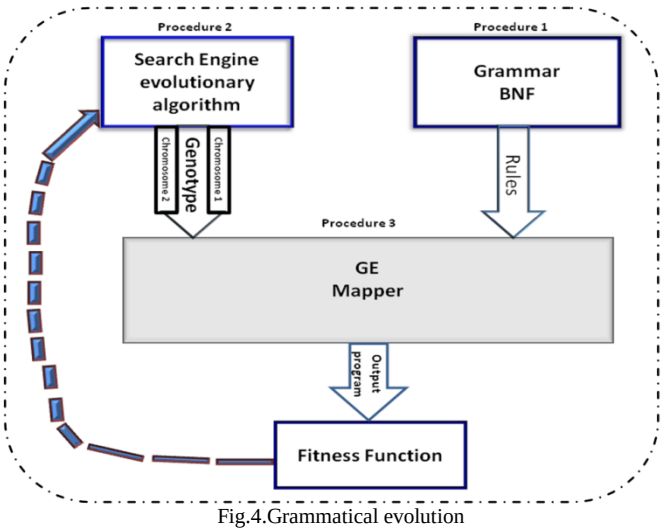


Fig.4.Grammatical evolution

A. The BNF Grammar

GE utilizes BNF to generate the output program [30],[31]. A suitable BNF grammar must be defined when solving a problem, and the definitions vary from one problem to another. The BNF grammar can be represented by a tuple <*T*, *N*, *S*, *P*> where *T* is the terminal set, *N* is the set of non terminals, *S* is the start symbol (a member of *N*) and *P* is a set of production rules. If more than one production rule is used within a particular *N*, the choice is delimited with the '|' symbol. Below is an example of BNF grammar (adopted from [30]):

T= {Sin, Cos, Tan, Log, +, -, /, *, (,)} // set of terminal
N= {expr, op, pre_op} // set of non-terminal
S= <expr>// starting symbol
and *P* can be represented as // production rules

(1) <expr>::=	<expr><op><expr>	(0)
	| (<expr><op><expr>)	(1)
	|<pre-op>(<expr>)	(2)
	|<var>	(3)
(2) <op>::=	+	(0)
	| -	(1)
	| /	(2)
	| *	(3)
(3) <var> ::=	X	(0)
(4) <pre_op> ::=	Sin	(0)

B. The Search Engine

GE uses a standard genetic algorithm as its search engine[30]. A candidate solution (genotype or chromosome) is represented by a one dimensional variable length string array. The gene in each chromosome is called a codon. Each codon is an 8-bit binary number (see Fig.5).

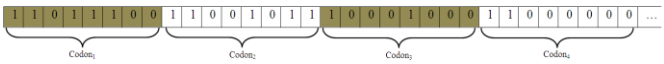


Fig.5. An example of genotype

The codon values are used in the mapper procedure to determine which rule to be selected for the non-terminal symbol when it is converted [30] (see Section III-C). The GA starts with a population of chromosomes, which are randomly generated. The fitness of each chromosome is calculated by executing its corresponding program. The fitness function varies from one domain to another. GA operators (selection, crossover, mutation and replacement) are then applied. At each generation, the evolved solutions (children) from the crossover and mutation operators are evaluated by converting them into its corresponding program via the mapper function. If the fitness of the new solution is better than the worst solution in the population, it will replace it. The process is repeated until a stopping condition is satisfied (e.g. number of generations).

C. The Mapper Procedure

The mapper function converts the genotype into a phenotype (i.e. a program). The function takes two inputs, the binary string (genotype) and the BNF grammar [30]. The

conversion from genotype to phenotype is carried out using the following rule:

Rule= (*codon integer value*) MOD (*number of rules for the current non-terminal*)

The mapper function begins by mapping the starting symbol into terminals. It converts each codon to its corresponding integer value. Assume we have the above BNF grammar (See Section III-A) and genotype (see Fig.5). First of all, convert all codon values to integers (with reference to Fig 4, this will be 220, 203, 17, 3, 109, 215, 104, 30). Then, starting from the starting symbol, apply the mapping rule to convert the leftmost non-terminal into a terminal until all non-terminals have been converted into terminals. The genotype-to-phenotype mapping process of the above BNF grammar and the solution (genotype) is illustrated in Table 1.

TABLE 1AN EXAMPLE OF THE MAPPING PROCESS

Input	No. of Choices	Rule	Result
<expr>	4	220 *MOD* 4= 0	<expr><op><expr>
<expr><op><expr>	4	203 *MOD* 4= 3	<var><op><expr>
X <op><expr>	4	17 *MOD* 4= 1	X -<expr>
X -<expr>	4	3 *MOD* 4= 3	X -<var>
X-X			

The mapper begins (see Table 1) with the starting symbol *<expr>*, and then reads the first codon (220). The starting symbol *<expr>* has four production rules to select from (see Section III-A). Following the mapping rules, the codon value and the number of production rules are used with the modular function to decide which rule to select, i.e. *220* MOD *4= 0*, which means we select the first production rule (*<expr><op><expr>*). Since this production rule is not a complete expression (it has at least one non-terminal), rules will be applied again. The process will continue from the leftmost non-terminal in the current production rule. Continuing with *<expr><op><expr>*, take the next codon value (*203*), the next production rule will be (*203* MOD *4= 3*) *<var><op><expr>*. Since *<var>* has only one choice, *<var>* will be replaced by *X* and the production rules will be *X<op><expr>*. Continuing with the same mapper rules until all non-terminals are converted to terminals, the complete expression will be *X-X*.

During the conversion process, not all codons may be used, or after using all codon values not all non-terminals have been replaced by terminals. In the case where non-terminals have been replaced with terminals but not all codon values have been used, the mapper process will simply ignore the rest. If all codon values have been used but the expression is still invalid, a wrapper procedure is invoked. The wrapper procedure reads the codon value from the left to right for a predefined number of iterations. If the wrapper procedure is finished but the complete expression is still not available, the genotype is given the lowest fitness value.

IV. THE GRAMMATICAL EVOLUTION HYPER-HEURISTIC FRAMEWORK

In this section we present the grammatical evolution hyper-heuristic (GE-HH) framework. Then, we introduce the adaptive memory mechanism, hybridizing it with GE-HH.

A. The Proposed Framework

It is well established that the efficiency of any problem solver relies on its ability to explore regions of the search space, which is strongly influenced by its structures and parameter values [7],[10],[12]. Therefore, the performance of any search methodology can potentially be enhanced by automatically adjusting its structures and/or parameter values. In this work, we propose a grammatical evolution hyper-heuristic (GE-HH) framework that generates a different local search template (problem solver) to suit the given problem instance. The proposed framework takes several basic heuristic components as input and generates a local search template by combining these basic components. The process of combining heuristic components will be carried out automatically. Thus, the benefit of this framework is not only to generate different local search templates by combining basic heuristic components, but also to discover new kinds of heuristics, without relying on human interference.

As we mentioned earlier (Section III), there are three essential procedures of grammatical evolution algorithm: a grammar, a search engine and a mapper function. Our search engine (genetic algorithm), and the mapper function are implemented as in the original algorithm [30]. The BNF grammar, which is problem dependent, must be defined in order to suit the problem at hand. Generally, the design of the BNF grammar, which decides which production rule will be selected, has a significant impact on the output, i.e. the programs. In our GE-HH framework, the basic heuristic components are represented by BNF. To design a complete BNF grammar one needs to carry out the following steps [30]:

- Determine the terminals, non-terminals and starting symbol.
- Design the BNF syntax which may have problem specific function(s).

In this work, three different heuristic components (acceptance criteria (*Ac*), neighborhood structures (*Ns*) and neighborhood combinations (*Nc*)) are used as basic elements of the BNF grammar. We have selected these three components because they are recognized as crucial components in designing problem solvers [3],[18]. These are explained as follows:

1. The acceptance criteria (*Ac*) decides whether to accept or reject a solution. A number of acceptance criteria have been proposed in the literature and each one has its own strengths and weaknesses. The strength of one acceptance criterion can compensate for the weakness of another if they can be integrated into one framework. In this work, we have employed several acceptance criteria. The acceptance criteria that are used in our GE-HH framework have been widely used in the literature [3],[6],[18],[29], and are presented below.

Ac	Description
IO	**Improving or equal only**: The generated solution is accepted if the objective value is equal or better than the previous one. The local search template that uses this acceptance criterion will be executed for a pre-defined number of iterations. In this work, we have experimentally set the pre-defined number of iterations

	to100 non-improvement iterations [18].
AM	**All Moves**: All generated solutions are accepted without taking into consideration their quality. This criterion can be seen as a mutational operator which aims to diversify the search. The local search template that uses this acceptance criterion will be run for a pre-defined number of iterations. In this work, we have experimentally set the pre-defined number of iterations to 50[18].
SA	**Simulated Annealing**: A move to a neighbor of the current solution is always accepted if it improves (or is equal to) the current objective value. However, non-improving moves are accepted based on a probability acceptance function, $R<\exp(-\delta/t)$, where R is a random number between [0, 1] and δ is the change in the objective value. The ratio of accepted moves to worse solutions is controlled by a temperature t which gradually decreases by β during the search process. In this work, $\beta= 0.85$ and the initial temperature t is 50% of the value of the initial solution, as suggested in [32],[33]. The local search template that uses the SA acceptance criteria is terminated when $t= 0$.
EMC	**Exponential Monte Carlo**: Improving solutions are always accepted. Worse solutions are accepted with a probability of $R<\exp(-\delta)$, where R is a random number between [0, 1] and δ is the change in the objective value. The probability of accepting worse solutions will decrease as δ increases [34]. The local search template that uses this acceptance criterion will be run for a pre-defined number of iterations. In this work, we have experimentally set the pre-defined number of iterations to 100.
RR	**Record-to-Record Travel**: A move to a neighbor solution is always accepted if it improves (or is equal to) the current objective value. Worse solutions are accepted if the objective value is less than $R+D$, where R is the value of the initial solution and D is a deviation. In this work, we set $D= 0.03$ and R is updated every iteration to equal the current solution. The local search template that uses the RR acceptance criteria is repeated until the stopping condition is met, set to 100 iterations [3].
GD	**Great Deluge**: Improving solutions are always accepted. A non-improving solution is accepted if its objective value is less than the *level* initially set to the value of the initial solution. The value of *level* is gradually decreased by β. β is calculated by β = (*f*(*initial solutions*) - estimated(*lower bound*) / *number of iterations*). In this work, we set the *number of iterations* to 1000. The local search template that uses the great deluge acceptance criteria will terminate when the *level* is equal to, or less than, the best known solution found so far [3],[33].
NV	**Naive acceptance**: accepts all improving moves. Non improving moves are accepted with 50% probability. The local search template that uses this acceptance criterion is executed for a pre-defined number of iterations (100 iterations) [35].
AA	**Adaptive Acceptance**: accepts all improving moves. Non improving moves are accepted according to an *acceptance Rate*, which is updated during the search. Initially, *acceptance Rate* is set to zero. However, if the solutions cannot be improved for a certain number of non improvement iterations (i.e. 10 consecutive non improvement iterations), then *acceptance Rate* is increased by 5%. Whenever a solution is accepted, *acceptance Rate* is reduced by 5%. The local search template that uses this acceptance criterion will be run for a pre-defined number of iterations, experimentally set in this work as 100 iterations [35].

2. The second heuristic component that is used in our GE-HH framework are the neighborhoods structures (*Ns*) or move operators. The aim of any neighborhood structure is to explore the neighbor of current solutions or to generate a neighborhood solution. The neighborhood solution is generated by performing a small perturbation or changing some attribute(s) of the current solution. The neighborhood structures are critical in the design of any local search method [36]. Traditionally, each neighborhood structure has its own characteristics (weaknesses and strengths), thus, several types of neighborhood structures may be needed to cope with changes in the problem landscape as the search progresses. In this work, we have employed several neighborhoods which are problem dependent. The descriptions of the neighborhood structures that have been used in our work, which are different from one domain to another, are presented in problem description sections (see Sections V-B4 and V-C4).
3. The third heuristic component employed in our framework is the neighborhood combinations/operators (*Nc*). The aim of the neighborhood combinations/operators is to combine the strength of two or more neighborhood structures into one structure. Such combination has been shown to be very efficient in solving many optimization problems [37]. The benefit of such an idea was first demonstrated using strategic oscillation in tabu search [38]. Recently, Lu et al. [37] conducted a comprehensive analysis to assess the performance of neighborhood combinations within several local search methods (tabu search, iterated local search and steepest decent algorithm) in solving university course timetabling problems. Their aim was to answer why some neighborhood structures can produce better results than others and what characteristics constitute a good neighborhood structure. They concluded that the use of neighborhood combinations can dramatically improve local search performance. Other works which have also studied the benefit of using neighborhood combinations include [39],[40],[41]. In this work, three kinds of neighborhood combinations/operators are used [37],[40],[18], which are described below.

Nc	Description
+	**Neighborhood Union**: involves the moves that can be generated by using two or more different neighborhoods structures. For example, consider two different neighborhoods N_1 and N_2, which can be represented as $N_1 \cup N_2$ or N_1+N_2, then the union move includes the solution that can be obtained by consecutively applying N_1 followed by N_2 then calling the acceptance criterion to decide whether to accept or reject the generated solution. Besides combining the strength of different neighborhoods [37], when the search space is highly disconnected, such a combination might help escape from disconnected search spaces, that may not happen when using N_1 alone. For example, in exam timetabling, the single move neighborhood structure which moves one exam from one timeslot to another one might lead the search to a disconnected search space when all exams which clash with another exam in every other timeslot often cannot be moved at all [42]. Thus, combining a single move neighborhood with another neighborhood i.e. swap two exams, can help to find a clash free timeslot for the selected exam to be moved to. The same issue can also be observed in capacitated vehicle routing problems when using a single move neighborhood that moves a customer from one route to another.
➡	**Random Gradient:** A neighborhood structure is repeatedly applied until no improvement is possible. This is followed by applying other neighborhood structures. For example, consider two different neighborhoods; N_1 and N_2 are random gradient operators which can be represented as $N_1 \Rightarrow N_2$. The local search template will keep applying N_1 as long as the generated solution is accepted by the local search acceptance criteria. When no improvement is possible the local search template stops applying N_1 and restarts from the local optimum obtained by N_1, but with neighborhood N_2 [6],[18].
T-R-S	**Token-Ring Search**: The neighborhood structures of the generated template are consecutively applied one after another until the end of sequence. When the generated template moves to the next neighborhood structure in the sequence, it restarts from the local

	optimum obtained by the previous neighborhood structure. If the generated template reaches the end of the sequence, it restarts the search from the first neighborhood in the sequence using the local optimum obtained by the last neighborhood structure in the sequence [37],[40],[43]. In this work, the token-ring search is set as a default in all generated local search template (there is no special symbol for it in the BNF grammar). Note that if there is no operator between neighborhood structures e.g. $N_1\ N_2$, each neighborhood is applied only one time. For example, if we have $N_1\ N_2\ N_3$ the local search template will apply N_1 one time only, and then move to N_2 which will also be applied once, and then move to N_3. This is because there is no combination operator between these sequences of neighborhood structures.

After determining the basic elements of the BNF grammar, we now need to specify the starting symbol (*S*), terminals (*T*), non-terminals (*N*) and the production rules (*P*) that will represent the heuristic components. These are as follows:

Objective	Symbols	Description
starting symbol (S)	*LST*	*Local Search Template*
non-terminal (N)	*Ac*	*Acceptance Criteria*
	Lc	*LST Configurations*
	Ns	*Neighborhood Structures*
	Nc	*Neighborhood Combinations*
terminal (T)	*IO*	*Improving Only or equal*
	AM	*All Moves*
	SA	*Simulated Annealing*
	EMC	*Exponential Monte Carlo*
	RR	*Record-to-Record Travel*
	GD	*Great Deluge*
	NA	*Naive Acceptance*
	AA	*Adaptive Acceptance*
	+	*Neighborhood Union*
	➡	*Random Gradient*
	Nb_1	*First neighborhood e.g. 2-opt*
	Nb_2	*Second neighborhood e.g. Swap*
	.	.
	.	.
	Nb_n	*Neighborhood n*
production rules (P)	(1) <*LST*>::= *AcLc* (0)	*Starting symbol rule. Number of choices available for LST =0*
	(2) <*Ac*>::= *IO* (0) |*AM* (1) |*SA* (2) |*EMC* (3) | *RR* (4) | *GD* (5) | *NA* (6) | *AA* (7)	*Acceptance Criteria production rules* *Number of choices available for Ac =8*
	(3) <*Lc*>::= *NsLc* (0) | *NsNcNs* (1) | *NsNsLc* (2) | *NcNsNs* (3) | *NsNsNcNs* (4) | *Lc* (5)	*LST Configurations production rules.* *Number of choices available for Lc =6*
	(4) <*Ns*>::= *Nb1* (0) | *Nb2* (1) | . (2) | . (3) | . (4) | . (5) | . (6) | *Nbn* (*n*)	*Neighborhood structures production rules.* *Number of choices available for Nb =1 to n* *Note that n represent the number of neighborhood structures that are used for each problem domain (see SectionsV-B4 and V-C4).*
	(5) <*Nc*>::= + (0) | ➡ (1)	*Neighborhoods combination production rules.* *Number of choices available for Nc =2*

The above BNF grammar is valid for every local search template (LST) for both problem domains in the work. This is because each local search template (LST) has different rules and characteristics. Finding the best BNF grammar for every local search template (LST) would be problem dependent, if not problem instance dependent. Please note that not all local search templates will improve the solution because the employed acceptance criteria might accept worse solutions with a certain probability. For example, the local search that uses all moves acceptance criterion (AM)

will accept any solution that does not violate any hard constraints regardless of its quality.

The programs in our GE-HH represent local search templates or problem solvers. The local search template starts with an initial solution and then iteratively improves it. The initial solution can be randomly generated or by using heuristic methods (see Sections V-B3 and V-C3). Please note that the initial solution generation method is not a part of the GE-HH. In this work, we use two fitness functions. The first one, penalty cost, is problem dependent, and is used by the inner loop of the generated local search template in deciding whether to accept or reject the perturbed solution (see Sections V-B and V-C for more details about the penalty cost). The second fitness function is problem independent and it measures the quality of the generated program (local search template) after executing it. At every iteration, if the generated programs are syntactically correct (all non-terminals can be converted into terminals), the programs are executed and their fitness is computed from their output. In this work, the fitness function of the generated programs is calculated as a percentage of improvement (PI). Assume $f_1$is the fitness of the initial solution and f_2 is the fitness of the solution after executing the generated programs, then PI= | (f_1-f_2)/ f_1| * 100, if f_2<= f_1. If $f_2 > f_1$ discard the generated program.

With all the GE-HH elements (grammar, search engine, mapper procedure and fitness function) defined, the proposed GE-HH framework is carried out as depicted in Fig.6.

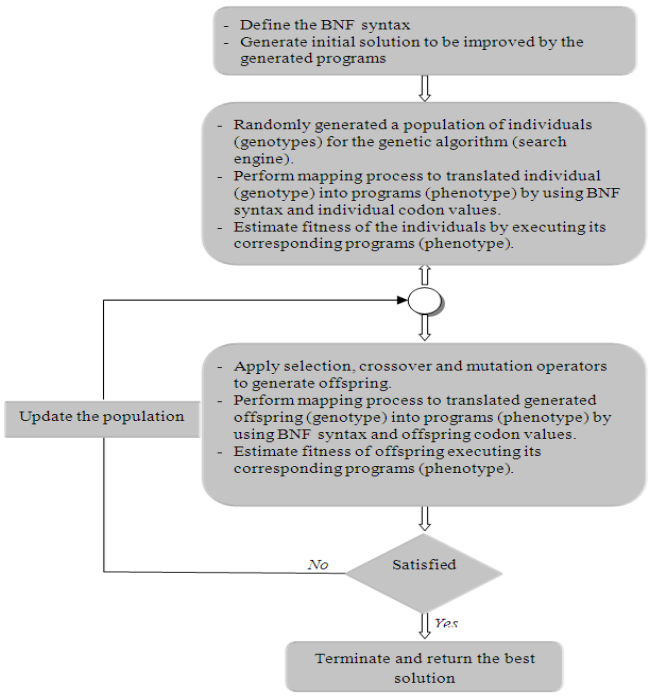


Fig.6.The proposed GE-HH framework

B. Hybrid Grammatical Evolution Hyper-heuristic and Adaptive Memory Mechanism

Traditionally, previous hyper-heuristic frameworks that have been proposed in the literature operate on a single solution [6],[18],[29]. Single solution based perturbative hyper-heuristics start with an initial solution and iteratively move from the current solution to another one by applying an operator such as 2-opt. Although single solution based methods have been widely used to solve several kinds of problems, it is accepted that pure single solution based methods are not well suited to fine tuning for large search spaces and heavily constrained problems [44],[45]. As a result, single solution based methods have been hybridized with other techniques to improve their efficiency [45]. Generally, it is widely believed that a good search methodology must have the ability of exploiting and exploring different regions of the solution search space rather than focusing on a particular region. That is, we must address the problem of exploitation vs. diversification, which is a key feature in designing efficient search methodologies [44].

In order to enhance the efficiency of the GE-HH framework and to diversify the search process, we hybridize it with an adaptive memory mechanism. This method has been widely used with several meta-heuristic algorithms such as tabu search, ant colonies, genetic algorithms and scatter search [46]. The main idea is to enhance the diversification by maintaining a population of solutions. For example, the reference set in scatter search [46] which includes a collection of both high quality and diverse solutions.

In this work, the adaptive memory mechanism (following the approach in [47],[48]) contains a collection of both high quality and diverse solutions, which are updated as the algorithm progresses. The size of the memory is fixed (equal to the number of acceptance criteria, which is 8). Our adaptive memory works as follows:

- Generate a set of diverse solutions. The set of solutions can be generated randomly or by using a heuristic method. In this work, the solutions are generated using a heuristic method (see SectionsV-B3 and V-C3).
- For each solution, associate a frequency matrix which will be used to measure solution diversity. The frequency matrix saves the frequency of assigning an object (exam or customer) to the same location. For example, in exam timetabling, the frequency matrix stores how many times the exam has been assigned to the same timeslot. Whilst, in the capacitated vehicle routing problem, it stores how many times a customer has been assigned to the same route. 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) and there are five available locations (represented by columns). The solution on the left of Fig.7 can be read as follows: object1 is assigned to location 1, object 2 is assigned to location 3, etc. The frequency matrix on the right side of the 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

Objects	1	2	3	4	5	
1	1	0	0	0	0	
2	0	0	1	0	0	
3	0	0	0	0	1	
4	0	0	0	1	0	
5	0	1	0	0	0	

solution

Objects	1	2	3	4	5
1	2	3	1	4	1
2	1	1	1	2	2
3	2	2	2	2	1
4	2	1	3	1	1
5	2	1	2	1	3

frequency matrix

Fig.7. Solution and it is corresponding frequency matrix.

- If any solution is improved by the GE-HH framework, we update the frequency matrix.
- Calculate the quality and the diversity of the improved solution. In this work, the quality represents the *penalty cost* which calculates the number of soft constraint violations (see Sections V-B and V-C). The diversity is measured using entropy information theory (1), (2) as follows [47],[48]:

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

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

Where

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

- Add the new solution to the adaptive memory by considering the solution quality and diversity.

Fig.8 shows the hybrid GE-HH framework with an adaptive memory mechanism. Algorithm 1 presents the pseudo-code of GE-HH.

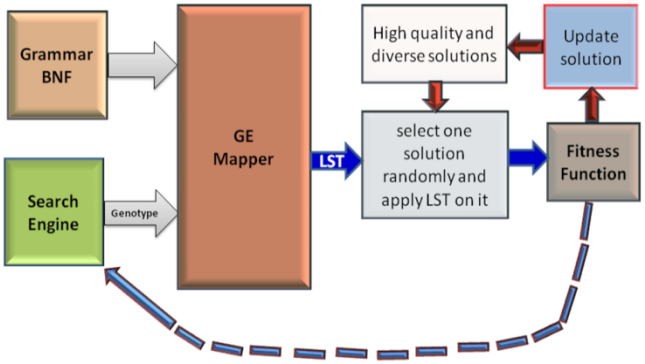


Fig.8.Hybrid grammatical hyper-heuristic framework and adaptive memory mechanism

The algorithm starts by generating a set of initial solutions for the adaptive memory mechanism (see SectionsV-B3 and V-C3) and defining the BNF grammar (see Section IV-A). It then initializes the genetic algorithm parameters and creates a population of solutions by assigning a random value between 0 and 255 for each chromosome gene (codons) [30].

For each solution (chromosome) in the population, the corresponding program is generated by invoking the mapping function. In order to ensure that there is no duplication in the generated program (i.e. the program does not have two consecutive operators) the program is checked by the *edit* function. For example, if the generated program is *SA: N1N2++N2+N4*, with consecutive ++ operators, the *edit* function will remove one of the + operators and the program will be *SA: N1N2+N2+N4*. One solution from the adaptive memory mechanism is then selected, to which the generated programs are applied. The adaptive memory is then updated.

Subsequently, the genetic algorithm is executed for a pre-defined number of generations. At every generation, offspring are generated by applying selection, crossover and mutation. The generated offspring (programs) are then executed. If the offspring is better than the worst chromosome, it is added to the population and the adaptive memory mechanism is updated.

Algorithm 1: Pseudo-code of grammatical evolution hyper-heuristic framework

Initialization step

Generate a set of initial solutions and initialize the adaptive memory, $adaptive_{memory}$
Defined the BNFgrammar, $BNF_{grammar}$

Generate initial population

Set number of generations, $population_{size}$, $chromosom_{numbits}$, $p_{crossover}$, $p_{muatation}$
$population \leftarrow initializepopulation(population_{size}, chromosom_{numbits})$
foreach $sol_i \in population$ **do**
  $sol_{i\text{-}integer} \leftarrow convert\ (chromosom_{numbits})$
  $sol_{i\text{-}program} \leftarrow map\ (BNF_{grammar}, sol_{i\text{-}integer})$
  $edit(sol_{i\text{-}program})$
  $initial_{sol} \leftarrow selectsoltuion(adaptive_{memory})$
  $sol_{i\text{-}cost} \leftarrow execute\ (sol_{i\text{-}program}, initial_{sol})$
  update $adaptive_{memory}$
end
while not stopping condition () do

selection

  $parent_i \leftarrow SelectParents(population_{size})$
  $parent_j \leftarrow SelectParents(population_{size})$

crossover

  $child_1 \leftarrow Crossover\ (parent_i, parent_j, p_{crossover})$
  $child_2 \leftarrow Crossover\ (parent_i, parent_j, p_{crossover})$

mutation

  $child_{1m} \leftarrow Mutation\ (child_1, p_{muatation})$
  $child_{2m} \leftarrow Mutation\ (child_2, p_{muatation})$

converting

  $child_{1m\text{-}integer} \leftarrow convert\ (child_{1m})$
  $child_{2m\text{-}integer} \leftarrow convert\ (child_{2m})$

mapping

  $child_{1m\text{-}program} \leftarrow map\ (child_{1m\text{-}integer}, BNF_{grammar})$
  $edit(child_{1m\text{-}program})$
  $child_{2m\text{-}program} \leftarrow map\ (child_{2m\text{-}integer}, BNF_{grammar})$
  $edit(child_{2m\text{-}program})$

executing

  $initial_{sol} \leftarrow selectsoltuion(adaptive_{memory})$
  $child_{1m\text{-}cost} \leftarrow execute\ (child_{1m\text{-}program}, initial_{sol})$
  $child_{2m\text{-}cost} \leftarrow execute\ (child_{2m\text{-}program}, initial_{sol})$

updating

  $population \leftarrow populationUpdate(child_1, child_2)$
  update $adaptive_{memory}$
end
return the best solution

V. Experimental Results

In this section, we evaluate and compare the proposed GE-HH with the state of the art of hyper-heuristics, and other search methodologies.

A. GE-HH Parameters Setting

In order to find appropriate parameter values for GE-HH, we utilize the Relevance Estimation and Value Calibration method (REVAC) [49]. REVAC is a steady state genetic algorithm that uses entropy theory to determine the parameter values for algorithms. Our aim is not to find the optimal parameter values for each domain, but to find generic values that can be used for both domains. To use the same parameter settings across instances of both domains, we tuned GE-HH for each domain separately and then used the average of them in value obtained by REVAC for all tested instances. In order to have a reasonable trade-off between solution quality and the computational time needed to reach good quality solutions, the execution time for each instance is fixed to 20 seconds. The number of iterations performed by REVAC is fixed at 100 iterations (see [49] for more details). For each domain, the average values over all tested instances for each parameter are recorded. Then, the average values over all parameters are set as the generic values for GE-HH. The parameter settings of GE-HH that have been used for both domains are listed in Table 2.

TABLE 2 GE-HH PARAMETERS

Parameters	Value
Population Size	100
Number of Generations	20
One point Crossover Probability	0.8
Point Mutation Probability	0.01
Chromosome Length	60
Probability of Swapping	0.01
Probability of Duplication	0.01
Maximum number of Wraps	5
Selection Mechanism	Roulette Wheel
Generational Model	Steady State

B. Problem Domain I: Exam Timetabling Problems

Exam timetabling is a well known NP-hard combinatorial optimization problem [50] and is faced by all academic institutions. The exam timetabling problem can be defined as the process of allocating a set of exams into a limited number of timeslots and rooms so as not to violate any hard constraints and to minimize soft constraint violations as much as possible[51]. In this work, we carried out experiments on the most widely used un-capacitated Carter benchmarks (Toronto *b* type I in [51]) and also on the recently introduced exam timetable dataset from the 2007 International Timetabling Competition, ITC 2007 [15].

1) Test Set I: Carter Uncapacitated Datasets

The Carter datasets have been widely used in the scientific literature[14],[51]. They are un-capacitated exam timetabling problems where room capacities are ignored. The constraints are shown in Table 3.

TABLE 3 CARTER HARD AND SOFT CONSTRAINTS

Symbols	Description
	Hard Constraints
$H1_{Carter}$:	*No student can sit more than one exam at the same time.*
	Soft Constraints
$S1_{Carter}$:	*Conflicting exams (with common enrolled students) should be spread as far apart as possible to allow sufficient revision time between exams for students.*

The quality of a timetable is measured based on how well the soft constraints have been satisfied. The proximity cost is used to calculate the penalty cost (equation 3) [14].

$$Soft_{carter} = \sum_{k=1}^{m-1} \sum_{l=k+1}^{m} (w_i \times s_{kl}) / S, \quad i \in \{0,1,2,3,4\} \quad \ldots (3)$$

Where:

- $w_i=2^{|4-i|}$ is the cost of scheduling two conflicting exams e_l and e_k (which have common enrolled students) with i timeslots apart, if $i=|t_l-t_k|<5$, i.e. $w_0=16$, $w_1=8$, $w_2=4$, $w_3=2$ and $w_4=1$; t_l and t_k as the timeslot of exam e_l and e_k, respectively.
- s_{kl} is the number of students taking both exams e_k and e_l, if $i=|t_l-t_k| <5$;
- m is the number of exams in the problem
- S is the number of students in the problems

Table 4 gives the characteristics of the un-capacitated exam timetabling benchmark problem (Toronto *b* type I in [51]) which comprises 13 real-world derived instances.

TABLE 4 Carter's un-capacitated benchmark exam timetabling dataset

Datasets	No. of timeslots	No. of exams	No. of Students	Conflict Density
Car-f-92-I	32	543	18419	0.14
Car-s-91-I	35	682	16925	0.13
Ear-f-83-I	24	190	1125	0.27
Hec-s-92-I	18	81	2823	0.42
Kfu-s-93	20	461	5349	0.06
Lse-f-91	18	381	2726	0.06
Pur-s-93-I	43	2419	30032	0.03
Rye-s-93	23	486	11483	0.07
Sta-f-83-I	13	139	611	0.14
Tre-s-92	23	261	4360	0.18
Uta-s-92-I	35	622	21267	0.13
Ute-s-92	10	184	2750	0.08
Yor-f-83-I	21	181	941	0.29

Note: conflict density = number of conflicts / (number of exams)^2

2) Test Set II: ITC 2007 Datasets

The second dataset was introduced in the second International Timetabling Competition, ITC 2007, aiming to facilitate a better understanding of real world timetabling problems and to reduce the gap between research and practice [15]. It is a capacitated problem and has several hard and soft constraints (see Tables 5&6, respectively).

The objective function from [15] is used (see equation 4). The ITC 2007 problem has 8 instances. Table 7shows the main characteristics of these instances.

$$Soft_{ITC\,2007} = \sum_{s \in S} (w^{2R} S_S^{2R} + w^{2D} S_S^{2D} + w^{PS} S_S^{PS}) + w^{NMD} S^{2NMD} + w^{FL} S^{FL} + w^{P} S^{P} + w^{R} S^{R} \quad \ldots (4)$$

TABLE 5 ITC 2007 HARD CONSTRAINTS

Symbols	*Description*
$H1_{ITC2007}$:	*No student can sit more than one exam at the same time.*
$H2_{ITC2007}$:	*There must be a sufficient number of seats to accommodate the exams being scheduled in a given room.*
$H3_{ITC2007}$:	*The length of exams assigned to each timeslot should not violate the timeslot length.*
$H4_{ITC2007}$:	*Some sequences of exams have to be satisfied. e.g. Exam_B must be scheduled after Exam_E.*
$H5_{ITC2007}$:	*Room related hard constraints must be respected e.g. Exam_B must be scheduled in Room 3.*

TABLE 6 ITC 2007 SOFT CONSTRAINTS

Symbols	*Mathematical Symbols*	*Description*
$S1_{ITC2007}$:	S_s^{2R}	*Two exams in a row: Minimize the number of students that have consecutive exams in a row.*
$S2_{ITC2007}$:	S_s^{2D}	*Two exams in a day: Student should not be assigned to sit more than two exams in a day. Of course, this constraint only becomes important when there are more than two exam periods in the same day.*
$S3_{ITC2007}$:	S_s^{PS}	*Exams spread: Conflicting exams should be spread as far apart as possible to allow sufficient revision time between exams for students.*
$S4_{ITC2007}$:	S_s^{2NMD}	*Mixed durations: Minimize exams that have different durations but assigned into the same timeslot and room.*
$S5_{ITC2007}$:	S^{FL}	*Larger exams: Minimize the number of exams of large size that appear later in the exam timetable.*
$S6_{ITC2007}$:	S^{P}	*Period Penalty: Some periods have an associated penalty. Minimize the number of exams assigned into these periods.*
$S7_{ITC2007}$:	S^{R}	*Room Penalty: Some rooms have an associated penalty; Minimize the number of exams allocated in penalized rooms.*

TABLE 7 THE ITC 2007 BENCHMARK EXAM TIMETABLING DATASETS

Datasets	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	**A9**	**A10**	**A11**	**A12**	**A13**
Dataset 1	7891	607	54	7	5	7	5	10	100	30	5	7833	5.05
Dataset 2	12743	870	40	49	5	15	1	25	250	30	5	12484	1.17
Dataset 3	16439	934	36	48	10	15	4	20	200	20	10	16365	2.62
Dataset 4	5045	273	21	1	5	9	2	10	50	10	5	4421	15.0
Dataset 5	9253	1018	42	3	15	40	5	0	250	30	10	8719	0.87
Dataset 6	7909	242	16	8	5	20	20	25	25	30	15	7909	6.16
Dataset 7	14676	1096	80	15	5	25	10	15	250	30	10	13795	1.93
Dataset 8	7718	598	80	8	0	150	15	25	250	30	5	7718	4.55

Note:			
A1:	*No. of students reported in [15].*	*A8:*	*No mixed duration penalty, S^{NMD}*
A2:	*Number of exams.*	*A9:*	*Number of largest exams, S^{FL}*
A3:	*Number of timeslots.*	*A10:*	*Number of last timeslots to avoid, S^{P}*
A4:	*Number of rooms.*	*A11:*	*Front load penalty, S^{R}, soft constraints weight[15]*
A5:	*Two in a day penalty, S^{2D}*	*A12:*	*Number of actual students in the datasets.*
A6:	*Two in a row penalty, S^{2R}*	*A13:*	*Conflict density*
A7:	*Timeslots spread penalty, S^{PS}*		

3) Problem Domain I: Initial Solutions

As mentioned in Section IV-A, GE-HH starts by initializing the adaptive memory mechanism which contains a population of solutions. In this work, we employ hybrid graph coloring heuristics [52] to generate an initial population of feasible solutions for both the Carter and the ITC 2007 instances. The three graph coloring heuristics we utilize are:

- Least Saturation Degree First (SD): exams are ordered dynamically, in an ascending order, by the number of remaining timeslots.
- Largest Degree First (LD): exams are ordered, in a decreasing order, by the number of conflicts they have with all other exams.
- Largest Enrolment First (LE): exams are ordered by the number of students enrolled, in decreasing order.

The solution construction method starts with an empty timetable and applies the hybridized heuristics to select and assign the unscheduled exams one by one until all exams have been scheduled. To select an exam, the hybridized heuristic (SD+LD+LE) firstly sorts the unscheduled exams in a non-decreasing order of the number of available timeslots (SD). Those with equal SD evaluations are then arranged in a non-increasing order of the number of conflicts they have with other exams (LD) and those with equal LD evaluations are then arranged in a non-increasing order of the number of student enrolments (LE). The first exam in the final order is assigned to the timetable. We assign exams to a random timeslot when it has no conflict with those that have already been scheduled (in case of ITC 2007, an exam is assigned to best fit a room), ensuring that all hard constraints are satisfied. If some exams cannot be assigned to any available timeslot, we stop the process and start again. Although there is no guarantee that a feasible solution can be generated, for all the instances used in this work, we were always able to obtain a feasible solution.

4) Problem Domain I: Neighborhood Structures

The neighborhood structures that we employed in the GE-HH framework for both Carter and ITC 2007, which are commonly used in the literature [42], are as follows:

Nbe1: *Select one exam at random and move it to any feasible timeslot-room.*

Nbe2: *Select two exams at random and swap their timeslots (if feasible).*
Nbe3: *Select two timeslots at random and swap all their exams.*
Nbe4: *Select three exams at random and exchanges their timeslots at random (if feasible).*
Nbe5: *Move the exam causing the highest soft constraint violation to any feasible timeslot.*
Nbe6: *Select two exams at random and move them to another random feasible timeslots.*
Nbe7: *Select one exam at random, select a timeslot at random (distinct from the one that was assigned to the selected exam) and then apply the Kempe chain neighborhood operator.*
Nbe8: *Select one exam at random, select a room at random (distinct from the one that was assigned to the selected exam) and then move the exam to the room (if feasible).*
Nbe9: *Select two exams at random and swap their rooms (if feasible).*

Note that neighborhoods *Nbe8* and *Nbe9* are applied to ITC 2007 datasets only because they consider rooms. The neighborhood solution is accepted if it does not violate any hard constraints. Thus, the search space of GE-HH is limited to feasible solutions only.

C. Problem Domain II: Capacitated Vehicle Routing Problems

The capacitated vehicle routing problem (CVRP) is a well-known challenging combinatorial optimization problem [53]. The CVRP can be defined as the process of designing a least cost set of routes to serve a set of customers [53]. In this work, we test GE-HH on two sets of benchmark capacitated vehicle routing problem datasets. These are the 14 instances introduced by Christofides [16] and 20 large scale instances introduced by Golden [17]. The CVRP can be represented as an undirected graph G (V, E), where V= $\{v_0, v_1 \ldots v_n\}$ is a set of vertices which represents a set of fixed locations (customers) and E= $\{(v_i, v_j): v_i, v_j \in V, i<j\}$ represents the arc between locations (customers). E is associated with non-negative costs or travel time defined by matrix C= (c_{ij}), where c_{ij} represents the travel distance between customers v_i and v_j. Vertex $v_0$represents the depot which is associated with m vehicles of capacity $Q_1 \ldots Q_m$ to start their routes $R_1 \ldots R_m$. The remaining vertices $v_1 \ldots v_n$ represent the set of customers and each customer requests$q_1 \ldots q_n$ goods and serving time δ_i. The aim is to find a set of tours that do not violate any hard constraints and minimize the distance. The hard constraints that must be respected are:

- 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
- The duration of each route does not exceed a global upper bound.

The cost of each route is calculated using (5) [53]:

$$C(R_i) = \sum_{j=1}^{n} c_{ij} + \sum_{j=0}^{n} \delta_i \quad \ldots\ldots\ldots (5)$$

and the cost for one solution is calculated using (6):

$$f = \sum_{i=1}^{m} C(R_i) \quad \ldots\ldots\ldots (6)$$

The two sets of benchmark problems that we have considered in this work have similar constraints and objective function. However, the complexity, instance sizes and customer distributions are different from one set to another.

1) Test Set I: Christofides Datasets

The first set comprises of 14 instances and was introduced by Christofides [16]. The main characteristics of the problem are summarized in Table 8. The instance size varies from 51 to 200 customers, including the depot. Each instance has a capacity constraint. Instances 6-10, 13 and 14 also have a maximum route length restriction and non-zero service times. The problem instances can be divided into two types: in instances 1-10, the customers are randomly located, whilst, in instances 11-14 the customers are in clusters.

TABLE 8 CHRISTOFIDES INSTANCES

Datasets	Customers	Capacity	Max. tour length	Service time	No. of vehicles
1	51	160	∞	0	5
2	76	140	∞	0	10
3	101	200	∞	0	8
4	151	200	∞	0	12
5	200	200	∞	0	17
6	51	160	200	10	6
7	76	140	160	10	11
8	101	200	230	10	9
9	151	200	200	10	14
10	200	200	200	10	18
11	121	200	∞	0	7
12	101	200	∞	0	10
13	121	200	720	50	11
14	101	200	1040	90	11

2) Test Set II: Golden Datasets

The second CVRP dataset involves 20 large scale instances presented by Golden [17] (see Table 9). The instances have between 200 and 483 customers, including the depot. Instances 1-8 have route length restrictions.

TABLE 9 GOLDEN INSTANCES

Datasets	Customers	Capacity	Max. tour length	Service time	No. of vehicles
1	240	550	650	0	10
2	320	700	900	0	10
3	400	900	1200	0	10
4	480	1000	1600	0	12
5	200	900	1800	0	5
6	280	900	1500	0	8
7	360	900	1300	0	9
8	440	900	1200	0	11
9	255	1000	∞	0	14
10	323	1000	∞	0	16
11	399	1000	∞	0	18
12	483	1000	∞	0	19
13	252	1000	∞	0	27
14	320	1000	∞	0	30
15	396	1000	∞	0	34
16	480	1000	∞	0	38
17	240	200	∞	0	22
18	300	200	∞	0	28
19	360	200	∞	0	33
20	420	200	∞	0	41

3) Problem Domain II: Initial Solutions

For both the Christofides and the Golden instances, the initial population of feasible solutions is constructed utilizing the savings algorithm [54].

4) Problem Domain II: Neighborhoods Structures

The neighborhood structures that we employ in GE-HH for both the Christofides and the Golden instances are the most common ones used to solve the capacitated vehicle routing problems in the literature. They are as follows:

Nbv1: *Select one customer at random and move it to any feasible route.*
Nbv2: *Select two customers at random and swap their routes.*
Nbv3: *Select one route at random and reverse a part of a tour between two selected customers.*
Nbv4: *Select three customers at random and exchanges their routes at random.*
Nbv5: *Select one route at random and perform the 2-opt procedure.*
Nbv6: *Perform the 2-opt procedure on all routes.*
Nbv7: *Select two distinct routes at random and swap a portion of the first route with the first portion and second route.*
Nbv8: *Select two distinct routes at random and from each route select one customer. Swap the adjacent customer of the selected one for both routes.*
Nbv9: *Select two distinct routes at random and swap the first portion with the last portion.*
Nbv10 *Select one customer at random and move it to another position in the same route.*

The neighborhood solution is accepted if it does not break any hard constraints. Thus, the search space of GE-HH is limited to feasible solutions only.

VI. Computational Results and Comparison

To assess the benefit of incorporating an adaptive memory mechanism in GE-HH, for each domain, we have carried out two sets of experiments. The first one compares the performance of the grammatical evolution hyper-heuristic with an adaptive memory (GE-HH) and the grammatical evolution hyper-heuristic without an adaptive memory (GE-HH*) using the same parameter values and computational resources. The second test compares and analyses the performance of GE-HH against the state of the art of hyper-heuristics and bespoke methods. For both experimental tests, we report the *best*, *average*, *standard deviation* and *average time* over 51 independent runs with different random seeds. By executing 51 runs, instead of 50, we can easily calculate the median value without the need for interpolation. The aim of executing the proposed hyper-heuristic framework 51 runs is to get more information and to have a good indication regarding the algorithm consistency and generality, as it's highly recommended in the literature to have more than 30 runs in statistical analysis on algorithm performance [3]. The results represent the cost of soft constraint violations. In addition, we also report, for each instance, the percentage deviation from the best known value found in the literature, calculated as follows (7):

$$\Delta(\%) = \frac{best_{GE-HH} - best*}{best*}\% \quad \ldots\ldots\ldots (7)$$

Where $best_{GE\text{-}HH}$ is the best result obtained over 51 independent runs by GE-HH and *best** represents the best known value found in the literature.

We evaluate the performance of GE-HH by considering the following three criteria:

- **Generality**: We define generality as the ability of GE-HH to work well, not only across different instances of the same problem, but also across two different problem domains.
- **Consistency**: This is the ability of GE-HH to produce stable results when executed several times for every instance. Typically, consistency is one of the most important criteria in evaluating any algorithm. This is because many search algorithms have a stochastic component, which leads to different solutions over multiple runs even if the initial solution is the same. We measure the consistency of GE-HH based on the *average* and the *standard deviation* over 51 independent runs.
- **Efficiency**: This is the ability of GE-HH to produce good results that are close or better than the best known value in the literature. We measure the efficiency of GE-HH by reporting, for each instance, the *best* and the *percentage deviation*, see $\Delta(\%)$ in (7), from the best known results in the literature.

For all tested instances, except the ITC 2007 problem instances, we compare the GE-HH results with the state of the art in terms of solution quality rather than computational time. This is because the different computer resources researchers use which make the comparison difficult, if not impossible [39],[55]. Therefore, we set the number of generations as the termination criteria. As for the ITC 2007 datasets, the organizer provided benchmark software to determine the allowed execution time [15]. We have used this software to determine the execution time using our computer resources (i.e. 10 minutes). We have given extra time to GE-HH, due to the use of the adaptive memory (i.e. 10.83 minutes). As a result, the execution time of our method is within the range of those published in the literature.

A. Problems Domain I: Computational Results on Exam Timetabling Problems

1) Test Set I: Carter Uncapacitated Datasets

Table 10 lists, for each instance, the best, average, standard deviation and average time obtained by GE-HH and GE-HH*.

From Table 10, one can clearly see that GE-HH outperforms GE-HH* across all instances. Furthermore, both the best and average results obtained by GE-HH are better than GE-HH* on all instances. We can also see that in GE-HH, on twelve of the thirteen instances, the *standard deviation* is lower than GE-HH*. However, the computational time is different where GE-HH* is lower than GE-HH. This is mainly due to the use of population of solutions and diversity updating mechanism in the GE-HH

framework. The results reveal that the use of the adaptive memory mechanism has an effect on the ability of the GE-HH in producing good quality and consistent results over all instances.

We compare the performance of GE-HH against hyper-heuristics and other bespoke methods (see Table 11).

Table 12 shows the comparison of the best and average results of GE-HH and other hyper-heuristic methods. We also report, for each instance, the percentage deviation (Δ (%)) from the best result obtained by other hyper-heuristics and instance ranking. As can be seen from Table 12, GE-HH finds better solutions for 7 out of 13 instances compared to other hyper-heuristic methods and obtained the second best results for the other 5 instances (except Rye-s-93 which obtained third best results).

Table 13 presents, for all instances, the best, average, percentage deviation (Δ(%)) and instance ranking by GE-HH along with a comparison with respect to the best known results (shown in bold) in the literature obtained by bespoke methods. It can be seen that, even though GE-HH does not obtain the best solutions for all instances, over all, it obtains competitive results especially when considering the percentage deviation (Δ(%)) from the best known value found in the literature. If we consider an individual comparison, GE-HH is able to obtain better solutions on instances 8, 12, 11, 6, 7 and 2 compared to Mc_7, Mc_8, Mc_9, Mc_{10}, Mc_{11}, and Mc_{12}, respectively. Furthermore, only Mc_{10} reported results for Pur-s-93 and Rye-s-93 instances, Mc_7 and Mc_{11} reported result for Rye-s-93 instance (we suspect, due to the complexity and inconsistencies in these instances).

Results in Tables 12 and 13 demonstrate that, across all instances, GE-HH outperforms other hyper-heuristic methodologies and obtained competitive results compared to other bespoke methods. Except instance Ute-s-92 (ranked 6), the instance ranking varies between 2 to 4. Also, the percentage deviation indicates that GE-HH results are very close to the best known results. This demonstrates that GE-HH is able to generalize well over a set of problem instances rather than only producing good results for one or more of the problem instances.

TABLE 10 RESULTS OF GE-HH COMPARED TO GE-HH*

	GE-HH				***GE-HH****			
Instances	***Best***	***Average***	***Std***	***Time***	***Best***	***Average***	***Std***	***Time***
Car-f-92-I	**4.00**	**4.44**	**0.36**	200.2	4.12	4.73	0.48	170.18
Car-s-91-I	***4.62***	**4.87**	**0.17**	441.32	***4.62***	5.15	0.25	410.23
Ear-f-83-I	**34.71**	**36.50**	**0.71**	52.03	35.92	36.64	0.81	38.56
Hec-s-92-I	**10.68**	**11.57**	**0.54**	65.41	10.96	11.54	0.52	49.41
Kfu-s-93	**13.00**	***13.58***	***0.36***	92.22	13.06	***13.58***	***0.36***	76.17
Lse-f-91	**10.11**	**11.35**	0.91	58.11	10.21	11.36	**0.90**	45.37
Pur-s-93-I	**4.80**	**6.29**	**1.10**	610.07	6.31	7.41	1.68	580.16
Rye-s-93	**10.79**	**11.09**	**0.69**	546.66	11.00	12.10	0.85	495.11
Sta-f-83-I	**158.02**	**158.47**	**0.43**	32.24	158.21	159.52	0.76	25.04
Tre-s-92	**7.90**	**8.46**	**0.41**	93.17	7.96	8.49	0.83	81.28
Uta-s-92-I	**3.12**	**3.70**	**0.32**	189.24	3.18	3.72	0.41	168.19
Ute-s-92	**26.00**	**27.1**	**0.69**	48.11	26.02	27.15	0.78	40.30
Yor-f-83-I	**36.20**	**36.91**	**0.47**	181.25	36.20	36.93	0.56	95.08

Note: GE-HH: GE-HH employing adaptive memory mechanism. GE-HH: without using adaptive memory. The time represents average time in minutes. Best results in the literature are highlighted in bold. The bold italic indicates that both methods produce the same result.*

TABLE 11 ACRONYMS OF COMPARED METHODS

#	***Symbol***	***References***	
1	Mc_1	[56]	*Hyper-heuristics*
2	Mc_2	[57]	
3	Mc_3	[23]	
4	Mc_4	[58]	
5	Mc_5	[42]	
6	Mc_6	[59]	
7	Mc_7	[60]	*Bespoke methods*
8	Mc_8	[61]	
9	Mc_9	[62]	
10	Mc_{10}	[63]	
11	Mc_{11}	[64]	
12	Mc_{12}	[65]	

TABLE 12 RESULTS OF GE-HH COMPARED TO HYPER-HEURISTIC APPROACHES

	GE-HH					***Hyper-heuristic***					
Instances	***Best***	***Average***	**Δ (%)**	**Δ*(%)**	***Rank***	Mc_1	Mc_2	Mc_3	Mc_4	Mc_5	Mc_6
Car-f-92-I	**4.00**	4.44	*	8.29	1	4.52	4.53	4.16	4.28	4.1	4.26
Car-s-91-I	**4.62**	4.87	*	*	1	5.2	5.36	5.16	4.97	4.9	5.09
Ear-f-83-I	34.71	36.50	4.54	9.93	2	37.02	37.92	35.86	36.86	**33.2**	35.48
Hec-s-92-I	10.68	11.57	3.68	12.3	2	11.78	12.25	11.94	11.85	**10.3**	11.46
Kfu-s-93	**13.00**	13.58	*	2.87	1	15.81	15.2	14.79	14.62	13.2	14.68
Lse-f-91	**10.11**	11.35	*	9.13	1	12.09	11.33	11.15	11.14	10.4	11. 2
Pur-s-93-I	4.80	6.29	9.83	43.9	2	-	-	-	**4.37**	-	-
Rye-s-93	10.79	11.09	11.81	14.9	3	10.35	-	-	**9.65**	-	-

Sta-f-83-I	158.02	158.47	0.71	1.00	2	160.42	158.19	159.00	158.33	**156.9**	158.28
Tre-s-92	**7.90**	8.46	*	1.92	1	8.67	8.92	8.6	8.48	8.3	8.51
Uta-s-92-I	**3.12**	3.70	*	12.12	1	3.57	3.88	3.59	3.4	3.3	3.15
Ute-s-92	26.00	27.1	4.41	8.83	2	27.78	28.01	28.3	28.88	**24.9**	27.9
Yor-f-83-I	**36.20**	36.91	*	1.68	1	40.66	41.37	41.81	40.74	36.3	40.49
TP(13)	323.95	334.33				-	-	-	337.57	-	-
TP(12)	319.15	328.04				337.87	-	-	333.2	-	-
TP(11)	308.36	316.95				327.52	326.96	324.36	323.55	**305.8**	309.3

Note: TP(13): total penalty of 13 instances. TP(12): Total penalty of 12 datasets (excluding Pur-s-93-I). TP(11): Total penalty of 11 datasets (excluding Pur-s-93-I and Rye-s-93). "" means GE-HH result is better than other methods. "-"indicates no feasible solution has been found. Best results are highlighted in bold.*Δ*(%): the percentage deviation of the average value with regard to the best known results.*

TABLE 13 RESULTS OF GE-HH COMPARED TO BESPOKE METHODS

	GE-HH					***Bespoke methods***					
Instances	***Best***	***Average***	**Δ (%)**	**Δ* (%)**	***Rank***	Mc_7	Mc_8	Mc_9	Mc_{10}	Mc_{11}	Mc_{12}
Car-f-92-I	4.00	4.44	6.95	18.71	3	4.3	4.10	4.1	6.0	3.93	**3.74**
Car-s-91-I	4.62	4.87	4.52	10.18	3	5.1	4.65	4.8	6.6	4.50	**4.42**
Ear-f-83-I	34.71	36.50	18.46	24.57	4	35.1	37.05	36.0	**29.3**	33.7	32.76
Hec-s-92-I	10.68	11.57	16.08	25.76	3	10.6	11.54	10.8	**9.2**	10.83	10.15
Kfu-s-93	13.00	13.58	0.30	4.78	2	13.5	13.90	15.2	13.8	13.82	**12.96**
Lse-f-91	10.11	11.35	5.31	18.22	3	10.5	10.82	11.9	**9.6**	10.35	9.83
Pur-s-93-I	4.80	6.29	29.72	70.00	2	-	-	-	**3.7**	-	-
Rye-s-93	10.79	11.09	58.67	63.08	4	8.4	-	-	**6.8**	8.53	-
Sta-f-83-I	158.02	158.47	0.63	0.91	3	157.3	168.73	159.0	158.2	158.3	**157.03**
Tre-s-92	7.90	8.46	1.93	9.16	2	8.4	8.35	8.5	9.4	7.92	**7.75**
Uta-s-92-I	3.12	3.70	1.96	20.91	2	3.5	3.20	3.6	3.5	3.14	**3.06**
Ute-s-92	26.00	27.1	6.55	11.06	6	25.1	25.83	26.0	**24.4**	25.39	24.82
Yor-f-83-I	36.20	36.91	3.90	5.94	2	37.4	37.28	36.2	36.2	36.35	**34.84**
TP(13)	323.95	334.33				-	-	-	316.7	-	-
TP(12)	319.15	328.04				319.2	-	-	313.0	316.76	-
TP(11)	308.36	316.95				310.8	325.45	316.1	306.2	308.23	301.36

Note: TP(13): total penalty of 13 instances. TP(12): Total penalty of 12 instances(excluding Pur-s-93-I). TP(11): Total penalty of 11 instances(excluding Pur-s-93-I and Rye-s-93). "-"means no feasible solution has been found. Best results in the literature are highlighted in bold. Δ*(%): the percentage deviation of the average value with regard to the best known results.*

2) Test Set II: ITC 2007 Datasets

The first set of experiments presents a comparison between GE-HH and GE-HH* as well as the results of GE-HH without the extra computational time (GE-HH**), i.e. the computational time is fixed the same as GE-HH*. The best, average, standard deviation of the results and the average time are reported in Table 14. It can be seen that, across all instances, GE-HH outperforms GE-HH* and GE-HH** (in most cases), not only on solution quality, but also on the average and the standard deviation. Comparing the results of GE-HH* with GE-HH**, the results demonstrate that GE-HH** outperforms GE-HH* on five out of eight instances. The average and standard deviation of GE-HH** are better than GE-HH* for all tested instances. The results demonstrate the importance of incorporating the adaptive memory mechanism within GE-HH as well as implying that GE-HH is more general and consistent.

We now compare the performance of GE-HH with the best available results in the literature which are divided into two groups (see Table 15): ITC 2007 winners (Table 16) and Post-ITC 2007 (Table 17 hyper-heuristic and bespoke methods). In addition, we also included the results of GE-HH** in the comparison to assess its ability in producing good quality solutions compared to ITC 2007 winners as well as post ITC 2007 methods. It is clear from Tables 16 and 17 that GE-HH is the overall best. The presented results demonstrate that GE-HH not only generalizes well over a set of problem instances, but also produces much higher quality solutions. One can also see that GE-HH** outperformed the ITC 2007 winners on 7 instances and post ITC 2007 methods on 4 out of 8 tested instances (see Tables 16 and 17).

TABLE 14 RESULTS OF GE-HH COMPARED TO GE-HH* AND GE-HH**

	GE-HH				***GE-HH****				***GE-HH***			
Instances	***Best***	***Average***	***Std***	***Time***	***Best***	***Average***	***Std***	***Time***	***Best***	***Average***	***Std***	***Time***
Dataset 1	**4362**	**4394.10**	**29.18**	10.83	4370	4439.31	71.71	10	4370	4401.12	44.24	10
Dataset 2	**380**	**399.80**	**12.56**	10.83	395	413.17	22.33	10	380	405.12	13.94	10
Dataset 3	**8991**	**9072.35**	**112.06**	10.83	8998	9140.67	206.48	10	8995	9120.67	180.15	10
Dataset 4	**15094**	**15483.42**	**402.25**	10.83	15394	16433.71	996.42	10	15184	15824.87	564.74	10
Dataset 5	**2912**	**3010.15**	**28.298**	10.83	2990	3042.06	57.53	10	2993	3018.27	43.62	10
Dataset 6	**25735**	**25792.35**	**56.247**	10.83	25818	25930.17	294.57	10	25786	25860.24	94.28	10
Dataset 7	**4025**	**4062.85**	**45.74**	10.83	4037	4083.92	54.68	10	4041	4068.15	44.93	10
Dataset 8	**7452**	**7500.48**	**64.99**	10.83	7465	7525.77	78.01	10	7472	7581.10	63.85	10

Note: GE-HH: with the adaptive memory mechanism. GE-HH: without adaptive memory. GE-HH**: with adaptive memory but the computational time fixed same as GE-HH* (10 minutes). Times represent average time in minutes. Best results are highlighted in bold.*

TABLE 15 ACRONYMS OF COMPARED METHODS

#	Symbol	References	
1	$Mitc_1$	[66]	ITC 2007 winners
2	$Mitc_2$	[67]	
3	$Mitc_3$	[68]	
4	$Mitc_4$	[69]	
5	$Mitc_5$	[70]	
6	$Mitc_6$	[71]	Post-ITC 2007 HH
7	$Mitc_7$	[72]	
8	$Mitc_8$	[73]	Post-ITC 2007 NON-HH
9	$Mitc_9$	Post- Müller	
10	$Mitc_{10}$	[74]	

Note: HH: hyper-heuristic. NON-HH: bespoke methods

TABLE 16 RESULTS OF GE-HH AND GE-HH** ON THE ITC 2007 EXAM TIMETABLING DATASETS COMPARED TO ITC 2007 WINNERS

	GE-HH					GE-HH**	ITC 2007 Winners				
Instances	Best	Average	Δ (%)	Δ* (%)	Rank	Best	$Mitc_1$	$Mitc_2$	$Mitc_3$	$Mitc_4$	$Mitc_5$
Dataset 1	**4362**	4394.10	*	0.55	1	4370	4370	5905	8006	6670	12035
Dataset 2	**380**	399.80	*	*	1	380	400	1008	3470	623	3074
Dataset 3	**8991**	9072.35	*	*	1	8995	10049	13862	18622	-	15917
Dataset 4	**15094**	15483.42	*	*	1	15184	18141	18674	22559	-	23582
Dataset 5	**2912**	3010.15	*	0.74	1	2993	2988	4139	4714	3847	6860
Dataset 6	**25735**	25792.35	*	*	1	25786	26950	27640	29155	27815	32250
Dataset 7	**4025**	4062.85	*	*	1	4041	4213	6683	10473	5420	17666
Dataset 8	**7452**	7500.48	*	*	1	7472	7861	10521	14317	-	16184

"" means GE-HH result is better than other methods. "-" indicates no feasible solution has been found. Best results are highlighted in bold.* Δ*(%): *the percentage deviation of the average value with regard to the best known results.*

TABLE 17 RESULTS OF GE-HH ON THE ITC 2007 EXAM TIMETABLING DATASETS COMPARED TO POST-ITC 2007 APPROACHES

	GE-HH					GE-HH**	Post ITC 2007				
							Hyper-heuristic		Bespoke methods		
Instances	Best	Average	Δ (%)	Δ* (%)	Rank	Bets	$Mitc_6$	$Mitc_7$	$Mitc_8$	$Mitc_9$	$Mitc_{10}$
Dataset 1	**4362**	**4394.10**	*	*	1	4370	6235	8559	4775	4370	4633
Dataset 2	**380**	**399.80**	*	3.84	1	380	2974	830	385	385	405
Dataset 3	**8991**	**9072.35**	*	0.84	1	8995	15832	11576	8996	9378	9064
Dataset 4	**15094**	**15483.42**	*	0.75	1	15184	35106	21901	16204	15368	15663
Dataset 5	**2912**	**3010.15**	*	0.74	1	2993	4873	3969	2929	2988	3042
Dataset 6	**25735**	**25792.35**	*	0.20	1	25786	31756	28340	25740	26365	25880
Dataset 7	**4025**	**4062.85**	*	*	1	4041	11562	8167	4087	4138	4037
Dataset 8	**7452**	**7500.48**	*	*	1	7472	20994	12658	7777	7516	7461

"" means GE-HH result is better than other methods. Best results are highlighted in bold.* Δ*(%): *the percentage deviation of the average value with regard to the best known results.*

B. Problems Domain II: Computational Results on Capacitated Vehicle Routing Problems

1) Test Set I: Christofides Datasets

The experimental results of GE-HH and GE-HH* are reported in Table 18, where for 4 out of 14 instances, GE-HH achieved better results than GE-HH* (tie on7 instances). The average results obtained by GE-HH on all instances are better than GE-HH* and the standard deviation is relatively small (varies between 0.00 and 0.93). Even though GE-HH did not outperform GE-HH* across all instances, however, the standard deviation reveals that GE-HH generalized well overall instances. Overall, the result implies that hybridizing the adaptive memory mechanism with GE-HH has made a significant improvement.

We compare the experimental results of GE-HH with the best available results in the literature in Table 19. To the best of our knowledge, only two hyper-heuristics have been tested on Christofides instances (first and second methods in Table 19) and both report the percentage deviation only. Due to the large number of bespoke methods that are available in the literature, we have only considered those that have produced the best known results and some of recent published methods. The considered methods are classified into single based and population based solution methods (see Table 19). Table 20 shows the comparison of GE-HH against hyper-heuristic methods in term of percentage deviation from the best known results. We can see that, for 9 instances GE-HH matches the best known results in the literature and for 4 instances, GE-HH produced a better quality (ranked first) when compared to other hyper-heuristics. The computational results of GE-HH compared to other bespoke methods are presented in Table 21, where for 9 out of 12 instances GE-HH has obtained the best known results. For the remaining instances, the quality of the solutions with regard to percentage deviation is between 1.9% and 0.11% and instance ranking varies between 2 and 4. According to this result, GE-HH is competitive with the presented bespoke methods. Considering the generality, it is obvious that GE-HH is able to produce good results across all instances and the percentage deviation is relatively small.

TABLE 18 RESULTS OF GE-HH COMPARED TO GE-HH*

	GE-HH				GE-HH*			
Instances	*Best*	*Average*	*Std*	*Time*	*Best*	*Average*	*Std*	*Time*
1	**524.61**	***524.61***	**0.00**	10.12	**524.61**	**524.61**	**0.00**	8.20
2	**835.26**	**835.86**	**0.80**	21.02	**835.26**	836.14	1.27	16.12
3	**826.13**	**827.09**	**0.62**	20.33	**826.13**	827.71	1.48	15.06
4	**1029.65**	**1034.13**	**0.92**	30.43	1032.51	1034.71	1.37	24.43
5	**1308.54**	**1316.89**	**0.87**	19.09	1310.62	1317.51	4.51	16.08
6	**555.43**	**555.43**	**0.00**	9.43	**555.43**	555.79	0.57	7.43
7	**909.67**	910.17	**0.91**	11.18	**909.67**	**910.10**	1.10	8.70
8	**865.94**	**866.10**	**0.35**	13.44	**865.94**	866.19	0.41	10.06
9	**1164.98**	**1170.96**	**0.27**	19.67	**1164.35**	1171.73	3.29	16.11
10	**1403.38**	**1412.49**	**0.96**	21.83	1405.94	1414.25	3.69	18.71
11	**1042.12**	**1054.84**	**0.93**	12.65	**1042.11**	1091.17	6.51	7.95
12	**819.55**	**819.55**	**0.00**	9.95	**819.55**	820.21	1.96	6.34
13	**1543.05**	**1551.59**	**0.18**	10.07	1543.83	1554.03	2.28	7.83
14	**866.36**	**866.36**	**0.00**	12.62	**866.36**	866.39	0.11	8.16

Note: GE-HH: with the adaptive memory mechanism. GE-HH: without adaptive memory. Time: represents average time in minutes. Best results are highlighted in bold.*

TABLE 19 ACRONYMS OF COMPARED METHODS

#	Symbol	References			
1	*Cvrp$_1$1*	[75]	*HH*		
2	*Cvrp$_1$2*	[76]			
3	*Cvrp$_1$3*	[77]	*HH*	*LS*	*NON-HH*
4	*Cvrp$_1$4*	[78]		*POP*	
5	*Cvrp$_1$5*	[79]			
6	*Cvrp$_1$6*	[80]			
7	*Cvrp$_1$7*	[81]			
8	*Cvrp$_1$8*	[82]			
9	*Cvrp$_1$9*	[83]			

Note: HH: hyper-heuristic methods. NON-HH: bespoke methods. LS: local search methods. POP: population based methods

TABLE 20 RESULTS OF GE-HH COMPARED TO HYPER-HEURISTIC METHODS

	GE-HH					Hyper-heuristics		
Instances	*Best*	*Average*	*Δ(%)*	*Δ*(%)*	*Rank*	*Cvrp$_1$1*	*Cvrp$_1$2*	*BK*
1	**524.61**	**524.61**	**0.00**	0	*	**0.00**	**0.00**	524.61
2	**835.26**	**835.86**	**0.00**	0.07	*	0.05	0.62	835.26
3	**826.13**	827.09	**0.00**	0.11	*	0.21	0.42	826.14
4	1029.65	1034.13	**0.11**	0.55	1	0.52	2.50	1028.42
5	1308.54	1316.89	**1.33**	1.98	1	2.05	5.07	1291.29
6	**555.43**	**555.43**	**0.00**	0	*	0.00	-	555.43
7	**909.67**	**910.17**	**0.00**	0.05	*	0.09	-	909.68
8	**865.94**	**866.10**	**0.00**	0.01	*	0.00	-	865.94
9	1164.98	1170.96	**0.20**	0.72	1	0.70	-	1162.55
10	1403.38	1412.49	**0.53**	1.19	1	1.24	-	1395.85
11	**1042.11**	1054.84	**0.00**	1.22	*	0.88	0.19	1042.11
12	**819.55**	**819.55**	**0.00**	0	*	**0.00**	**0.00**	819.56
13	1543.05	1551.59	1.90	2.47	2	**1.00**	-	1514.14
14	**866.36**	**866.36**	**0.00**	0	*	**0.00**	-	866.37

Note: '' indicates that the obtained result is the same as the best known result. BK: best known results in the literature. "-" indicates no feasible solution has been found. Best results are highlighted in bold. Δ*(%): the percentage deviation of the average value with regard to the best known results.*

TABLE 21 RESULTS OF GE-HH COMPARED TO BESPOKE METHODS

	GE-HH				Bespoke methods						
					Single solutions based					Population based	
Instances	*Best*	*Average*	*Δ(%)*	*Rank*	*Cvrp$_1$3*	*Cvrp$_1$4*	*Cvrp$_1$5*	*Cvrp$_1$6*	*Cvrp$_1$7*	*Cvrp$_1$8*	*Cvrp$_1$9*
1	**524.61**	524.61	0.00	*	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	**524.61**	**524.71**
2	**835.26**	835.86	0.00	*	**835.26**	835.77	**835.26**	838.60	840.47	**835.26**	849.77
3	**826.13**	827.09	0.00	*	**826.14**	829.45	**826.14**	828.56	**826.14**	**826.14**	844.72
4	1029.65	1034.13	0.11	2	**1028.42**	1036.16	**1028.42**	1033.21	1032.19	**1028.42**	1059.03
5	1308.54	1316.89	1.33	4	1298.79	1322.65	**1291.45**	1318.25	1309.72	1294.21	1302.33
6	**555.43**	555.43	0.00	*	**555.43**	**555.43**	**555.43**	**555.43**	-	**555.43**	**555.43**
7	**909.67**	910.17	0.00	*	**909.68**	913.23	**909.68**	920.72	-	**909.68**	**909.68**
8	**865.94**	866.10	0.00	*	**865.94**	**865.94**	**865.94**	869.48	-	**865.94**	866.32
9	1164.98	1170.96	0.20	3	**1162.55**	1177.76	**1162.55**	1173.12	-	1163.41	1181.60
10	1403.38	1412.49	0.53	4	1397.94	1418.51	**1395.85**	1435.74	-	1397.51	1417.88
11	**1042.11**	1054.84	0.00	*	**1042.11**	1073.47	**1042.11**	1042.87	1042.11	**1042.11**	**1042.11**
12	**819.55**	819.55	0.00	*	**819.56**	**819.56**	**819.56**	919.56	819.56	**819.56**	847.56
13	1543.05	1551.59	1.90	2	**1541.14**	1573.81	**1541.14**	1545.51	-	1544.57	1542.86

14	**866.36**	866.36	0.00	*	**866.37**	**866.37**	**866.37**	**866.37**	-	**866.37**	**866.37**

Note: '' indicates that the obtained result is the same as the best known result. "-" indicates no feasible solution has been found. Best results are highlighted in bold.*

2) Test Set II: Golden Datasets

The computational results of GE-HH and GE-HH* are tabulated in Table 22. The presented results clearly show that GE-HH outperformed GE-HH* across all instances. Furthermore, the average and standard deviation of GE-HH is much better than GE-HH*, again indicating that the adaptive memory mechanism has a big impact on the performance and generality.

In order to assess the performance of GE-HH, the results of GE-HH are compared with the best available results in the literature. Again, due to the uncountable number of methods that have been tested on Golden instances, only those produced the best known results and few recent methods are considered as shown in Table 23. To the best of our knowledge, only one hyper-heuristic (first method in Table 23) has been tested on Golden instances. Table 24 gives the comparison results. From Table 24, one can find that, GE-HH reached the best known results for 4 out of 20 instances. For the other instances, the quality of solution (percentage deviation) is between 0.17% and 0.68% and instance ranking varies between 2 and 5. Compared to the hyper-heuristic method (first method in Table 24), GE-HH is able to obtain better solutions on 14 instances. When comparing with bespoke methods, for 4 instances GE-HH reached the best known results. GE-HH produces competitive results for the remaining 16 instances compared to other bespoke methods and very close to the best known value (percentage deviation). It should be noted that bespoke methods are specifically designed to produce the best results for one or more instances, whilst, one can see that GE-HH is able to obtain a much higher level of generality across all instances.

TABLE 22 RESULTS OF GE-HH COMPARED TO GE-HH*

	GE-HH				***GE-HH****			
Instances	***Best***	***Average***	***Std***	***Time***	***Best***	***Average***	***Std***	***Time***
1	**5626.81**	**5631.56**	**0.92**	15.04	5703.21	5697.56	1.81	10.27
2	**8446.19**	**8457.16**	**1.24**	22.13	8484.16	8457.16	1.67	18.09
3	**11081.60**	**11120.40**	**1.07**	32.06	11138.44	11120.40	1.18	27.31
4	**13658.84**	**13673.64**	**1.30**	37.31	13708.26	13673.64	1.46	32.19
5	**6460.98**	**6494.86**	**0.84**	17.24	6468.83	6494.86	1.53	14.27
6	**8462.10**	**8488.93**	**1.03**	19.11	8485.30	8488.93	1.16	16.42
7	**10202.24**	**10280.32**	**1.10**	31.08	10262.43	10280.32	1.20	28.40
8	**11690.82**	**11795.80**	**1.03**	41.64	11784.50	11795.80	1.11	36.08
9	**583.39**	**596.19**	**0.75**	18.52	589.92	596.19	1.26	13.92
10	**740.91**	**769.98**	**1.02**	22.18	758.22	789.98	1.13	18.13
11	**919.80**	**986.60**	**0.90**	29.37	949.38	986.60	1.31	25.08
12	**1111.43**	**1126.64**	**1.02**	40.19	1155.76	1186.64	1.10	36.10
13	**857.19**	**868.73**	**0.86**	30.08	876.64	898.73	1.21	26.06
14	**1083.59**	**1108.12**	**0.96**	24.40	1097.61	1108.12	1.42	19.20
15	**1350.17**	**1390.16**	**0.84**	35.08	1376.42	1390.16	1.38	29.06
16	**1631.91**	**1682.98**	**0.93**	42.15	1640.19	1682.98	1.29	37.12
17	**707.76**	**718.56**	**0.60**	18.07	714.52	720.56	1.01	14.10
18	**1003.43**	**1017.13**	**1.08**	19.11	1017.24	1057.13	1.15	16.02
19	**1368.12**	**1390.62**	**1.30**	26.30	1374.11	1390.62	1.46	21.14
20	**1820.09**	**1855.16**	**0.77**	32.08	1830.48	1855.16	1.09	28.06

Note: GE-HH: with the adaptive memory mechanism. GE-HH: without adaptive memory. Time represents average time in minutes. Best results are highlighted in bold.*

TABLE 23 ACRONYMS OF COMPAREDMETHODS

#	***Symbol***	**References**
1	*$Cvrp_2 1$*	[84]
2	*$Cvrp_2 2$*	[85]
3	*$Cvrp_2 3$*	[86]
4	*$Cvrp_2 4$*	[17]
5	*$Cvrp_2 5$*	[82]
6	*$Cvrp_2 6$*	[81]
7	*$Cvrp_2 7$*	[87]

TABLE 24 RESULTS OF GE-HH COMPARED TO BESPOKE METHODS

	GE-HH					***HH***	***Bespoke methods***					
Instances	***Best***	***Average***	**Δ(%)**	**Δ*(%)**	***Rank***	***$Cvrp_2 1$***	***$Cvrp_2 1$***	***$Cvrp_2 3$***	***$Cvrp_2 4$***	***$Cvrp_2 5$***	***$Cvrp_2 6$***	***$Cvrp_2 7$***
1	**5626.81**	5631.56	0.00	0.08	*	5650.91	5627.54	**5626.81**	5759.61	5670.38	5638.42	5643.27
2	8446.19	8457.16	0.17	0.30	2	8469.32	8447.92	**8431.66**	8501.67	8459.73	8457.04	8455.12
3	11081.60	11120.40	0.43	0.76	3	11047.01	**11036.22**	**11036.22**	11364.69	11101.12	11098.93	11083.49
4	13658.84	13673.64	048	0.59	4	13635.31	13624.52	**13592.88**	14136.32	13698.17	13816.35	13671.18
5	**6460.98**	6494.86	0.00	0.52	*	6466.68	**6460.98**	**6460.98**	6512.27	**6460.98**	**6460.98**	**6460.98**
6	8462.10	8488.93	0.68	1.00	5	8416.13	8412.88	**8404.26**	8553.19	8470.64	8430.66	8461.18
7	10202.24	10280.32	0.44	1.21	5	10181.75	10195.56	**10156.58**	10422.65	10215.14	10209.64	10198.25
8	11690.82	11795.80	0.23	1.13	2	11713.62	**11663.55**	11691.06	11986.73	11750.38	11785.11	11695.24

9	583.39	596.19	0.51	2.71	2	585.14	583.39	**580.42**	586.68	586.87	585.29	583.39
10	740.91	769.98	0.32	4.26	2	748.89	741.56	**738.49**	748.89	746.56	745.25	743.19
11	919.80	986.60	0.55	7.85	3	922.70	918.45	**914.72**	924.70	925.52	924.74	922.17
12	1111.43	1126.64	0.42	1.79	4	1119.06	1107.19	**1106.76**	1125.71	1114.31	1123.29	1111.28
13	**857.19**	868.73	0.00	1.34	*	864.68	859.11	**857.19**	867.29	865.19	861.94	860.17
14	1083.59	1108.12	0.28	2.55	3	1095.40	1081.31	**1080.55**	1098.86	1089.21	1097.49	1085.24
15	1350.17	1390.16	0.56	3.54	4	1359.94	1345.23	**1342.53**	1356.65	1355.28	1356.34	1346.18
16	1631.91	1682.98	0.68	3.83	4	1639.11	1622.69	**1620.85**	1642.90	1632.21	1643.74	1625.89
17	**707.76**	718.56	0.00	1.52	*	708.90	707.79	**707.76**	712.26	712.18	709.84	710.87
18	1003.43	1017.13	0.83	2.21	5	1002.42	998.73	**995.13**	1017.91	1006.31	1005.97	1001.17
19	1368.12	1390.62	0.15	1.80	4	1374.24	1366.86	**1365.97**	1384.93	1373.24	1387.93	1366.86
20	1820.09	1855.16	0.003	1.93	2	1830.80	1820.09	**1820.02**	1855.91	1831.17	1872.45	1824.14

'' indicates that the obtained result is the same as the best known result. HH: hyper-heuristic method. Best results are highlighted in bold.*

VII. Discussion

As shown throughout this work, in both problem domains (exam timetabling and capacitated vehicle routing problems), GE-HH obtained competitive results, if not better (on some instances), when compared against existing best methods in the literature. GE-HH is able to update the best known results for some instances (on both domains). In both domains, our GE-HH outperformed previously proposed hyper-heuristic methods. We note that, for both domains, the standard deviation is relatively small. Also, the percentage deviation demonstrates that, in both domains, GE-HH results are very close to the best known. This positive result reveals that our GE-HH is efficient, consistent and generalizes well over both domains. In our opinion, this is due to the following. (i) The capability of GE-HH in dealing with different problem instances by evolving different local search templates during the problem solving process. By evolving different local search templates, GE-HH can easily adapt to any changes that might occur during problem solving. (ii) Since some problem instances are very difficult to solve and have many local optima, GE-HH struggles in obtaining good quality solutions without getting stuck in local optima. Therefore, by incorporating the adaptive memory mechanism, GE-HH is more effective in diversifying the search of solutions by exploring different regions. Overall, the benefit of the proposed method is its ability to find the best solver from the supplied pool of solvers (local search acceptance criteria) as well as the best configuration for the selected solver. This alleviates the question of which solver one should use and what is the best configuration for it. Furthermore, it does not rely on complicated search approaches to find out how to generate a local search template. Rather, it provides a general mechanism regardless of the nature and complexity of the problems. It is simple to implement, and can be easily applied to other domains without significant effort (i.e. users only need to change the set of neighborhood structures).

VIII. Conclusions

In this work, we have proposed a new improvement based hyper-heuristic framework for combinatorial optimization problems. The proposed framework employs a grammatical evolution algorithm (GE-HH) to search the space of basic heuristic components. These are: a set of acceptance criteria, neighborhood structures and neighborhood combinations and are represented by a grammar definition. The proposed framework takes these heuristic components as input and evolves several templates of perturbation heuristics during problem solving. The performance of the GE-HH is enhanced by hybridizing it with an adaptive memory mechanism which contains a set of high quality and diverse solutions. To demonstrate the generality, consistency and efficiency of the proposed framework, we have tested the proposed framework on two different and challenging problem domains, exam timetabling and capacitated vehicle routing benchmark problems, using the same parameter settings. The results demonstrate that GE-HH produces highly competitive solutions, if not better, and generalizes well across both problem domains. The main contributions of this work are:

- The development of a GE-HH framework that automatically generates templates of perturbation heuristics, demonstrating that strengths of different search algorithms can be merged into one hyper-heuristic framework.
- The integration of an adaptive memory mechanism, which contains a collection of high quality and diverse solutions, within a hyper-heuristic framework, and which also obtained consistent results, generalized across different problem domains and produced high quality solutions which are either competitive or better than (on some cases) other bespoke methods.
- The development of a hyper-heuristic framework which can be easily applied to different problem domains without much effort (i.e. the user only needs to change the neighborhood structures).

Experimental results have demonstrated the effectiveness and the generality of this method on very well established benchmarks. In our future work, we intend to investigate the effectiveness of integrating GE-HH in the HyFlex framework (a benchmark framework for cross-domain heuristic search) that has been recently introduced [88, 89].

Acknowledgment

The authors wish to thank Universiti Kebangsaan Malaysia for supporting this work under the UKM Action Research Grant Scheme (UKM-PTS-011-2009) and Fundamental Research Grant Scheme (UKM-TT-02- FRGS 0121- 2009).

References

[1] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*: Morgan Kaufmann, 2005.

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

[4] M. Gendreau and J. Y. Potvin, *Handbook of Metaheuristics*: Springer Verlag, 2010.

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

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

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

[8] R. Poli and M. Graff, "There is a free lunch for hyper-heuristics, genetic programming and computer scientists," in *Genetic Programming*, 2009, pp. 195-207.

[9] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, pp. 1-25, 2008.

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

[11] R. Battiti and M. Brunato, "Reactive search optimization: learning while optimizing," *Handbook of Metaheuristics*, pp. 543-571, 2010.

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

[13] 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.

[14] M. W. Carter, G. Laporte, and S. Y. Lee, "Examination timetabling: Algorithmic strategies and applications," *The Journal of the Operational Research Society*, vol. 47, pp. 373-383, 1996.

[15] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu, and E. K. Burke, "Setting the research agenda in automated timetabling: The second international timetabling competition," *INFORMS Journal on Computing*, vol. 22, pp. 120-130, 2010.

[16] N. Christofides, A. Mingozzi, and P. Toth, "The vehicle routing problem," *Combinatorial optimization*, vol. 11, p. 315338, 1979.

[17] F. Li, B. Golden, and E. Wasil, "Very large-scale vehicle routing: new test problems, algorithms, and results," *Computers & Operations Research*, vol. 32, pp. 1165-1179, 2005.

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

[19] 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.

[20] 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.

[21] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical Evolution of Local Search Heuristics," *IEEE Transactions on Evolutionary Computation*, 2011.

[22] P. Garrido and M. C. Riff, "DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *Journal of Heuristics*, pp. 1-40, 2010.

[23] 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.

[24] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart, "Hyper-heuristics: learning to combine simple heuristics in bin-packing problems," in *In Proceedings of GECCO'2002*, 2002, pp. 942–948.

[25] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary Computation*, vol. 16, pp. 31-61, 2008.

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

[27] 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.

[28] R. Poli, J. Woodward, and E. K. Burke, "A histogram-matching approach to the evolution of bin-packing strategies," in *Proceedings of the IEEE Congress of Evolutionary Computation (CEC 2007)*, 2007, pp. 3500-3507.

[29] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A Survey of the State of the Art," *Journal of the Operational Research Society, to appear*, 2012.

[30] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 349-358, 2001.

[31] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based Genetic Programming: a survey," *Genetic Programming and Evolvable Machines*, vol. 11, pp. 365-396, 2010.

[32] E. Soubeiga, "Development and application of hyperheuristics to personnel scheduling," *PhD thesis, School of Computer Science and Information Technology, The University of Nottingham*, 2003.

[33] D. Ouelhadj and S. Petrovic, "A cooperative hyper-heuristic search framework," *Journal of Heuristics*, vol. 16, pp. 835-857, 2010.

[34] M. Ayob and G. Kendall, "A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine," in *Proceedings of the International Conference on Intelligent Technologies, InTech'03*, 2003, pp. 132-141.

[35] E. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. Vazquez-Rodriguez, and M. Gendreau, "Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms," in *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 2010, pp. 1-8.

[36] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, "Variable neighbourhood search: methods and applications," *Annals of Operations Research*, vol. 175, pp. 367-407, 2010.

[37] Z. Lü, J. K. Hao, and F. Glover, "Neighborhood analysis: a case study on curriculum-based course timetabling," *Journal of Heuristics*, pp. 1-22, 2011.

[38] F. Glover, "Tabu search and adaptive memory programming-advances, applications and challenges," *Interfaces in computer science and operations research*, vol. 1, 1996.

[39] D. S. Johnson, "A theoretician's guide to the experimental analysis of algorithms," *American Mathematical Society*, vol. 220, pp. 215-250, 2002.

[40] L. Di Gaspero and A. Schaerf, "Neighborhood portfolio approach for local search applied to timetabling problems," *Journal of Mathematical Modelling and Algorithms*, vol. 5, pp. 65-89, 2006.

[41] A. Goëffon, J. M. Richer, and J. K. Hao, "Progressive tree neighborhood applied to the maximum parsimony problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 136-145, 2008.

[42] E. K. Burke, A. J. Eckersley, B. McCollum, S. Petrovic, and R. Qu, "Hybrid variable neighbourhood approaches to university exam timetabling," *European Journal of Operational Research*, vol. 206, pp. 46-53, 2010.

[43] R. Bellio, L. Di Gaspero, and A. Schaerf, "Design and statistical analysis of a hybrid local search algorithm for course timetabling," *Journal of Scheduling*, pp. 1-13, 2011.

[44] C. Blum, J. Puchinger, G. Raidl, and A. Roli, "Hybrid metaheuristics," *Hybrid Optimization*, pp. 305-335, 2011.

[45] E. G. Talbi, "A taxonomy of hybrid metaheuristics," *Journal of Heuristics*, vol. 8, pp. 541-564, 2002.

[46] E. D. Taillard, L. M. Gambardella, M. Gendreau, and J. Y. Potvin, "Adaptive memory programming: A unified view of metaheuristics," *European Journal of Operational Research*, vol. 135, pp. 1-16, 2001.

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

[48] C. Fleurent and J. Ferland, "Genetic hybrids for the quadratic assignment problem," *American Mathematical Society*, vol. 16, pp. 173-187, 1993.

[49] V. Nannen and A. Eiben, "Efficient relevance estimation and value calibration of evolutionary algorithm parameters," in *Proceedings of the IEEE Congress of Evolutionary Computation (CEC 2007)*, 2007, pp. 103-110.

[50] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, pp. 87-127, 1999.

[51] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, "A survey of search methodologies and automated system development for examination timetabling," *Journal of Scheduling*, vol. 12, pp. 55-89, 2009.

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

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

[54] G. Clarke and J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, pp. 568-581, 1964.

[55] J. Silberholz and B. Golden, "Comparison of metaheuristics," *Handbook of Metaheuristics*, pp. 625-640, 2010.

[56] H. Asmuni, E. Burke, J. Garibaldi, and B. McCollum, "Fuzzy multiple heuristic orderings for examination timetabling," *Practice and Theory of Automated Timetabling V*, pp. 334-353, 2005.

[57] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177-192, 2007.

[58] N. Pillay and W. Banzhaf, "A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem," *European Journal of Operational Research*, vol. 197, pp. 482-491, 2009.

[59] R. Qu, E. K. Burke, and B. McCollum, "Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems," *European Journal of Operational Research*, vol. 198, pp. 392-404, 2009.

[60] L. Merlot, N. Boland, B. Hughes, and P. Stuckey, "A hybrid algorithm for the examination timetabling problem," *Practice and Theory of AutomatedTimetabling IV*, pp. 207-231, 2003.

[61] E. Burke and J. Newall, "Enhancing timetable solutions with local search methods," *Practice and Theory of AutomatedTimetabling IV*, pp. 195-206, 2003.

[62] S. Abdullah and E. Burke, "A Multi-start large neighbourhood search approach with local search methods for examination timetabling," 2006, pp. 6-10.

[63] M. Caramia, P. Dell'Olmo, and G. F. Italiano, "Novel local-search-based approaches to university examination timetabling," *INFORMS Journal on Computing*, vol. 20, p. 86, 2008.

[64] Y. Yang and S. Petrovic, "A novel similarity measure for heuristic selection in examination timetabling," *Practice and Theory of Automated Timetabling V*, pp. 247-269, 2005.

[65] E. K. Burke and Y. Bykov, "Solving exam timetabling problems with the flex-deluge algorithm," in *Proceedings of the Sixth International Conference on the Practice and Theory of Automated Timetabling*, 2006, pp. 370–372.

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

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

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

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

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

[71] E. Burke, R. Qu, and A. Soghier, "Adaptive Selection of Heuristics for Improving Constructed Exam Timetables," in *proceedings of PATAT10*, 2010, pp. 136-151.

[72] N. Pillay, "Evolving Hyper-Heuristics for a Highly Constrained Examination " in *In proceedings of PATAT10*, 2010, pp. 336-346

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

[74] B. McCollum, P. McMullan, A. Parkes, E. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," in *Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 2009, pp. 424–434.

[75] P. Garrido and C. Castro, "Stable solving of CVRPs using hyperheuristics," in *Proceeding GECCO '09*, 2009, pp. 255-262.

[76] D. Meignan, A. Koukam, and J. C. Créput, "Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism," *Journal of Heuristics*, pp. 1-21, 2010.

[77] É. Taillard, "Parallel iterative search methods for vehicle routing problems," *Networks*, vol. 23, pp. 661-673, 1993.

[78] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Science*, vol. 40, pp. 1276-1290, 1994.

[79] Y. Rochat and É. D. Taillard, "Probabilistic diversification and intensification in local search for vehicle routing," *Journal of Heuristics*, vol. 1, pp. 147-167, 1995.

[80] P. Toth and D. Vigo, "The granular tabu search and its application to the vehicle-routing problem," *INFORMS Journal on Computing*, vol. 15, p. 333, 2003.

[81] C. Alabas-Uslu and B. Dengiz, "A self-adaptive local search algorithm for the classical vehicle routing problem," *Expert Systems With Applications*, 2011.

[82] Y. Marinakis and M. Marinaki, "A hybrid genetic-Particle Swarm Optimization Algorithm for the vehicle routing problem," *Expert Systems With Applications*, vol. 37, pp. 1446-1455, 2010.

[83] A. I. Yurtkuran and E. Emel, "A new Hybrid Electromagnetism-like Algorithm for capacitated vehicle routing problems," *Expert Systems With Applications*, vol. 37, pp. 3427-3433, 2010.

[84] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & Operations Research*, vol. 34, pp. 2403-2435, 2007.

[85] D. Mester and O. Braysy, "Active-guided evolution strategies for large-scale capacitated vehicle routing problems," *Computers & Operations Research*, vol. 34, pp. 2964-2975, 2007.

[86] Y. Nagata, "Edge assembly crossover for the capacitated vehicle routing problem," in *Evolutionary Computation in Combinatorial Optimization*, 2007, pp. 142-153.

[87] Y. Marinakis and M. Marinaki, "Bumble Bees Mating Optimization Algorithm for the Vehicle Routing Problem," *Handbook of Swarm Intelligence*, pp. 347-369, 2010.

[88] 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 Evolutionary Computation in Combinatorial Optimization," in *EVOLUTIONARY COMPUTATION IN COMBINATORIAL OPTIMIZATION*, 2012, pp. 136-147.

[89] E. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A. Parkes, and S. Petrovic, "The Cross-Domain

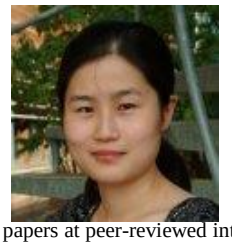
Heuristic Search Challenge – An International Research Competition " in *Learning and Intelligent Optimization*, 2011, pp. 631-634.

Nasser R. Sabar received the B.Sc. degree in Computer Science from University of Al-Anbar, Iraq and M.Sc. degree in Computer Science from National University of Malaysia (UKM), in 2005 and 2010, respectively. Currently, he is working toward the PhD degree in Computer Science at Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent (CAIT), National University of Malaysia. He has published 3 papers at international journals and 7 papers at peer-reviewed international conferences. His research interests include the design and development of hyper-heuristic framework, adaptive algorithm, evolutionary computation, meta-heuristics with a specific interest in combinatorial optimization problems, dynamic optimization and data mining problems.

Dr. Masri Ayob is a lecturer in the Faculty of Information Science and Technology, the National University of Malaysia (UKM). She obtained her PhD in Computer Science at The University of Nottingham in 2005. Her main research areas include meta-heuristics, hyper-heuristics, scheduling and timetabling, especially educational timetabling, healthcare personnel scheduling and routing problems. She has published more than 10 papers at international journals and 40 papers at peer-reviewed international conferences. She was a member of ASAP research group at The University of Nottingham. Currently, she is a principle researcher in Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent (CAIT), UKM.

Graham Kendall is a Professor of Computer Science. He is currently based at the University of Nottingham's Malaysia campus, where he is serving as Vive-Provost (Research and Knowledge Transfer). He is a member of the Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science. Graham was awarded a BSc (Hons) First Class in Computation from the University of Manchester Institute of Science and Technology (UMIST), UK in 1997 and received his PhD from The University of Nottingham (School of Computer Science) in 2000. He is a Fellow of the Operational Research Society. Before entering academia he spent almost 20 years in the IT industry, working for various UK companies (including the Co-operative Wholesale Society and Provincial Insurance), undertaking a variety of roles including Computer Operator, Technical Support Manager and Service Manager. He has edited and authored 9 books and has published almost 50 refereed journal papers (the vast majority in ISI ranked journals) and over 90 papers in peer reviewed conferences. He is an Associate Editor of 8 international journals (including two IEEE Transactions). He chairs the Steering Committee of the Multidisciplinary International Conference on Scheduling: Theory and Applications, in addition to chairing several other international conferences in recent years. He has been awarded externally funded grants worth over £5.5M from a variety of sources including the Engineering and Physical Sciences Research Council and commercial organizations. Professor Kendall's expertise lies in Operational Research, Meta- and Hyper-Heuristics, Evolutionary Computation and Artificial Intelligence, with a specific interest in scheduling, including timetabling, sports scheduling, cutting and packing and rostering.

Dr. Rong Qu is an Associate Professor in the School of Computer Science at the University of Nottingham. She obtained her PhD in Computer Science at The University of Nottingham in 2002. Her main research areas include



meta-heuristics, constraint programming, mathematical programming, case based reasoning and knowledge discovery techniques on scheduling, especially educational timetabling, healthcare personnel scheduling and network routing problems, and a range of combination optimization problems including portfolio optimization. She has published more than 30 papers at international journals and 30 papers at peer-reviewed international conferences. Dr Qu is a guest editor of the special issue on "Artificial Intelligence Planning and Scheduling" at the Journal of Scheduling, and the special issue on "Evolutionary Computation in Scheduling" at IEEE Transactions on Evolutionary Computation. She has been the program chair of six workshops, special sessions or IEEE symposiums.