Grammar Based Hyper-Heuristic

By

Mohamed Adnan Abdulmuttaleb

Supervisor
Prof. Mohamed Bettaz

Co-Supervisor
Prof. Sunil Kumar Khatri

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree In Computer Science.

Deanship of Academic Research and Graduate Studies

Philadelphia University

January, 2019

جامعة فيلادلفيا
التفويض نموذج

أنا محمد عدنان فؤاد عبدالمطلب، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع:
التاريخ:

Philadelphia University
Authorization Form

I, Mohamed Adnan Fuad Abduluttaleb authorize Philadelphia University to supply copies of my Thesis to libraries or establishments or individuals upon request.

Signature:
Date:

Grammar Based Hyper-Heuristic

By

Mohamed Adnan Abdulmuttaleb

Supervisor
Prof. Mohamed Bettaz

Co-Supervisor
Prof. Sunil Kumar Khatri

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree In Computer Science.

Deanship of academic Research and Graduate Studies

Philadelphia University

January , 2019

Committee Decision

Successfully defended and approved on ----------

Examination Committee	**Signature**
Dr.,…….., Chairman Academic Rank: ……	----------
Dr. ……, member. Academic Rank: ……	----------
Dr. ……, member. Academic Rank: ……	----------
Dr. …….. , External Member. Academic Rank: ……	----------

(Name of University)

Dedication

To my family, to the revolution.

Acknowledgement

I want to express my greatest gratitude to my mother and my father. And all the thanks to my supervisor Dr. Mohamed Bettaz and for the committee.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Acronym	Meaning
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
VUS	Velocity Updating Strategy
GP	Genetic Programming
GGGP	Grammar Guided Genetic Programming
RCGA	Real-coded Genetic Algorithm
H	Heuristic
GHH	Generative Hyper-Heuristic
AC	Acceptance Criterion
TSP	Traveling Salesman Problem
VCP	Vehicle Crashworthiness Problem
TG3P	Tree Grammar-Guided Genetic Programming

Abstract

The development of proper algorithmic solution for a given class of problems requires a deep understanding of some optimization algorithms and this process is time consuming. In this study, we investigate the hyper-heuristic methodology which is a high-level search methodology that operates on search space of heuristic/meta-heuristic algorithms. Hyper-Heuristic aims at finding the most suitable algorithmic solution for a given class of problems. Hyper-heuristic is classified into two major classes: *selective* and *generative hyper-heuristic.* Our focus is on generative hyper-heuristic, especially on generative hyper-heuristic that operates on the meta-heuristic components of Particle Swarm Optimization (PSO) and Real-Coded Genetic Algorithm (RCGA). The study uses a modified Tree-based Grammar-guided Genetic Programming (TG3P), in order to generate adaptive hybrid PSO and RCGA solvers for continuous global optimization problems. We compared our results with two prominent PSO algorithms, and the results show that our proposed hyper-heuristic has very competitive efficiency.

1. Chapter One:

Introduction

1.1. Introduction

This chapter defines optimization problems and their characteristics. It spotlights the *Heuristic* and *Meta-Heuristic* strategies and their associated problems, then, it introduces the *Hyper-heuristic* strategy in its both approaches: the selective and the generative. Finally, we present the research problem along with its motivation and objectives.

1.2. Optimization Problems

Before introducing the hyper-heuristic methodology, we have to understand the relation between conventional methodologies and hyper-heuristic methodologies. Hyper-heuristic does not replace the conventional approaches, on the contrary, it emphasizes their power. It answers the question of which technique should we use to solve a given problem? And at which manner should we apply it? What is the suitable algorithm's configuration? But again, before discussing any technique that may solve any problem, we should understand the problems that we are trying to solve, for instance, the classes of the problems, the hardness of problems, etc. So, we give at first more insight on the types of problems that may be solved by heuristic, meta-heuristic, and hyper-heuristic, then, we introduce each approach separately.

Optimization problems are the problems in which we try to find the optimal solution (possibly more than one exists) from a set of all feasible solutions. Typically, in these kinds of problems we are not able to find the optimal solution or to determine if a solution is the optimal one. So, in practice, we try to find a near-optimal or an acceptable solution using heuristic, meta-heuristic or other stochastic techniques. Optimization problems can be divided mainly into two classes: continuous or discrete optimization problems. The two classes have an objective or multiple objectives which we try to minimize or maximize.

As an example of discrete optimization problem is the Traveling Salesman Problem (TSP) in which we try to find the minimum path for delivery vehicle visiting all the cities on a given map only once and then returning to the source city. The TSP is discrete since the set

of feasible solutions is finite, and it is simply the enumeration of all possible paths. In a map where each city is connected to all the other cities (complete graph) the size of this set would be equal to N!, where N is the number of cities in the map.

An example of continuous optimization problem is the Vehicle Crashworthiness Problem (VCP) where crashworthiness means the capability of a vehicle to protect its occupants during a crash (Bois et al. 2004). In this problem, we have three objectives which we try to minimize. The objectives are namely: weight, acceleration characteristics, and toe-board intrusion of the vehicle. Any solution to this problem has to set the values of five decision variables, each of which is bounded in upper-bound and lower-bound constraints, this formulation of VCP is presented in (Zhang 2007).

Combinatorial problems (discrete optimization) and continuous problems are further divided in the literature, for instance, we may classify combinatorial problems based on the type of the produced solution as follows:

- Selection problems (binary vector): Where we have a set of items and we want to select a subset from them. E.g. the knapsack problem.
- Ordering problems (permutation vector): Where we have a set of items and we want to impose order on them. E.g. the traveling salesman problem, process allocation in single CPU system.
- Resource allocation problems (graph): Here, we have two distinct sets of entities, a set of resources and a set of resources consumers, and we want to allocate the set of consumers to the set of resources. The resources and consumers may have different types and attributes. E.g. university courses timetabling problem, etc.

Our focus in this study is more on continuous optimization problems which are formulated as follows:

$$\text{Minimize } f(x) \text{ subject to: } x \in \Omega$$

Where x is a continuous vector with the domain $\Omega \in \mathbb{R}^n$, and $f(x): \Omega \to \mathbb{R}$ is a continuous real-valued function. Each element in the vector x is called decision variable where each decision variable may have constraints defined upon it such as upper and lower bounds. If the problem contains constraints on its definition we call it 'constrained optimization problem', on the other hand, if the problem is constraints-free we call it 'unconstrained optimization problem'.

Continuous optimization problems have many characteristics that describe them. One property that may be used to describe a given problem is the modality of its objective function. Modality relates to the landscape of the objective function. Modality has three types:

- Uni-modal problems: The problems where a single local maxima (peak) exists.
- Bi-modal problems: The problems where two local maxima (peaks) exist.
- Multi-modal problems: The problems where more than two local maxima exist.

Additionally continuous optimization problems can be described by their dimensionality and separability. Dimensionality refers to the number of decision variables inside the problem, while separability means how much the decision variables are correlated together. Separable function is easy to solve compared to a non-separable function because it can be decomposed into multiple simpler functions that can be solved and combined linearly. More in separability can be found in (Tang et al. 2009).

Optimization can be performed on two levels, the first level is local optimization which only aims at finding local minima/maxima, and the second is global optimization which aims at finding the global minima/maxima. Later, we find that heuristic techniques are more like local optimizer i.e. cannot escape from local minima/maxima, while meta-heuristic and hyper-heuristic are more like global optimizer i.e. have the capability of escaping local maxima/minima.

The majority of optimization problems cannot be solved with an exact solution in polynomial time due to the large search space of such problems. For instance, the previously mentioned

problem, TSP, is a NP-Hard problem with the worst-case time complexity O(n^2*2^n) (using dynamic programming solution). So, while solving such problems, we usually seek an acceptable solution. For this reason, optimization problems are tackled conventionally using heuristic techniques, meta-heuristic techniques, stochastic techniques, or more recently, using hyper-heuristic techniques. Although each technique performs differently from the others, none of them guarantees optimality. But differently from the conventional techniques, hyper-heuristic has the advantage of generality since it can handle more than one class of problems. Also, hyper-heuristic reduces the efforts of development and tuning of an algorithmic solution to solve a specific problem type. In the next two sections, we shall have a quick glance on the heuristic and meta-heuristic techniques with their advantages and limitations, then, we introduce the hyper-heuristic methodology in depth.

1.3. Heuristic Strategy

When you engage with a problem, you gain some knowledge, and when you face the same problem again you apply this knowledge to solve the new problem. So, you have exploited the problem structure. This knowledge is what we call a heuristic, so, heuristic is a rule of thumb used to solve a particular problem without a grantee of optimality. Heuristic algorithms have no capability of escaping local maxima/minima, thus, they are not appropriate for complex problems. Typical applications of heuristic are when a fast solution is required, or it may be applied in conjunction with other techniques such as meta-heuristic algorithms, this hybridization produces what we call a *Memetic Algorithm*.

To illustrate the nature of heuristic algorithm more clearly, we give an example. We will continue with the TSP problem, a simple heuristic to solve the problem is simply to pick the nearest city from the current city, regardless of the future impact of this decision. For instance, the solution of the complete graph given in Figure 1 according to the Nearest Neighbor heuristic will be (A → D → C → B → E → A), where A is source city.

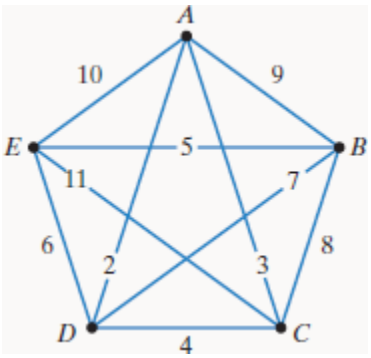

Figure 1: Complete weighted graph

Although heuristic techniques are more common for combinatorial optimization problems, heuristic techniques for continuous optimization problem exist such as the 'Nelder-mead method' (Nelder and Mead 1965).

1.4. Meta-Heuristic Strategy

While heuristic approach requires you to have some knowledge about the problem, meta-heuristic does not. As the name suggests, Meta means more an abstract heuristic, a heuristic that can be applied regardless of the underlying problem. In (Boussaïd, Lepagnot, and Siarry 2013) the authors mentioned some of the features that characterize meta-heuristic algorithms:

- They are nature inspired,
- Have components that are stochastic,
- Have control parameter.

The source of inspiration may be physics, biology, sociology, etc. The stochastic components of the algorithm may range from a single random variable to a complete operator, an example of stochastic operator is the mutation operator that can be found in the Genetic Algorithm 'GA'.

Meta-heuristic algorithms operate on two orthogonal dimensions, one of them expresses the diversification capability of the algorithm (exploration), while the other expresses the

intensification capability of the algorithm (exploitation). In the exploration stages of the algorithm's runtime, the algorithm tries to increase the area covered in the search space (explores previously unseen areas). While in the exploitation stages, the algorithm tries to intensify the search in specific areas in the search space (usually the more promising areas). More about the classification and analysis of meta-heuristic algorithms can be found in (Beheshti and Shamsuddin 2013; Boussaïd, Lepagnot, and Siarry 2013). For more insight on the nature of meta-heuristic algorithms and how they operate, two prominent meta-heuristic algorithms will be presented, namely, the *Particle Swarm Optimization* (PSO) and the *Genetic Algorithm* (GA).

1.4.1. Particle Swarm Optimization

PSO algorithm was originally proposed by Kennedy and Eberhart in the mid of 1990. The inspiration of PSO comes from the flocks of birds and swarms of the insects that search for food. In the original PSO algorithm, these are the main constituents:

- Particles: Which are evolved through the life time of the PSO algorithm. Each particle has a state that includes the following components:
    - Current Position: It represents the current state of the particle and a possible solution to the problem.
    - Velocity vector: It is a vector of scalars that determines the movement direction and speed of the particle.
    - Particle best position *pbest*: It is the fittest position in the trajectory of the particle.
- Velocity updating strategy: It determines how the velocity vector of each particle is computed. In other word, this strategy specifies how the particles are influenced by other particles in the swarm.
- Swarm: It is simply the set of all particles. The fittest position in the whole swarm is retained as the global best position *gbest*.

A noticeable characteristic of PSO is its memorization of state. This memorization is done on two levels: On particles level through the *pbest,* and on swarm level through the *gbest.* PSO as a concept is a set of particles that search in their localities and exchange useful information. Although PSO in practice is tuned to handle a class of problems specifically, theoretically it is applicable to any optimization problem as long as the problem is structured probably, thus, PSO is a meta-heuristic algorithm.

In the original PSO implementation, the velocity updating strategy is conducted as follows:

$$V_i^d \leftarrow V_i^d + c_1 * rand_i^d\left(Pbest_i^d - X_i^d\right) + c_2 * rand_i^d\left(Gbest^d - X_i^d\right)$$

Where X_i, V_i are the current position and velocity of the i_{th} particle. Obviously, each field d in the velocity vector of the i_{th} particle depends on three components: The previous velocity V_i^d, the global best in the swarm $gbest^d$, and the local best of the i_{th} particle $pbest_i^d$. Additionally, in the above updating strategy there are two acceleration coefficients, *c1* and *c2*, and random variable $rand_i^d$. And after we compute the new velocity for the i_{th} particle, its current position get updated as follows:

$$X_i^d \leftarrow X_i^d + V_i^d$$

The *gbest* emphasizes the exploration aspect of PSO, while the *pbest* emphasizes the exploitation capability aspect of PSO. The acceleration coefficients control the convergence speed of the algorithm.

1.4.2. Genetic Algorithm

GA is inspired from biological procedures, and it is one of the most commonly used algorithm in the optimization field. GA operates on a population of solutions called chromosomes, each component in the chromosome is called gene, at each iteration of GA a new population of children solutions (offspring) are reproduced from a subset of the current population (parents). GA exists with many variants. GA search strategy is based on three

main concepts:

- *Inheritance*: Is the process where the children chromosomes inherent the parents chromosomes. This is achieved by using the crossover operator which mixes the parts of the parents' chromosomes to form new children chromosomes, Figure 2 shows the single-point crossover.
- *Selection:* Is the *mechanism* that determines which individuals are selected for the reproduction.
- *Variation*: Is a mechanism that provides the individuals with new genetic information that does exist in the parent individuals. This is achieved by using the mutation operator.

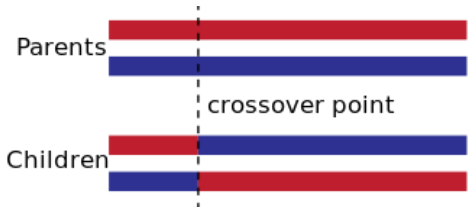

Figure 2: Single-point crossover

The general structure of GA is given in Figure 3.

```
1. Initialize the Population
2. Repeat until stop condition
3. Select individuals for reproduction
4. Apply crossover on selected individuals
5. Apply mutation on individuals with a predefined probability
5. Update population
6. End repeat
```

Figure 3: GA workflow.

For example, if we have two individuals selected from the population X_1, X_2 with binary

encoding, which are:

$$X_1 = (1,0,1,0,1,1,0,0,1,1)$$
$$X_2 = (0,0,1,1,0,0,0,0,1,0)$$

These two individuals (chromosome) with size m may represent a combination from a set of items with size m, 0 in the chromosome vector means that we are not selecting the i_{th} item while 1 means that we are selecting it. Now, if we want to perform a single-point crossover operator on these parent chromosomes we get the following child chromosomes:

$$X_{n1} = (1,0,1,0,1,0,0,0,1,0)$$
$$X_{n2} = (0,0,1,1,0,1,0,0,1,1)$$

Although we choose the crossover-point to be exactly in the middle, it does not have to be in the middle, it could be anywhere. Now, we can perform mutation operator on the first child individual by inversing the value on a random gene inside the chromosome, for instance, the 3_{th} gene:

$$X_{n1} = (1,0,0,0,1,0,0,0,1,0)$$

Or, we can swap the values of two randomly selected genes. After performing these operations, we usually update the population by replacing the parents with the child individuals if they have better fitness values.

Since heuristic algorithms may be trapped easily in local-optima, meta-heuristic algorithms are more likely to be used because of their capability of escaping local optima. But the development of an appropriate meta-heuristic algorithm for new encountered problems becomes more challenging because the development now requires a consideration of a huge and an increasing number of design aspects. Some of these choices are:

- What algorithm to use? The solvers needs to review the literature with tens of options

to make an adequate choice. This review requires too much time and efforts, and some previous knowledge in the field.

- What variant of the algorithm should be adopted? The original definition of each meta-heuristic algorithm is very general, and this may have an effect on the performance. So, many variants of a given algorithm exist, where each variant may favor the exploitation aspect over the exploration aspect or via versa. The solver needs to make an appropriate choice with respect to the difficulty of the problem or other characteristics.

- How to tune algorithm parameters? Every algorithm has multiple control parameters whose responsibilities vary according to the algorithm. Currently, the values of the different parameters are set according to the result of previous studies, test and evaluation, and using automatic learning mechanism. The first option may not be suitable, especially if the problem under consideration has features that are different from the features of the problem that has been previously solved. The second option employs trial and error cycles, obviously this is time consuming and it may lack accuracy. The last option is to employ some automatic learning mechanisms (offline learning prior to the search process, or online learning during the search process), this option provides more powerful results and probably solutions with higher quality, but it still needs more expertise to be accomplished.

In addition to the previous ones, the solvers need to handle other huge design issues that are algorithm specific such as the topology of the population in *particle swarm optimization,* initialization of the population in the *genetic algorithm*, etc.

These difficulties caused the emergence of hyper-heuristic algorithms. The intuitive idea behind hyper-heuristic is the ability to develop acceptable solutions for optimization problems with the minimum efforts and with a very high level of generality. In the next section, we will discuss hyper-heuristic in more details.

1.5. Hyper-heuristic

The authors in (E. K. Burke et al. 2010) defined hyper-heuristics as follows:

"*A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational optimization problems"*

Several definitions are proposed in the literature, but the aforementioned one is the most adopted definition. Although this definition overlooks that a hyper-heuristic is able to generate or select meta-heuristics, this definition is intuitive and general enough to cover the majority of researches in the hyper-heuristic field. A more mathematical and formal definition of hyper-heuristic can be found in (Swan et al. 2014):

$$h: W \rightarrow W$$
$$H: W \rightarrow W$$

In the above definition, both the low-level heuristics *h* and the hyper-heuristic *H* operate on workspace *W* that maintains the state of the heuristics and the state of the search. According to this definition, the hyper-heuristic becomes recursive in that the set of heuristics that *H* can access from the workspace may themselves be a hyper-heuristics. Although this definition does not make clear the boundary between heuristics and hyper-heuristics, in practice, a clear boundary is drawn between the high level strategy (hyper-heuristic) and the low-level operators (heuristic).

To give a theoretical structure for this field, the authors in (E. K. Burke et al. 2010) conducted a classification and criteria that help in providing a clear reading of the literature. According to the authors, there are two major classes that can be found in the literature:

- *Heuristic Selection:* It is the approach that is concerned with the strategies of choosing one or more heuristic from a set of heuristics.
- *Heuristic Generation:* It is the approach that is concerned with the strategies of

generating new heuristics from basic building blocks of existing heuristics.

The selective or the generative hyper-heuristic can be of any kind, it may be a heuristic, meta-heuristic, rule-based technique or any other technique. There is no constraint about its nature.

Considering the nature of the low-level heuristics that the hyper-heuristic may operate on, the authors have classified the low–level heuristics into two classes:

- *Constructive heuristic:* It is the heuristic that is intended to grow an incomplete solution.

- *Perturbation heuristic:* It is the heuristic that is intended to improve an already complete solution.

Perturbation heuristics are neighboring search techniques that are used in hill climbing or one of its variants. The solutions that the perturbation heuristics operate on are generated either randomly or through using some constructive heuristics.

Further on, the authors have classified hyper-heuristic approaches depending on whether the hyper-heuristic learns or not. If it is a learning hyper-heuristic and if the learning is performed during solving one problem instance, then, it is called *online learning hyper-heuristic.* If the learning is performed using a set of training problem instances and the learning generalization is applicable to any problem instance, then, the hyper-heuristic is called *offline learning hyper-heuristic*.

Now we will elaborate more on the major techniques used in heuristic-selection and heuristic-generation methodologies.

1.5.1. Selective Hyper-Heuristic

As described earlier, selective hyper-heuristic selects one or more heuristic from a set of heuristics. The common case is to use an *Acceptance Criterion (AC)* in addition to the

selection strategy. The main responsibility of the AC is to accept or to reject the new produced solution as the new incumbent solution. Usually, all ACs accept the improving solutions (solutions with higher quality) and accept the worsening solutions with some probability. The reason behind the acceptance criterion is escaping local optima that may occur if we reject all worsening solutions. So, one may review the literature using two perspectives: one of them considers the selection technique, while the other considers the AC. In our discussion of selective hyper-heuristic, we focus mainly on the selection technique and especially the used AC. In Figure 4, a general scheme of selective hyper-heuristic is given. In this scheme, we start our search by an empty or randomly initialized solution then we apply a selected heuristic (either perturbative or constructive) on the current solution, and the resultant new solution is either rejected or accepted according to the used AC.

- Current ← empty or randomly initialized solution.
- Repeat until some criteria is met.
- H ← Selected heuristic from a heuristic pool.
- New ← Apply H on current.
- New solution is either:
    - Accepted if It is better than current
    - Discarded with probability P
- End repeat.

Figure 4: Scheme for selective hyper-heuristic.

1.5.2. Generative Hyper-heuristic

The second approach in the hyper-heuristic field is the generative hyper-heuristic approach. This approach relates to the field of *Automatic Algorithm Design*, and it uses similar techniques, but the difference is that the automatically designed algorithms are of heuristic nature. The source of variability in selective hyper-heuristics comes mainly from the ability to select heuristics with different characteristics from the heuristic pool of a given class of problems, while in generative hyper-heuristics, the variability may cover the entire generated algorithms. In other words, in generative hyper-heuristics, we often have a range of possibilities for each design aspect of the algorithms being evolved. In this sense, we can

consider selective hyper-heuristics a special case of generative hyper-heuristics. Although selective hyper-heuristics do not operate on grammar, they contain the grammar implicitly which specifies what pool of heuristics we have for a given problem class. The grammar plays the key role in achieving good results in generative hyper-heuristic since the designed grammar should allow us to derive a wide range of algorithms with different capabilities because the problems that we want to solve have widely varying properties and complexities.

Typically in this approach, Genetic Programming (GP) is used as a hyper-heuristic to generate the algorithms. An early review of the employment of GP as a hyper-heuristic can be found in (E. K. Burke et al. 2009). GP is similar to the Genetic Algorithm (GA), the main difference is that instead of working in search space of solutions, GP works in search space of programs. Two common representations of the population's individuals in GP exist in the literature:

- *Tree Genetic Programming (TGP)* that is the conventional approach where programs are represented as a tree in which the leaf nodes are terminals and the inner nodes are non-terminals.
- *Grammatical Evolution (GE)* is initially proposed in (Ryan, Collins, and Neill 1998). It is a variant of GP where the population's individuals are represented as strings of integers (Genotype). Then, these strings are mapped to executable trees (Phenotype) using mapping function. This representation facilitates the manipulation procedures that operate on the population's individuals.

Figure 6 and Figure 5 give two examples of the crossover operator and the mutation operator of the Tree-based *Grammar Guided GP (GGGP)* respectively. In Figure 6, the crossover operator is applied by selecting one internal node from each parent individual with the same non-terminal, then, by swapping these nodes (multiplication) in both parents' trees we reproduce two children individuals. And in Figure 5, one non-terminal is selected from the tree and then regenerated randomly according to the used grammar.

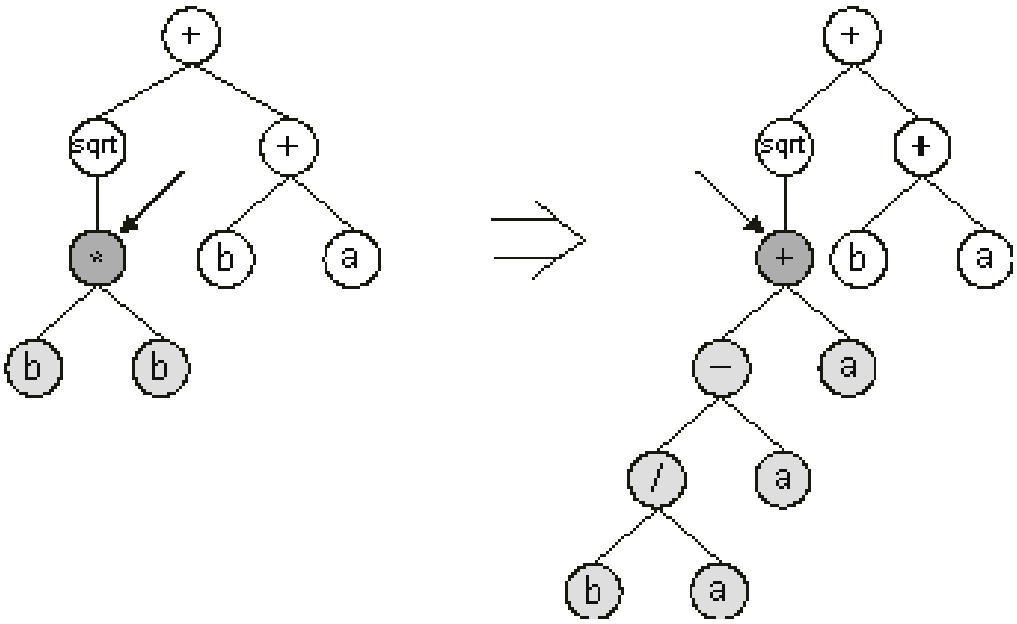

Figure 5: Example of mutation in tree-based GP.

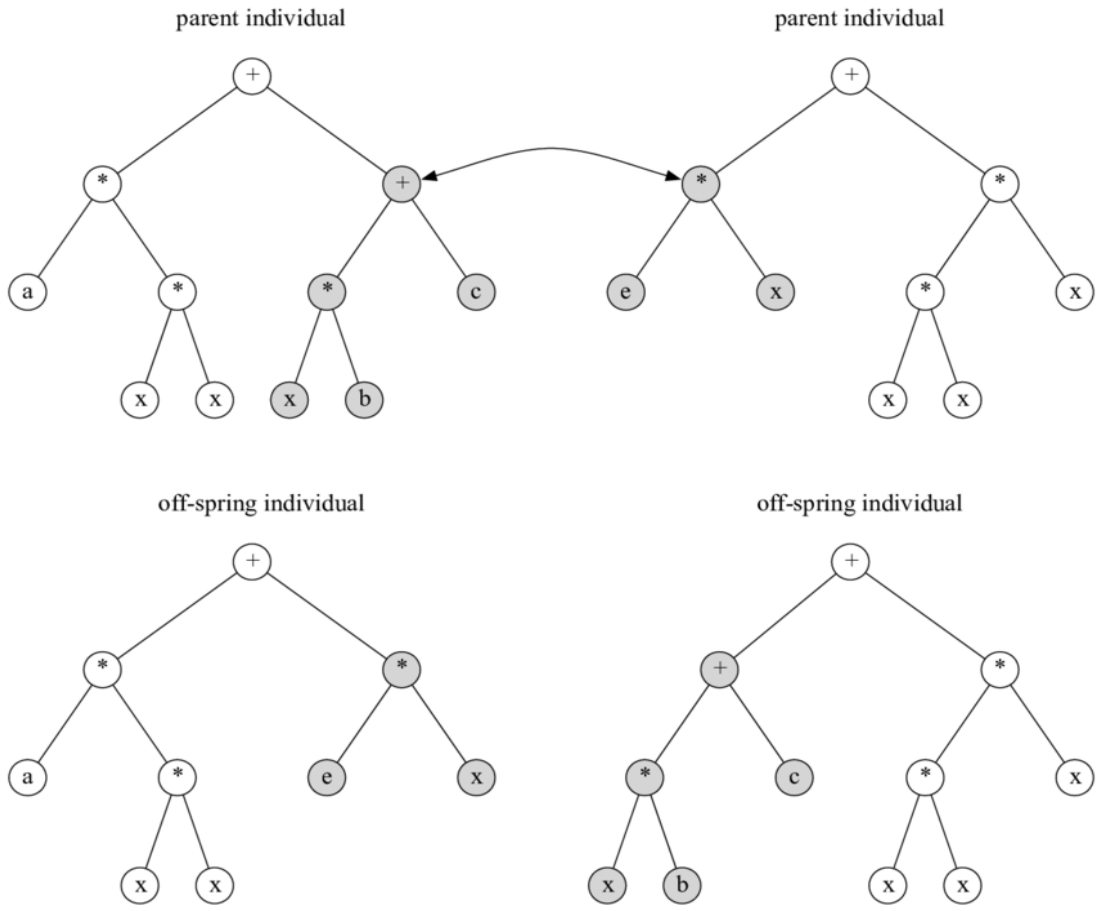

Figure 6: Example of crossover operator in tree-based GP

The Grammatical Evolution (GE) GP is also a Grammar-Guided GP, but it has linear representation of the chromosomes. Each gene in the chromosome is called codon. Each codon is 8 bit string, and the number of these codons in the chromosomes are not restricted in the original GE. The operators of GE, namely, crossover and mutation, are applied to the chromosomes (genotypes), and in order to evaluate the fitness of the chromosomes, a mapping procedure is employed. The mapping procedure transforms each genotype into its corresponding derivation tree (phenotype) by the following procedure:

Starting from the start symbol S of the grammar, use the first codon value to determine the rule to be selected,

$$CODON_i \ \% \ \#S$$

Where # produces the number of rules of a non-terminal. Then, repeat the previous procedure with the first left-hand non-terminal, but this time with the next codon value. This process is repeated until:

- A complete program is generated. This occurs when all the non-terminals in the expression being mapped are turned into elements from the terminal set of the BNF grammar.
- The end of the genome is reached, that is the last codon value that has been used. In this case, a wrapping operator is applied. Wrapping operator restarts the genome from the beginning. The number of applications of the wrapping operator may be limited by some threshold value.

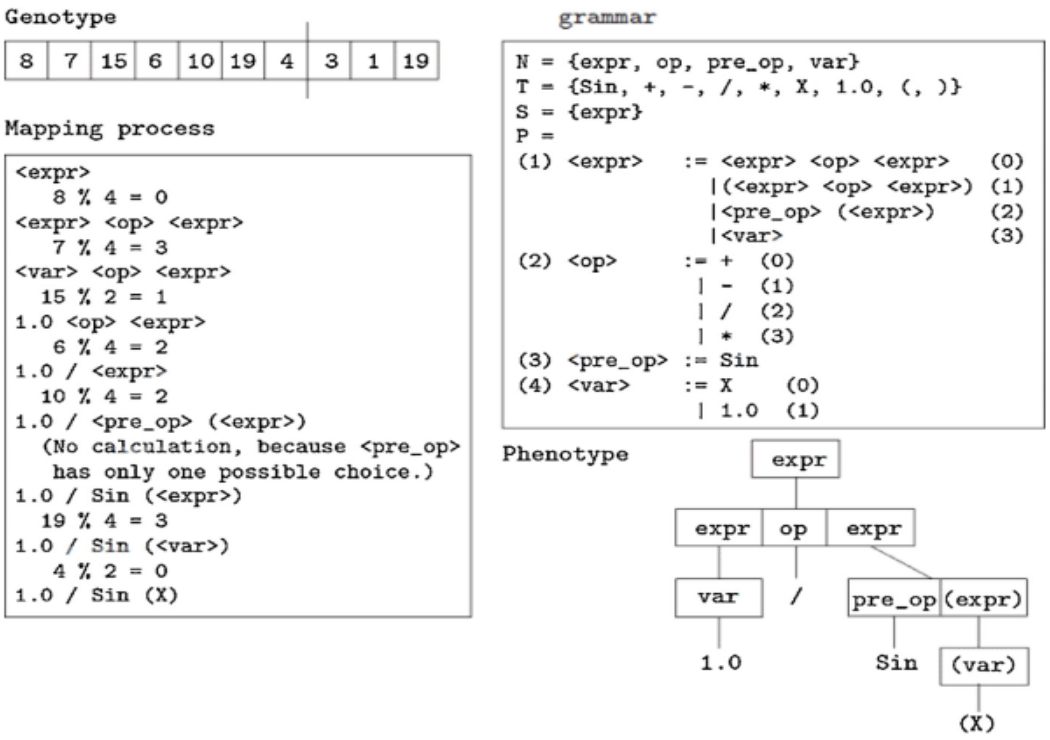

Figure 7: An example of GE workflow.

The standard single-point, two-point and uniform crossover may be used in GE to produce the offspring individuals. Tree representation does not suffer from the low locality problem that GE suffers from. Locality means how much the neighboring genotypes correspond to the neighboring phenotypes. High locality means high correspondence, while low locality means low correspondence. More on locality can be found in (Rothlauf and Oetzel 2006).

After we have introduced the three strategies which are namely: Heuristic, Meta-Heuristic and Hyper-Heuristic in this chapter, we discussed the characteristics of each strategy and how the problem of the two former strategies influenced the emergence of the hyper-heuristic strategy. Then, we highlighted the different techniques of implementing hyper-heuristic strategy (selective and generative). In the next chapter, a literature overview and analysis of generative hyper-heuristic is presented. This overview focuses on generative hyper-heuristics and further classifies the generative hyper-heuristics (GHH) according to the nature of the search-space on which they operate, so, we identify two main classes: GHHs that operate on heuristic components and GHHs that operate on meta-heuristic components.

Also, we present in this overview the last trend on the hyper-heuristic research field. After deep analysis of the literature, we concluded with these limitations and gaps:

- The grammar used does not cope with state-of-the-art PSO algorithms such as the ones proposed in (Wang et al. 2011; Liang et al. 2006; Mendes, Kennedy, and Neves 2004).
- None of the previous studies had investigated the use of crossover with PSO in the grammar which is proven to enhance the performance of PSO as in (Chen 2012), (Jong-Bae Park et al. 2010) .
- Adaption capability of the evolved algorithm is not considered in the grammar of any of the previous studies.

We will try in our study to overcome the previous limitations by introducing a more powerful and flexible grammar that utilizes state-of-the-art techniques for both GA, PSO and the possible hybridization of the two algorithms. This allows us to explore the various design options and to find the true capability of the hyper-heuristic methodology. So, the main objectives of our study can be summarized as follows:

- Adapting state-of-the-art research techniques in the grammar of the velocity updating strategy of PSO such as the ones proposed in (Mendes, Kennedy, and Neves 2004; Wang et al. 2011) and (Liang et al. 2006). This makes the hyper-heuristic capable of generating solutions that are suitable for different classes of problems.
- Designing a grammar that allows the incorporation of other search techniques and operators with PSO. Specifically, we aim to embed the crossover and mutation operators of Real-coded GA inside PSO. This provides the hyper-heuristic with the capability of exploring various design options and making use of the strength of the different techniques.
- Designing a grammar that is capable of generating hybrid PSO algorithms that have multiple velocity updating strategies. The reasoning behind this is that the cost of

generating solution for a given instance of problem by using the hyper-heuristic is very expensive, so, by making the generated algorithm adaptive, we allow the algorithm to maintain an acceptable performance even with small-mid changes in the problem specifications.

- Providing a friendly framework with high programmability for solvers that eases the development of a suitable solutions. The designed framework only requires the specification of the problem to be solved as input i.e. problem structure and objective function.

2. Chapter Two:

Literature Review

2.1. Introduction

A prominent survey in the hyper-heuristic field was accomplished in 2013 (E. K. Burke et al. 2013), this survey gives a wide overview of the various trends in the field in addition to a historical investigation on the origin of the idea of hyper-heuristic. After a wide review of the hyper-heuristic literature, we have drawn the following noticeable characteristics:

- The majority of the studies are concerned with selective hyper-heuristics.
- The studies that handle the generation of meta-heuristic components are scarce.
- More attention is paid recently to the relation between the hyper-heuristic methodology and the machine-learning field.

In this section, we provide a general overview of the current state of research in the hyper-heuristic field. Generative hyper-heuristics (GHH) can be classified based on the nature of their search space, which results in two classes: one that operates on the search space of the heuristic components, and one that operates on the search space of the meta-heuristic components, also another dimension of classification is possible. This alternative classification is based on whether the generative hyper-heuristic tries to put a solution for combinatorial optimization problems or for continuous optimization problems, but in this study we will adopt the first dimension.

2.2. Selective Hyper-Heuristic

In this section, we introduce three prominent techniques (in the selective hyper-heuristic field) and some recent studies that tackle them. These techniques are: Choice-Function, Markov-chain and Multi-armed Bandit.

2.2.1. Choice-Function Based Hyper-heuristics

The most common type of selective hyper-heuristics is the choice-function based hyper-

heuristic, which is a function that accesses the internal state of the hyper-heuristic and assesses all heuristics on which the hyper-heuristic operates. The choice-function is a polynomial constituted of multiple weighted terms; the weight reflects the importance of its corresponding term. The following function is an example of a choice-function:

$$f(H_k) = a * f_1(H_k) + b * f_2(H_k) + c * f_3(H_k, H_j)$$

Where:

a, b, c are weights which reflect the importance of each term. $f_1(H_k)$, $f_2(H_k)$, $f_3(H_k, H_j)$ are assessment functions of the heuristic (or meta-heuristic) H_k, where f_1 and f_2 may measure the recent performance of a heuristic, its execution time, or how many times it was invoked, and f_3 measures the efficiency of a consecutive application of a pair of heuristics.

After evaluating this choice-function for all candidates of heuristics, we could simply choose the heuristic with the maximum evaluation value as follows:

$$Max\big(f(H_k)\big)\ for\ k = 1 \dots n$$

The Roulette-wheel strategy may be used also to select a heuristic; where the Roulette-wheel strategy associates with each candidate heuristic a probability that is computed as follows:

$$P(H_i) = \frac{f(H_i)}{\sum_{j=1}^{n} f(H_j)}\ for\ \ i = 1 \dots n$$

Then, a heuristic is selected randomly according to the computed probabilities using the mechanism presented in algorithm 1.

The authors in (Cowling, Kendall, and Soubeiga 2001) introduced a *choice-function* as a polynomial composed of three terms f_1, f_2, and f_3. Term f_1 measures the previous performance of a given heuristic, term f_2 measures the performance of the consecutive appliance of two given heuristics, while f_3 is a term that improves diversification as it represents the time that elapsed since the last appliance of a given heuristic. (Drake, Özcan, and Burke 2012) proposed a modification of the previous choice function whose name is

'*modified choice-function*', this modification unifies the coefficients of f_1 and f_2 and correlates the unified coefficient with the coefficient of f_3 using an equation.

The authors in (Drake, Ozcan, and Burke 2015) further improved the performance of the '*modified choice-function*' by adding the crossover operator to the pool of low-level heuristics. The crossover operator's inputs were moderated by the hype-heuristic, where the first input solution was the current incumbent solution and the second input was a random solution maintained from a set of elite solutions by the hyper-heuristic.

The authors in (Maashi, Özcan, and Kendall 2014) proposed multi-objective choice-function based hyper-heuristic. The proposed hyper-heuristic has two ACs, namely, *the great deluge* and *the late acceptance* criteria. As a heuristic selection technique, the authors used a choice-function composed of two terms, where the first one values the intensification and the second one values diversification. This methodology was tested on the *walking Fish Groups* problem, and on the multi-objective design of *vehicle crashworthiness.*

2.2.2. Markov Chain based Hyper-heuristic

Markov chain is a statistical model that describes the sequence of possible states, where the probability of each state depends on a probability distribution attained in the previous state. This model can be a heuristic selection technique by equating the set of states to the set of heuristics that may be selected. Figure 9 illustrates an example of Markov chain model, where *A, B, C* are the heuristics to be selected, and the weights of the transitions are used to model the probabilities of moving among the different heuristics. Initially, all heuristics have equal probability to be selected, or we may select any state randomly. Then, at each decision, we have to choose the next state (heuristic) with respect to the probability distribution of the current state; one way to achieve this is given in Figure 8, which is based on the *Cumulative Density Function* (CDF).

1. Let S be a list of (i,p_i) for each state s_i with the transition probability p_i.

2. Sort S descendingly according to the states' probabilities.

3. Compute the cumulative probabilities for each state in S.

4. Generate random number *r*.

5. Choose the first state from S with a cumulative probability greater or equal to *r*.

Figure 8: Sampling technique based on CDF

For example: suppose that we are in state B, the randomly generated number $r = 0.6$ and the list of possible next states S are sorted descendingly [(A, 0.5), (C, 0.4), (B, 0.1)]. Then, by computing the cumulative probabilities, S becomes: [(A, 0.5), (C, 0.9), (B, 1)] and by choosing the state with a probability that is greater or equal to r (state C), we move to state C (apply heuristic C).

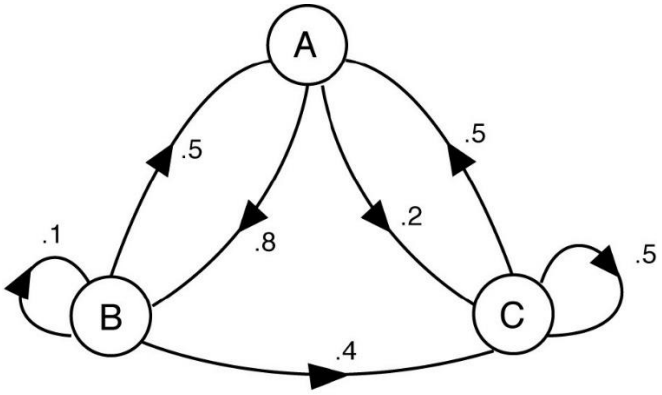

Figure 9: Example of markov-chain model.

In (McClymont and Keedwell 2011), the authors used Markov chain technique with online reinforcement learning that adapts the transitions' weights. The authors used Pareto dominance metric to measure the performance of the low-level heuristics. Pareto Dominance simply measures the quality of solutions produced (children solutions) over the quality of the parent solutions, and according to the measured performance, the weights in Markov chain is updated.

2.2.3. Multi-armed Bandit (MAB) Based Hyper-heuristic

The name comes from imagining a gambler at a row of slot machines having finite amount of coins (resources), the gambler has to decide which machines (choices) to play, how many times to play each machine and in which order to play them, and whether to continue playing the current machine or trying a different one. The objective of the gambler is to maximize the sum of rewards. The crucial trade-off that the gambler faces at each trial is between "exploitation" of the machine that has the highest expected payoff and "exploration" to get more information about the expected payoffs of the other machines. As an analogue, the previous description of the problem also applies to the hyper-heuristic selection problem where:

- **Resources:** In the hyper-heuristic methodology the only resource to be consumed is computation time.
- **Choices:** Are the pool of heuristics that may be applied.
- **Exploration and exploitation:** The trade-off in the search process, where the search algorithm has to decide whether to visit new areas or to intensify the search on already visited areas.

An *Adaptive Operator Selection* model based on the MAB named ‘Dynamic Multi-armed Bandit (DMAB)’ was proposed in (DaCosta et al. 2008). This model, DMAB, selects the arm according to its average reward and the number of invocations, with respect to the total number of invocations of all arms. In (Soria-Alcaraz et al. 2017) the authors used DMAB embedded in iterated-local search procedure, in each iteration a slight perturbation operator is applied to the incumbent solution, then a heuristic is selected using DMAB, then the selected heuristic is applied to the perturbed incumbent solution until no improvement is made (similarly to a gambler who keeps playing in the same slot machine until no reward is made). In this hyper-heuristic, an AC that accepts only improving solutions was used. DMAB was also used in (Sabar et al. 2015), but this time with an AC that is grammatically evolved and not humanly designed (later, we will discuss generative hyper-heuristic in more depth).

2.3. Heuristic-based GHH

In (E. K. Burke, Hyde, and Kendall 2012), the authors introduced an initial study of using GE as a hyper-heuristic. This hyper-heuristic was tailored for solving the One-*dimensional Bin Packing* problem. This hyper-heuristic evolves a population of local-search heuristics. The grammar used in this study is given in Figure 11. The <start> non-terminal represents the starting state which specifies how the pieces are removed from the bins by using one or more of the five terminals: highest-filled, lowest-filled, etc. and how the pieces are then repacked (the same terminals of the <repack> may be used for the initial configuration of the problem). The combinations of the procedures to select the bins and repack the pieces along with the different parameters values represent the different local search heuristics that can be generated. An example is given in Figure 10 where all pieces are removed from 10 random bins ignoring the bins that are 99.5% filled, then, the removed pieces are repacked by using the 'first_fit_decreasing' constructive heuristic.

```
<start>      →   <choosebins> remove_pieces_from_bins() <repack>
<choosebins> →   <type> | <type> <choosebins>
<type>       →   highest_filled(<num>,<ignore>,<remove>)
                 | lowest_filled(<num>,<ignore>,<remove>)
                 | random_bins(<num>,<ignore>,<remove>)
                 | gap_lessthan(<num>,<threshold>,<ignore>,<remove>)
                 | num_of_pieces(<num>,<numpieces>,<ignore>,<remove>)
<num>        →   2 | 5 | 10 | 20 | 50
<threshold>  →   average | minimum | maximum
<numpieces>  →   1 | 2 | 3 | 4 | 5 | 6
<ignore>     →   0.995 | 0.997 | 0.999 | 1.0 | 1.1
<remove>     →   ALL | ONE
<repack>     →   best-fit-decreasing  | worst-fit-decreasing
                 | first-fit-decreasing
```

Figure 11: Grammar for the bin-packing problem

```
Random-bins(10, 0.995, ALL)
remove_pieces_from_bins()
first_fit_decreasing()
```

Figure 10: Example of auto-generate heuristic for the bin-packing problem

In (Sabar, Ayob, and Kendall 2013), the authors employed the GE as a hyper-heuristic, but differently from the previous methodology, the grammar used here is more general in that it is not tailored to a specific problem class. Basically, this study divides the grammar system into three components: The first one is responsible for choosing the appropriate acceptance criterion from a set of options for the given problem, the second component is the list of candidate neighborhood structures i.e. a pool of local search operators, this second component is just a placeholder and it needs to be settled properly for the problem being solved. The last component is responsible for mixing different local search operators in one structure in different manners. So, for instance, if we want to solve the one-dimensional bin-packing problem by using this framework, we can do this simply by setting the placeholder of neighborhood structures list with the local search operators presented in (E. K. Burke, Hyde, and Kendall 2012)

In (Bader-El-Den, Poli, and Fatima 2009), the authors also used the grammar-based hyper-heuristic to solve the exam timetabling problem, but instead of using chromosomes of condones and a mapper procedure of GE, this hyper-heuristic evolves trees and performs the different GP operators on those trees. Each evolved tree in this framework is reduced from the complete derivation tree to a tree that contains only the terminal symbols (leaf nodes) of derivation tree.

In (Tan, Ma, and Mei 2018), the authors proposed genetic programming hyper-heuristic to automatically generate suitable heuristic for allocating containers in a cloud for homogeneous physical machines on online fashion with the objective being reduced accumulated power consumption.

2.4. Meta-heuristic based GHH

The authors in (Miranda, Prudêncio, and Pappa 2017) introduced the so-called *Hybrid Hyper-heuristic for Algorithm Design (H3AD)*. In H3AD, the authors employed GP to generate PSO algorithms. The grammar used in H3AD is presented in Figure 12. This grammar is rich since it considers several types of initialization procedure, several

topological structures of swarm, additionally, it utilizes different mutation operators which do not exist in the original definition of the PSO.

```
<INITIALIZATION>::= Uniform Random |
                    Opposition |
                    Non-Linear Simplex Model.

<TOPOLOGY>::= Star |
              Ring |
              Niche |
              Clubs |
              Cluster |
              Species |
              Dynamic Ring.
<MUTATION>::= Gaussian(<PROB_MUTATION>) |
              Cauchy(<PROB_MUTATION>) |
              Michelewicz(<PROB_MUTATION>) |
              Levy(<PROB_MUTATION>) |
              Random(<PROB_MUTATION>) |
              λ.
<PROB_MUTATION>::= 0.1 | 0.2 | ... | 1.0
<UDATE_VELOCITY>::= χ * ( υ + <EXP>)  |
                    ω * υ + <EXP>.
<EXP>::= <EXP> <OP> <EXP> |
         <VAR>.
<OP>::= + | - | * | /.
<VAR>::= x | pBest | gBest | r.
```

Figure 12: Grammar of H3AD

In (Miranda and Prudêncio 2017), a similar grammar to H3AD is used, but it only uses a random procedure to initialize the swarm in contrast with H3AD that employs three different procedures. In both studies, each evolved individual is evaluated by binding the individual tree in an algorithmic template such as the one given in Figure 13 , this binding results in a complete algorithm that can be tested. Both (Miranda, Prudêncio, and Pappa 2017) and (Miranda and Prudêncio 2017) constrain the velocity updating strategy in specific forms, namely: The interia-weight based strategy and the constriction coefficient based strategy. Both strategies are mathematically equivalent which adds more limitation to the variability that we can achieve in the design of the velocity updating strategy.

```
Input: size: size of swarm, nIt: number of iterations
  swarm = <INITIALIZATION> of size particles
  Evaluate all p in swarm
  gbest = Choose best according to <TOPOLOGY>
  for i = 0 to maxIt do
    for all p in swarm do
      <UPDATE-VELOCITY> of p
      Update position of p
      if (rand(0-1) < <PROB_MUTATION>) then
        Mutate p according to <MUTATION>
      end if
      Evaluate p
      Update pbest
    end for
    Update gbest according to <TOPOLOGY>
  end for
```


Figure 13: Template for H3AD framework.

In (Hong et al. 2018), the authors employed GP to generate the mutation operators of the Evolutionary Programming (EP) algorithm. In this study, the proposed grammar includes as terminal-set three different probability distributions for sampling, namely: The uniform, Normal distributions, and Cauchy distributions. And for the non-terminal-set additionally to the arithmetic operators, the cos, sin, log, sqrt operators are used.

In this study we are specifically concerned with GHH that operates on PSO, so we list the main insufficiencies and gaps found in (Miranda and Prudêncio 2017; Miranda, Prudêncio, and Pappa 2017):

- The grammar used does not cope with state-of-the-art PSO algorithms such as the ones proposed in (Wang et al. 2011; Liang et al. 2006; Mendes, Kennedy, and Neves 2004).
- None of the previous studies had investigated the use of crossover with PSO in the grammar which is proven to enhance the performance of PSO as in (Chen 2012), (Jong-Bae Park et al. 2010) .
- Adaption capability of the evolved algorithm is not considered in the grammar of any of the previous studies.

In the next sub-section, we present the new trends in the hyper-heuristic field.

2.5. New Trends in the Hyper-heuristic Field

In this section, we try to introduce the recent trends in the hyper-heuristic field. There are two noticeable trends with varying impacts. In the following subsections, we present them along with their related works.

2.5.1. Filtering the Heuristic Pool

This trend is concerned with reducing the number of heuristics that the high-level hyper-heuristic operates on since a very large pool of heuristics may increase the complexity of the hyper-heuristic. In (Soria-Alcaraz et al. 2017), the authors filtered the heuristics by using non-parametric test to rank the performance of low-level heuristics, where the performance is measured by using two metrics namely: *Evolvability* that measures the fitness of children solutions (which are produced by applying the measured low-level heuristic) in comparison with the parents solutions, and *landmarking* that measures the performance of low-level heuristics in the simplest form. Also, in (Gutierrez-Rodriguez et al. 2017), the authors applied heuristic filtering that is based on the concept of *feature selection* in the machine learning field.

2.5.2. Generation of Selective Hyper-heuristic

In (Swan, Özcan, and Kendall 2011) and (Swan et al. 2014), the authors notified to the recursive definition of the hyper-heuristic which implies that the pool of heuristics that hyper-heuristic operates on may contain hyper-heuristics in addition to the low-level heuristic, but a criticism of this definition of hyper-heuristic may appear because of the over generalization.

In (Sabar, Ayob, and Kendall 2014), the authors introduced a gene expression programming framework that evolves a population of selective hyper-heuristics. Each individual in the evolved population contains two components: Heuristic selection mechanism and

acceptance criterion. This framework achieved a good generalization and it was tested upon the HyFlex (Ochoa et al. 2012) framework which contains 6 problem domains.

In (Sabar et al. 2015), a similar work can be found but instead of evolving complete selective hyper-heuristic, the selection mechanism was fixed to the a dynamic multi-armed bandit mechanism DMAB (DaCosta et al. 2008), and only the AC mechanism is evolved. Similarly to the previous work, this framework was tested upon the HyFlex (Ochoa et al. 2012) framework with good results.

3. Chapter 3:
Contribution

3.1. Introduction

In our study, we overcame the limitations mentioned previously by introducing a more powerful and flexible grammar that utilizes state-of-the-art techniques for both GA, PSO and the possible hybridization of the two algorithms. This allows us to explore the various design options and look for the true capability of the hyper-heuristic methodology. The general structure of our framework is given in Figure 14 .

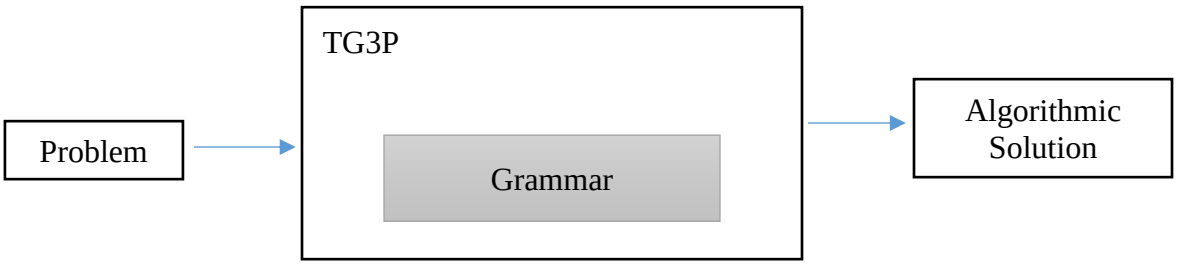

Figure 14: General structure of TG3P.

3.2. Tree Grammar-Guided Genetic Programming

The enumeration and evaluation of all possible algorithms from the grammar is not feasible because it is too expensive computationally, so, GP is used commonly to evolve a population of algorithmic solutions driven from the designed grammar. At each iteration, the known genetic operators which are namely: Selection, crossover and mutation are applied to the individuals of the population.

In the following sub-section, we will introduce the major headlines of our framework (TG3P). As discussed earlier, the grammar is the major design issue while developing a generative hyper-heuristic. So, we will start our discussion with our methodology for writing the grammar that we intend to use in our framework.

3.2.1. Grammar

Grammar constitutes the core of generative hyper-heuristic since it is the determinant of the nature of the algorithms being generated. While developing a grammar for generative hyper-heuristics, the following trade-off has to be considered:

"The exploration of more generic design decisions, versus the exploitation of more specific design aspects."

To clarify this trade-off more, consider two grammars: The first one has the capabilities of generating many variants of a specific meta-heuristic algorithm, while the other has the capabilities of generating limited number of variants of multiple meta-heuristic algorithms. Clearly, the first grammar favors exploitation over exploration since it allows us to discover specific design aspects, like: What is the suitable control parameters settlement to solve the problem? The second grammar favors exploration over exploitation, since it allows us to discover more general design decisions, like: What is the most suitable algorithm to solve the problem?

In order to make a balance between exploration and exploitation in this study, the major design decision of "what algorithm should be used to solve the problem?" is solved by determining the algorithms beforehand, which are PSO and GA. This hyper-heuristic framework is devoted for solving continuous optimization problems. The reason behind using PSO and GA is their widespread use and competitive results through the literature.

In each of the subsequent sub-sections, we first analyze state-of-the-art techniques, then, we attempt to devise a grammar that is designed in accordance with those techniques. To devise our grammar, we disassemble the studied techniques to their basic constructs, then, a classification for each construct as terminal or as non-terminal is performed, and finally a set of production rules is defined that models the relations between the different constructs.

3.2.1.1. **Particle Swarm Optimization**

The original PSO has a pretty good convergence ability, but it suffers the demerit of premature convergence. To overcome this problem, there are many attempts that try to improve the performance, specifically, the exploration capability of the PSO algorithm. Below we mention some of the techniques that try to improve the performance of the PSO algorithm. In the next subsections, we discuss some of those techniques.

- **Swarm Topology**

The topology constrains the nature of the communication that may occur in the swarm, as a result, it affects how the velocity of the particles in the swarm is updated. Some of the well-recognized topologies are: Star topology, ring topology, focal topology, Von-Neumann topology, and additionally as a high level topology, the swarm may be structured as clusters or hierarchies. Some work that investigate the effect of the topology in the search capability can be found in (Kennedy and Mendes 2002). Below a brief description of each topology is given:

- Ring topology: In this topology, each particle has only two neighbors. The convergence of the algorithm through using this topology is slow and it is suitable for complex multi-modal problems. Sometimes this topology is referred to as the star topology.

- All topology: This topology is the first and the most commonly used topology where each particle is connected to all other particles. This topology has fast convergence rate and it is suitable for simple uni-modal problems.

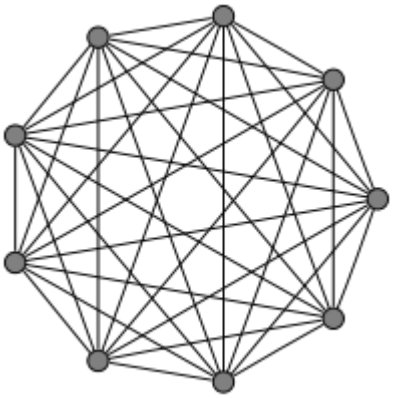

Figure 15: ALL topology.

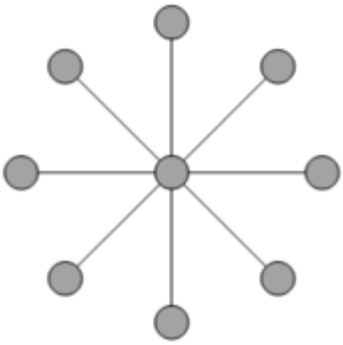

Figure 16: Focal topology.

- Focal topology: In this topology, all particles are connected to a focal particle, aka proxy particle.
- Cluster topology: In this topology, the swarm is divided into clusters, usually 4, and each cluster communicates with the other clusters by using a representative member from the inside of the cluster. The internal structure of the cluster may be ring, star or any other topology, but the star is usually used.

- Von-Neumann topology: In this topology, each particle is connected to four neighbors in the four directions, consequently, this makes the particles on the borders connected to each other. This topology had achieved good results, and sometimes it is referred to as the square topology.

- Tree topology: In this topology, the swarm is organized as undirected tree with a predetermined branching factor.

More on typologies and their analysis can be found in (Mendes, Kennedy, and Neves 2004). In this study, we adopt all the previously mentioned topologies, additionally, we propose a new topology for the swarm. The new topology's name is 'Wheel', it allows a flow of information in the swarm that is faster than the ring and the focal topology and slower than the other topologies (in term of swarm degree). The wheel topology is illustrated in Figure 19.

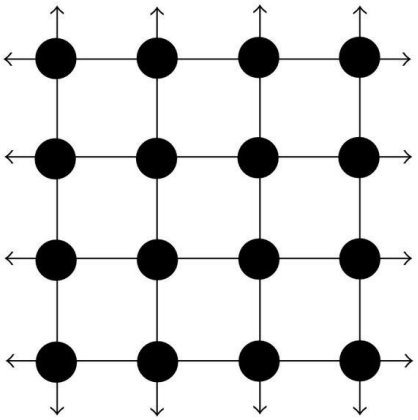

Figure 17: Von-Neumann topology.

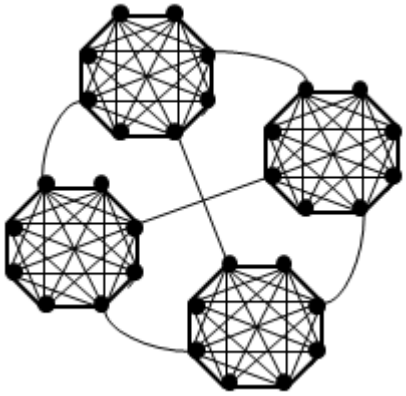

Figure 18: Cluster topology

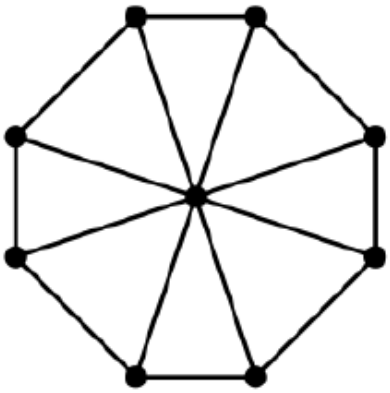

Figure 19: Wheel topology

- **Velocity Updating Strategy**

Several modifications to the original updating strategy of PSO are proposed in the literature. Some modifications suggest adding control parameters to the formula, while others have focused on the bases in which the velocity vector gets updated. In the next sub-section, we mention some of these modifications.

a. Interia-weight strategy

The original updating equation adopts the past experience as it is, as a consequence, this may hugely affect the search result. So, in order to have more control over the past experience,

the authors in (Y. Shi and Eberhart 1998) introduced interia-weight parameter ω so the updating strategy becomes as follows:

$$V_i^d \leftarrow \omega * V_i^d + c_1 * rand_i^d\left(Pbest_i^d - X_i^d\right) + c_2 * rand_i^d\left(Gbest^d - X_i^d\right)$$
$$X_i^d \leftarrow X_i^d + V_i^d$$

b. *Comprehensive learning PSO*

Another updating strategy variant named *Comprehensive Learning Particle Swarm Optimization* (CLPSO) is proposed in (Liang et al. 2006). CLPSO employs a learning strategy in the velocity updating strategy. CLPSO eliminates the *gbest* from the equation, also it makes it possible for each particle in the swarm to be influenced by all *pbest* of all other particles. So, in CLPSO the updating strategy becomes as follows:

$$V_i^d \leftarrow \omega * V_i^d + c_1 * rand_i^d\left(Pbest_{f_{i(d)}}^d - X_i^d\right)$$

$$X_i^d \leftarrow X_i^d + V_i^d$$

$$X_i^d \leftarrow MIN(X_{max}^d, MAX(X_i^d, X_{min}^d))$$

Where $f_{i(d)}$ is a tournament selection procedure that chooses from which particle the i_{th} particle will learn for the d_{th} component. While V_{max}^d, V_{min}^d are thresholds that control the minimum and maximum velocity of the particles in the swarm. CLPSO provides much higher exploration ability than the original PSO.

c. Difference based velocity updating strategy

This strategy, according to the author in (Wang et al. 2011), is used to quickly adapt to changes in the different optimization phases, so, it does not build the velocity vector incrementally, but it recombines the velocity vector totally based on some difference information. The equation of DbV is given as follows:

$$Viad_i^d \leftarrow X_k^d - X_j^d,$$
$$c = N(0.5, 0.2).$$
$$V_i^d \leftarrow c \times Viad_i^d + c \times \left(pbest_i^d - X_i^d\right).$$
$$V_i^d = \min\left(V_{\max}^d, \max\left(V_i^d, V_{\min}^d\right)\right),$$

Where X_j^d, X_k^d are the d_{th} variables of two randomly selected particles, and $N(0.5,0.2)$ represents one randomly generated number according to the Gaussian distribution with mean 0.5 and standard deviation 0.2.

d. Estimation based velocity updating strategy

The authors in (Wang et al. 2011) also proposed an Estimation based Velocity updating strategy (EbV). This strategy is developed to achieve a high convergence rate, specifically for complex multi-modal problems. This strategy makes estimation of the distribution of the population and updates the population distribution according to this estimation. The EbV is described as follows:

$$c = \frac{(D-1)N(0,1)}{D} + \frac{C(0,1)}{D}.$$
$$V_i^d \leftarrow \left(mean_i^d - X_i^d\right) + \frac{c}{\sqrt{3}}\sqrt{\left(pbest_i^d - mean_i^d\right)^2 + \left(X_i^d - mean_i^d\right)^2 + \left(X_k^d - mean_i^d\right)^2}.$$
$$V_i^d = \min\left(V_{\max}^d, \max\left(V_i^d, V_{\min}^d\right)\right),$$

Where c is a random coefficient derived from a mixed Gaussian and Cauchy distribution, $mean_i^d$ is the mean of the 20% best particles in the swarm.

e. Modified CLPSO

The modified CLPSO, proposed in (Wang et al. 2011), which is named by the author as PSO-CL-Pbest is developed to increase the exploitation ability of CLPSO by reintroducing the $pbest$ of the updated particle to the equation. But of course, increasing the exploitation means decreasing the exploration ability to some degree. The PSO-CL-Pbest strategy is described as follows:

$$V_i^d \leftarrow w \times V_i^d + 0.5 \times c \times rand_i \times \left(pbest_{f_i(d)}^d - X_i^d + pbest_i^d - X_i^d\right).$$

$$V_i^d = \min\left(V_{\max}^d, \max\left(V_i^d, V_{\min}^d\right)\right).$$

f. Fully informed PSO

In (Mendes, Kennedy, and Neves 2004), the authors introduced the *Fully Informed Particle Swarm Optimization* (FIPSO). Unlike the original PSO, FIPSO extends the source of information that the particle may learn from and that include in the original PSO the local best and the global best while in FIPSO it includes all the neighboring particles. FIPSO is dependent on the underlying topology of the swarm. Multiple alternative implementations of FIPSO exist such as the weighted FIPSO where each neighbor contributes to the updating strategy proportionally with its fitness. FIPSO can be described as follows:

$$V_i^d \leftarrow \chi * (V_i^d + c * (P_m^d - X_i^d))$$

$$X_i^d \leftarrow X_i^d + V_i^d$$

$$P_m \leftarrow \frac{\sum_{k \in N} W(k) * c_k * Pbest_k}{\sum_{k \in N} W(k) * c_k}$$

$$c \leftarrow c_1 + c_2 + \cdots + c_n$$

Where $W(k)$ is a function that weights the influence of each neighbor particle of the i_{th}

particle. For instance, this function may return the fitness value of the given particle, or it may return a constant value for all particles, in this case all particles are treated equally.

Through analyzing the previous velocity updating strategies, we can notice a difference in the prospect of what particles to consider while updating the current particle. The following are the most common choices:

- The global best.
- The local best
- Aggregation on the local best of the Neighboring particles.
- Aggregation on the local best of a subset of particles.
- Local best of a random particle.

The previous velocity vector exists in all the previous updating strategies except the EbV strategy. The control parameters also represent a major design decision in each of the previous strategies. The observed parameters and coefficients include:

- Interia-weight
- Constriction coefficients
- Acceleration coefficients
- Max and Min velocity vectors

Thus, the proposed grammar is given in Figure 20. In this grammar, the VelocityPool non-terminal specifies the number of velocity updating strategies that a generated algorithm would have, the number of strategies is limited between one and four. The *Influencer* non-terminal specifies how an influencer vector is produced to update a particle and to which

degree (based on the Φ coefficient). The *Scope* non-terminal specifies what subsets of particles are used to compute the influencer vector. Two scopes are used:

- Neighborhood: This scope implies that the whole neighborhood of a particle is used to update it.

- Elite: This scope implies that only the elite particles in the swarm are used to update a particle. Different values for the elite subset size are possible in this grammar.

The *Influencer* non-terminal specifies how a position vector that is used to update the particle is obtained. To obtain it, we can use one of the following ways:

- Average: Compute the average of the local best of a subset of particles.

- Max: Choose the local best of the fittest particle from the provided set of particles.

- Random: Choose the local best of randomly selected particle from the provided set of particles.

- Local: Choose the local best of the updated particle.

The output vector from the influencer non-terminal in each method is highly dependent on the scope non-terminal (except for the Local choice which always return the Pbest of the updated particle). For example, $Max(Elite)$ is interpreted as: Select the best position from the elite particles subset, while $Random(Neighborhood)$ would mean: Select a random particle position from the whole Neighborhood. The concrete meaning of the Neighborhood scope is determined by the type of topology used in the current version of PSO. The symbol ω is the interia-weight, and it is a uniform random number between [0.4- 1), similarly, the acceleration coefficient Φ is a random number between (1- 2]. In this grammar, we constraint the formula to maximally three terms plus the previous velocity.

$VelocityPool \leftarrow Velocity\{1,4\}$
$Velocity \leftarrow \omega * Velocity_i + Terms$
$Terms \leftarrow Term + Term + Term \mid Term + Term \mid Term$
$Term \leftarrow U(0,1) * \phi * (Influencer - Xi)$
$Influencer \leftarrow Max\ (Scope) \mid Average\ (Scope) \mid Random\ (Scope) \mid Local$
$Scope \leftarrow Elite_{\ eliteSize} \mid Neighborhood_{\ i}$
$eliteSize \leftarrow 5\% \mid 10\%$
$\omega \leftarrow U(0.4, 1)$
$\Phi \leftarrow U(1, 2)$

Figure 20: Proposed grammar for VUS

- **Adaptive operator selection**

In PSO, different algorithm's configurations should be considered as the search proceeds. For example, in the early stages of the search, more exploration should be done to enlarge the covered search space, while in later stages, more exploitation should be done to tune the evolved solutions. To achieve this requirement, different approaches exist in the literature, some of them employ time-varying inertia weight and acceleration coefficients, such work can be found in (Nickabadi, Ebadzadeh, and Safabakhsh 2011). Other approaches employ multiple velocity updating strategies which are selected adaptively based on the current state of the search, an example of this is SLPSO which can be found in (Wang et al. 2011). In SLPSO, the following mechanism is used:

For each strategy, assign an execution probability $proSTR_i$ to determine the probability that the ith strategy gets selected to update each particle. Initially, assign equal probabilities for all strategies, that is $proSTR_i = 0.25, for\ i = 1 \ldots n$, and set an accumulators for each strategy $S_i = 0\ , for\ i = 1 \ldots n$. At each generation, the particles are sorted based on their fitness values. Then, each particle is assigned a weight $w_j = log\ (ps - j + 1)/\big(log(1) + \ldots + log(ps)\big)$ for $j = 1 \ldots ps$. Finally, the weights are added to the accumulators of their associated updating strategies. After a fixed number of generations Gs, the following rule is used to update the execution probability of jth updating strategy:

$$pro\acute{S}TR_J = (1 - \alpha) proSTR_j + \alpha\, S_j/Gs,$$

$$proSTR_j = pro\acute{S}TR_j / \left(pro\acute{S}TR_1 + pro\acute{S}TR_2 + \cdots + pro\acute{S}TR_n\right)$$

Where $pro\acute{S}TR_j$ is the temporal execution probability; α is the learning coefficient which is used to control the updating proportion.

In our framework, the adaptive mechanism of SLPSO is used. According to our knowledge, there is not any previous study that addresses the adaptability aspect of the generated algorithm.

Other techniques that attempt to enhance the performance of PSO exist in the literature such as the way the algorithm may initialize the positions and velocity vectors of the particles, initialization of coefficients, time varying coefficients, migration of particles, etc. Such works can be found in (Vandenbergh and Engelbrecht 2006), (Konstantinos Parsopoulos and Michael N. Vrahatis 2002), (Gang, Wei, and Xiaolin 2012).

3.2.1.2.**Real-coded GA**

Many strategies that try to hybridize PSO with other search techniques exist in the literature, but our focus here will be in the possible hybridization of PSO with the *Genetic Algorithm (GA),* more precisely, with the *Real Coded Genetic algorithm* (RCGA). RCGA represents the solutions as chromosomes of real numbers unlike the standard GA. Similarly to GA, RCGA does not preserve a memory. RCGA search strategy is based on the same concepts of GA, which are: Selection, inheritance, and variation, for more information review page 8. Later in this section, the terms GA and RCGA are used interchangeably.

- **RCGA operators**

In addition to the single, two, and N point crossovers in the traditional GA, several types of crossover for RCGA are presented in the literature, in the following sub-sections, we mention some of them.

a. Parent Centric Crossover

In (Lozano et al. 2004), a Parent Centric (PBX-α) crossover was introduced. PBX-α is described as follows: If we have two real-coded chromosomes X = $(x1 \dots xn)$ and Y = $(y1 \dots yn)$, $(xi, yi \in [a_i, b_i] \subset R, i = 1 \dots n)$, X and Y are selected to undergo the crossover operator, PBX-α generates randomly one of two possible offsprings: Z_1 = $(z_1^1 \dots z_n^1)$ or Z_2 =$(z_1^2 \dots z_n^2)$, where z_i^1 is a uniformly sampled random number from the interval $[l_i^1, u_i^1]$ with:

$$l_i^1 = max(a_i,\ x_i - I.\alpha), \qquad u_i^1 = min(b_i,\ x_i + I.\alpha)$$

And z_i^2 is sampled from the range $[l_i^2, u_i^2]$with:

$$l_i^2 = max(a_i,\ y_i - I.\alpha), \qquad u_i^2 = min(b_i,\ y_i + I.\alpha)$$

Where:

$$I = |x_i - y_i|.$$

According to the author, PBX-α generates solutions that are closer to their parents, and its diversification ability can be increased by adjusting the α parameters to higher values. Additionally, PBX-α is self-adaptive since it adapts according to the distance between the parents solutions.

b. Multi parent crossover

The authors in (Elsayed, Sarker, and Essam 2011) introduced Multi Parent Crossover (MPC). The procedures of MPC are as follows:

I. Select three different solutions.
II. Sort them in an ascending manner according to their fitness values.
III. Generate three offsprings as follows:

$$o_1 = x_1 + \beta * (x_2 - x_3)$$
$$o_2 = x_2 + \beta * (x_3 - x_1)$$
$$o_3 = x_3 + \beta * (x_1 - x_2)$$

From the above equations, it can be noticed that the first and the third offspring are generated with the aim of being positioned in more promising areas in the search space, while the second offspring is generated with the aim of keeping diversity in the swarm.

c. *Linear crossover*

Linear crossover (LC) is based on the concept of linear combination of vectors. Let the vectors $\overrightarrow{v_1}, \overrightarrow{v_2}, \overrightarrow{v_n}$ be vectors in $\mathbb{R}^n$ and $c1, c2, \cdots, cn$ be scalars. Then, the vector $\vec{b}$, where $\vec{b} = c_1 * \overrightarrow{v_1} + c_2 * \overrightarrow{v_2} \ldots + c_n * \overrightarrow{v_n}$ is called a linear combination of $\overrightarrow{v_1}, \overrightarrow{v_2}, \overrightarrow{v_n}$. The scalars $c1, c2, \cdots, cn$ are called the "weights". So, linear crossover can be applied to many parent solutions, but two parents are usually used. In case of two-parent linear crossover, the weights are usually set to 0.5 (uniform), however, different weights could be used to favor the fittest parent.

d. Pbest crossover

Pbest crossover operates on single parent, and it produces a single offspring by computing the average of the current position of the parent and its best local position (Pbest) as follows:

$$X` = (X_i + Pbest_i)/2$$

This crossover clearly emphasizes the exploitation capability of the algorithm.

e. Mutation operators

Three types of mutation will be used in our study: Gaussian mutation, Cauchy mutation and

Random (uniform) mutation. In the three types, we perform mutation by adding a randomly sampled vector (according to the type of mutation) with a size equal to the size of the mutated particle, then we add this vector to the current position of the mutated particle.

f. Selection and replacement strategies

Selection and replacement strategies specify how particles are selected to undergo crossover, and which particles are selected to be replaced by the new offspring individuals. For both selection and replacement, Tournament-based selection is used in this study. In tournament selection, a subset of size k is selected randomly from the swarm, then the best/worst particle from the subset is selected to undergo crossover or to be replaced.

- **Mutation and Crossover Probabilities**

The probability of performing crossover and mutation at any iteration is determined by the variables PC and PM, different values for both probabilities should be considered depending on the problem being solved. So, the value of PC and PM will be automatically tuned by the hyper-heuristic.

Accordingly, the grammar that we use for RCGA in our hyper-heuristic framework is given in Figure 21.

CROSSOVER → LC | MPC | PBX-α. | Pbest
Mutation →Gaussian | Cauchy | Random
PC → 0 | … | 1
PM → 0 | … | 0.4

Figure 21: Grammar for RCGA

3.2.1.3.Hybridization Scheme

As we will generate hybrid PSO and RCGA algorithms, we need to decide upon the hybridization scheme of the two algorithms i.e. how do they corporate together? What is the state that they share? In the literature, two main approaches are found, one approach is to treat each algorithm as a black-box while providing some degree of interaction between them. While the second approach is to blend the two algorithms into one body and in order to achieve this, the two algorithm must share some state variables. More on the possible hybridization between PSO and GA can be found in (Thangaraj et al. 2011).

In (Kao and Zahara 2008), the author proposed a hybridization mechanism for PSO with the Genetic Algorithm. Figure 22 illustrates how this mechanism works. The repeated cycle of the algorithm starts by sorting all population, then, the fittest half is updated by applying the GA operators, namely crossover (linear combination of vectors) and mutation, while the other half is updated using PSO operators. The cycle is repeated until convergence. Apparently, in this framework both algorithms (PSO and GA) are independent from each other.

Similarly in (X. H. Shi et al. 2005), the authors proposed the so-called *Variable Population Size Genetic Algorithm (VPGA)*. In this framework, VPGA and PSO are run independently initially, then, at equal intervals, a migration of N individuals is performed from the swarm of PSO to VPGA, and similarly from the population of VPGA to PSO.

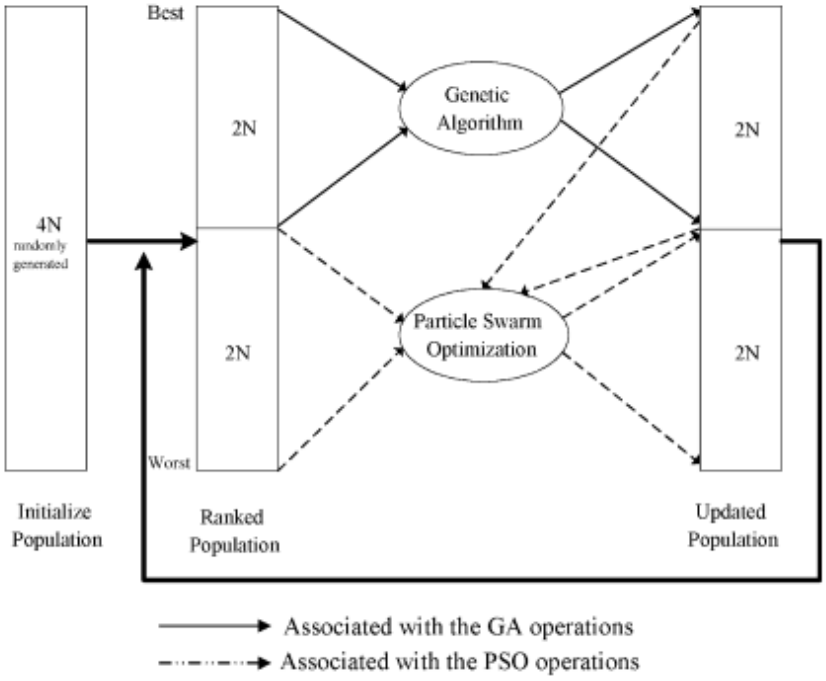

Figure 22: Hybrid PSO and RCGA

Differently from the two previous studies, the works of (Chen 2012), (Pant, Thangaraj, and Abraham 2007), and (Jong-Bae Park et al. 2010) use GA operators alongside the operators of PSO. Both hybridization approaches yield an improved and competitive results compared to the original algorithms. In this study, similarly to the second approach, both PSO an RCGA are generated as single component, where they share the same state i.e. Population, Current iteration, etc.

The complete grammar that we use in our framework is given in Figure 23, and the algorithmic template is given in Figure 24.

Swarm ← ALL | Focal | Ring | Von-Neumann | 4-Cluster | Wheel | Tree
EliteSize ← 5% | 10%

$VelocityPool \leftarrow Velocity\{1,4\}$
$Velocity \leftarrow \omega * Velocity_i + Terms$
$Terms \leftarrow Term + Term + Term \mid Term + Term \mid Term$
$Term \leftarrow U(0,1) * \phi * (Influencer - Xi)$
$Influencer \leftarrow Max\ (Scope) \mid Average\ (Scope) \mid Random\ (Scope) \mid Local$
$Scope \leftarrow Elite_{\ eliteSize} \mid Neighborhood_{\ i}$
$\omega \leftarrow U(0.4,1)$
$\Phi \leftarrow U(1,2)$

CROSSOVER → LC | MPC | PBX-α. | Pbest
Mutation →Gaussian | Cauchy | Random
PC → 0 | … | 1
PM → 0 |… | 0.4

Figure 23: Full Grammar that is used in this study

Hybrid PSO and RCGA Algorithm:

```
N: Swarm size
T: Number of generation
Swarm ← Random initialization of n particles
Gbest ← Null
For t = 0 to T do
    For each p in Swarm do
        Evaluate p
        Update the Pbest of particle p and the Gbest
    End for
    For each p in Swarm do
        Select a velocity updating strategy from the <VelocityPool>
        Apply the selected strategy upon p
    End for
    If U(0,1) < <PC> then
        Select parents from Swarm using Tournament-based Selection
        Crossover parents according to <Crossover>
        Replace parents using Tournament-based Selection
    End if
    For each p in Swarm do
        If U(0,1) < <PM> then
            Mutate p according to <Mutation>
        End if
    End for
End for
Return Gbest
```

Figure 24: Algorithmic template of TG3P.

3.2.2. **Population initialization**

The population in our designed GP is initialized randomly based on the proposed grammar (Figure 23: Full Grammar that is used in this study). And we use an elitism by keeping the best individual in the population intact from being modified by the genetic operators (mutation and crossover) during the algorithm runtime.

3.2.3. Genetic operators

In our study, the normal sub-tree crossover and mutation are used, but unlike the standard sub-tree crossover and mutation, the selection probabilities of the crossover and mutation points are not uniformly random, instead, we propose a custom probability distribution that assigns a probability of selecting a node is based on its level in the tree, this probability distribution favors nodes that are in the low-levels in the tree. The procedure used to generate this distribution is given in Figure 25.

N ← Maximum depth of tree

Generate random list L of size N.

Sort L ascendingly.

Probs ← $L_i \Big/ \sum_{i=1}^{n} L_i$

Figure 25: Procedure to generate probability distribution for selecting mutation/crossover points.

In our designed GP, we apply crossover at each iteration, but for the mutation, we use time varying probability. The reasoning behind this is to emphasize exploration at the earlier stages of the algorithm, while emphasizing exploitation in the latter stages. The equation used to compute the probability of mutation at each iteration is given as follows:

$$PM(t) = \frac{(T-t)}{T}(MAX_{PM} - MIN_{PM}) + MIN_{pm}$$

Where: T is the maximum number of iteration, MAX_{PM}, MIN_{PM} are the maximum and minimum probabilities of doing mutation.

4. Chapter Four:
Evaluation

4.1. Introduction

In this chapter, we present the benchmarks that we used and our experimental settings, these settings are mainly specific to our hyper-heuristic (Tree-based Grammar-Guided Genetic Programming TG3P) and to the generated algorithms. Then, we present the alternative PSO variants that we compare with. Finally, we present our results and their discussion.

4.2. Test Functions

All tests in this study are conducted by the single-objective and unconstrained continuous test functions of the Deap framework (Fortin 2012). The list of the test functions in Deap is:

a) Cigar

$$\text{Minimize:} \quad f(\mathbf{x}) = x_0^2 + 10^6 \sum_{i=1}^{N} x_i^2$$

b) Plane

$$\text{Minimize:} \quad f(\mathbf{x}) = x_0$$

$$x_i \in [-100, 100]$$

c) Sphere

This function is simple and it is easy to solve. The Sphere function is given as follows:

$$\text{Minimize:} \quad f(\mathbf{x}) = \sum_{i=1}^{N} x_i^2$$

d) Ackley

Ackley's function has one narrow global optimum basin and many minor local optima. The Ackley's function is given as follows:

Minimize:

$$f(\mathbf{x}) = 20 - 20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) + e - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right)$$

$$x_i \in [-15, 30]$$

e) Bohachevsky

The Bohachevsky's function is uni-modal problem and it has a bowl shape. The Bohachevsky's function is given as follows:

Minimize:

$$f(\mathbf{x}) = \sum_{i=1}^{N-1}(x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_{i+1}) + 0.7)$$

$$x_i \in [-100, 100]$$

f) H1

Simple two-dimensional function containing several local maxima. The H1 function is given as follows:

Maximize:

$$f(\mathbf{x}) = \frac{\sin(x_1 - \frac{x_2}{8})^2 + \sin(x_2 + \frac{x_1}{8})^2}{\sqrt{(x_1 - 8.6998)^2 + (x_2 - 6.7665)^2} + 1}$$

$$x_i \in [-100, 100]$$

g) Schwefel

.

The complexity of Schwefel's function is due to its deep local optima being far from the

global optimum. It will be hard to find the global optimum if many particles fall into one of the deep local optima. The Schwefel's function is given as follows:

Minimize: $$f(\mathbf{x}) = 418.9828872724339 \cdot N - \sum_{i=1}^{N} x_i \sin\left(\sqrt{|x_i|}\right)$$

$$x_i \in [-500, 500]$$

h) Griewank

Griewank's function has a $\prod_{i=1}^{N} \cos\left({x_i}/{\sqrt{i}}\right) + 1$ component which cause linkages between variables, so reaching the global optimum becomes more difficult. The Griewank's function is given as follows:

Minimize: $$f(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Range: $x_i \in [-600, 600]$

i) Himmelblau

The Himmelblau's function is multimodal with 4 defined minimums. The Himmelblau's function is given as follows:

Minimize: $$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Range: $x_i \in [-6, 6]$

In this study, we set in all experiment the dimensions of all test functions to be 30. Except for the H1 and the Himmelblau functions which will have dimension of 2. Each generated algorithm is evaluated in 10 independent runs for any problem.

4.3. Experimental settings

The settings of our hyper-heuristic (TG3P) and the settings of the generated Hybris PSO and RCGA algorithms (during the training phase) are given in Table 1 and Table 2.

Table 1: Settings of TG3P

Variables	**TG3P**
Iterations	200
Population size	300
Crossover Probability	0.9
Min Mutation Probability	0.1
Max Mutation Probability	0.5
Tournament size	10
Elite size	1

Table 2: Settings of the generated algorithms.

Variables	**Hybrid PSO & RCGA**
Iterations	200
Swarm size	50
Tournament size	7

4.4. Comparison

In this study, we compare the best generated algorithms with SLPSO (Wang et al. 2011) and CLPSO (Liang et al. 2006) under the same test conditions. For each test function, the average, the standard deviation, and the max of 30 independent runs are used to compare our best generated algorithm with CLPSO and SLPSO. For both the generated algorithms, CLPSO, and SLPSO a swarm of size 50 is used and the maximum number of function evaluation is set as 5000, and the bounds on the velocity vector (VMAX, VMIN) are set to 10% of (b - a) where $x_i \in [a, b]$. The results for each test function are given in Table 3 and Table 4.

Table 3: Mean (with std.) of 30 runs of CLPSO, SLPSO, and TF3P.

	CLPSO		SLPSO		TG3P	
Porblem	Avg.	Std.	Avg.	Std.	Avg.	Std.
Schwefel	8.50E+3	4.67E+2	**1.37E+3**	4.14E+2	8.58E+3	9.07E+2
H1	1.04E+0	3.84E-1	1.96E+0	2.27E-2	**2.00E+0**	4.44E-16
Cigar	3.66E+10	6.20E+9	3.25E+9	1.22E+9	**7.57E+8**	2.89E+8
Ackley	1.81E+1	3.95E-1	1.16E+1	1.07E+0	**1.10E+1**	1.11E+0
Sphere	3.63E+4	5.10E+3	3.09E+3	8.57E+2	**6.87E+2**	2.44E+2
Plane	**-1.00E+2**	3.42E-2	**-1.00E+2**	0.00E+2	**-1.00E+2**	'0.00E+2
Bohachevsky	1.08E+5	1.58E+4	9.67E+3	3.56E+3	**1.10E+3**	5.09E+2
Griewank	3.29E+2	5.15E+1	2.93E+1	8.70E+0	**4.06E+0**	1.98E+0
Himmelblau	5.11E-2	8.47E-2	4.77E-5	5.21E-5	**5.26E-32**	1.97E-31

Table 4: Max and Min of 30 runs of CLPSO, SLPSO and TG3P.

	CLPSO		SLPSO		TG3P	
Problem	Max	Min	Max	Min.	Max	Min
Schwefel	9.32E+3	7.70E+3	2.39E+3	5.33E+2	1.01E+4	6.63E+3
H1	1.74E+0	4.33E-1	2.00E+0	1.90E+0	2.00E+0	2.00E+0
Cigar	4.84E+10	2.43E+10	5.91E+9	1.63E+9	1.43E+9	2.69E+8
Ackley	1.89E+1	1.73E+1	1.36E+1	9.17E+0	1.31E+1	8.61E+0
Sphere	4.55E+4	2.53E+4	5.11E+3	1.80E+3	1.41E+3	3.71E+2
Plane	-9.98E+1	-1.00E+2	-1.00E+2	-1.00E+2	-1.00E+2	-1.00E+2
Bohachevsky	1.40E+5	6.85E+4	1.68E+4	4.01E+3	2.39E+3	4.68E+2
Griewank	4.28E+2	2.10E+2	5.31E+1	1.26E+1	1.08E+1	1.43E+0
Himmelblau	4.61E-1	6.67E-4	2.48E-4	2.06E-6	7.89E-31	0.00E+2

4.5. Discussion

The results shown in Table 3 and Table 4 clearly emphasize the outperforming performance of the generated hybrid PSO and RCGA against SLPSO and CLPSO. We achieved better results in seven functions from the nine tested functions. Our poor results of the Schwefel's

function is possibly due to using maximum number of iteration during the training process which is different from the maximum number of iteration during the evaluation process. And the reason behind using few number of iterations in the training process is the expensive computational cost. In Table 5, we show the major design features of the best generated algorithms for each solved problem.

Table 5: Features of generated hybrid PSO & RCGA.

Problem	Topology	Number of VUSs	Mutation	Crossover	Elite Size
Schwefel	Grid	1	Cauchy/0.4	MPC/0.7	10%
H1	All	1	0.0	Linear/0.7	5%
Cigar	4-Cluster	1	Gaussian/0.05	Linear/0.2	10%
Ackley	Grid	1	Gaussian/0.2	PBXa/0.7	5%
Sphere	Tree	2	Gaussian/0.05	Linear/0.2	10%
Plane	Ring	4	0.0	Pbest/1.0	5%
Bohachevsky	Tree	3	Gaussian/0.2	PBxa/0.2	10%
Griewank	Grid	1	Gaussian/0.2	PBxa/1.0	10%
Himmelblau	Tree	2	Cauchy/0.05	MPC/0.7	5%

5. Conclusion

Hyper-heuristic is an emerging methodology that aims to solve optimization problems with a high level of generality. Two main classes of hyper-heuristic exist in the literature: Selective hyper-heuristics and generative hyper-heuristics. In this study, we focus on developing a general, yet efficient generative hyper-heuristic framework. In this study we proposed a modified Tree-based Grammar-guided Genetic Programming (TG3P) as a hyper-heuristic that operate on a grammar that has the capability of generating adaptive hybrid Particle Swarm Optimization (PSO) and Real-coded Genetic Algorithm (RCGA) for solving contentious optimization problems. In this study we compared our results with SLPSO (Wang et al. 2011) and CLPSO (Liang et al. 2006), and the results show that our proposed hyper-heuristic has very competitive efficiency. In the future we aim to further support other classes of problems such as, the constrained and the unconstrained multi-objective optimization problems. Also we will try to consider other aspects of Particle Swam Optimization (PSO) in the grammar such as, the Island-model (Izzo, Ruciński, and Biscani 2012), new topologies, etc.

References

Bader-El-Den, Mohamed, Riccardo Poli, and Shaheen Fatima. 2009. "Evolving Timetabling Heuristics Using a Grammar-Based Genetic Programming Hyper-Heuristic Framework." *Memetic Computing* 1 (3): 205–19. https://doi.org/10.1007/s12293-009-0022-y.

Beheshti, Zahra, and Siti Mariyam Hj Shamsuddin. 2013. "A Review of Population-Based Meta-Heuristic Algorithm" 5 (1): 35.

Bois, Paul Du, Clifford C Chou, Bahig B Fileta, Tawfik B Khalil, Albert I King, Hikmat F Mahmood, Harold J Mertz, Jac Wismans, Priya Prasad, and Jamel E Belwafa. 2004. "Vehicle Crashworthiness and Occupant Protection," 388.

Boussaïd, Ilhem, Julien Lepagnot, and Patrick Siarry. 2013. "A Survey on Optimization Metaheuristics." *Information Sciences* 237 (July): 82–117. https://doi.org/10.1016/j.ins.2013.02.041.

Burke, Edmund K, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. "Hyper-Heuristics: A Survey of the State of the Art." *Journal of the Operational Research Society* 64 (12): 1695–1724. https://doi.org/10.1057/jors.2013.71.

Burke, Edmund K., Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R. Woodward. 2009. "Exploring Hyper-Heuristic Methodologies with Genetic Programming." In *Computational Intelligence*, edited by Christine L. Mumford and Lakhmi C. Jain, 1:177–201. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01799-5_6.

Burke, Edmund K., Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. 2010. "A Classification of Hyper-Heuristic Approaches." In *Handbook of Metaheuristics*, edited by Michel Gendreau and Jean-Yves Potvin, 146:449–68. Boston, MA: Springer US. https://doi.org/10.1007/978-1-4419-1665-5_15.

Burke, Edmund K., Matthew R. Hyde, and Graham Kendall. 2012. "Grammatical Evolution of Local Search Heuristics." *IEEE Transactions on Evolutionary Computation* 16 (3): 406–17. https://doi.org/10.1109/TEVC.2011.2160401.

Chen, Stephen. 2012. "Particle Swarm Optimization with Pbest Crossover." In *2012 IEEE Congress on Evolutionary Computation*, 1–6. Brisbane, Australia: IEEE. https://doi.org/10.1109/CEC.2012.6256497.

Cowling, Peter, Graham Kendall, and Eric Soubeiga. 2001. "A Hyperheuristic Approach to Scheduling a Sales Summit." In *Practice and Theory of Automated Timetabling III*, edited by Edmund Burke and Wilhelm Erben, 2079:176–90. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-44629-X_11.

DaCosta, Luis, Alvaro Fialho, Marc Schoenauer, and Michèle Sebag. 2008. "Adaptive Operator Selection with Dynamic Multi-Armed Bandits." In , 913. ACM Press. https://doi.org/10.1145/1389095.1389272.

Drake, John H., Ender Özcan, and Edmund K. Burke. 2012. "An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search." In *Parallel Problem Solving from Nature - PPSN XII*, edited by Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and

Mario Pavone, 7492:307–16. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-32964-7_31.

Drake, John H., Ender Ozcan, and Edmund K. Burke. 2015. “A Modified Choice Function Hyper-Heuristic Controlling Unary and Binary Operators.” In , 3389–96. IEEE. https://doi.org/10.1109/CEC.2015.7257315.

Elsayed, Saber M., Ruhul A. Sarker, and Daryl L. Essam. 2011. “GA with a New Multi-Parent Crossover for Solving IEEE-CEC2011 Competition Problems.” In , 1034–40. IEEE. https://doi.org/10.1109/CEC.2011.5949731.

Fortin, Felix-Antoine. 2012. “DEAP: Evolutionary Algorithms Made Easy” 13 (July): 2171–75.

Gang, Ma, Zhou Wei, and Chang Xiaolin. 2012. “A Novel Particle Swarm Optimization Algorithm Based on Particle Migration.” *Applied Mathematics and Computation* 218 (11): 6620–26. https://doi.org/10.1016/j.amc.2011.12.032.

Gutierrez-Rodriguez, Andres E., Jose C. Ortiz-Bayliss, Alejandro Rosales-Perez, Ivan M. Amaya-Contreras, Santiago E. Conant-Pablos, Hugo Terashima-Marin, and Carlos A. Coello Coello. 2017. “Applying Automatic Heuristic-Filtering to Improve Hyper-Heuristic Performance.” In , 2638–44. IEEE. https://doi.org/10.1109/CEC.2017.7969626.

Hong, Libin, John H. Drake, John R. Woodward, and Ender Özcan. 2018. “A Hyper-Heuristic Approach to Automated Generation of Mutation Operators for Evolutionary Programming.” *Applied Soft Computing* 62 (January): 162–75. https://doi.org/10.1016/j.asoc.2017.10.002.

Izzo, Dario, Marek Ruciński, and Francesco Biscani. 2012. “The Generalized Island Model.” In *Parallel Architectures and Bioinspired Algorithms*, edited by Francisco Fernández de Vega, José Ignacio Hidalgo Pérez, and Juan Lanchares, 415:151–69. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28789-3_7.

Jong-Bae Park, Yun-Won Jeong, Joong-Rin Shin, and K.Y. Lee. 2010. “An Improved Particle Swarm Optimization for Nonconvex Economic Dispatch Problems.” *IEEE Transactions on Power Systems* 25 (1): 156–66. https://doi.org/10.1109/TPWRS.2009.2030293.

Kao, Yi-Tung, and Erwie Zahara. 2008. “A Hybrid Genetic Algorithm and Particle Swarm Optimization for Multimodal Functions.” *Applied Soft Computing* 8 (2): 849–57. https://doi.org/10.1016/j.asoc.2007.07.002.

Kennedy, J., and R. Mendes. 2002. “Population Structure and Particle Swarm Performance.” In , 2:1671–76. IEEE. https://doi.org/10.1109/CEC.2002.1004493.

Konstantinos Parsopoulos, and Michael N. Vrahatis. 2002. “Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method,” 6.

Liang, J.J., A.K. Qin, P.N. Suganthan, and S. Baskar. 2006. “Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions.” *IEEE Transactions on Evolutionary Computation* 10 (3): 281–95. https://doi.org/10.1109/TEVC.2005.857610.

Lozano, Manuel, Francisco Herrera, Natalio Krasnogor, and Daniel Molina. 2004. “Real-Coded Memetic Algorithms with Crossover Hill-Climbing.” *Evolutionary Computation* 12 (3): 273–302. https://doi.org/10.1162/1063656041774983.

Maashi, Mashael, Ender Özcan, and Graham Kendall. 2014. “A Multi-Objective Hyper-

Heuristic Based on Choice Function." *Expert Systems with Applications* 41 (9): 4475–93. https://doi.org/10.1016/j.eswa.2013.12.050.

McClymont, Kent, and Ed C Keedwell. 2011. "Markov Chain Hyper-Heuristic (MCHH): An Online Selective Hyper-Heuristic for Multi-Objective Continuous Problems," 8.

Mendes, R., J. Kennedy, and J. Neves. 2004. "The Fully Informed Particle Swarm: Simpler, Maybe Better." *IEEE Transactions on Evolutionary Computation* 8 (3): 204–10. https://doi.org/10.1109/TEVC.2004.826074.

Miranda, Péricles B.C., and Ricardo B.C. Prudêncio. 2017. "Generation of Particle Swarm Optimization Algorithms: An Experimental Study Using Grammar-Guided Genetic Programming." *Applied Soft Computing* 60 (November): 281–96. https://doi.org/10.1016/j.asoc.2017.06.040.

Miranda, Péricles B.C., Ricardo B.C. Prudêncio, and Gisele L. Pappa. 2017. "H3AD: A Hybrid Hyper-Heuristic for Algorithm Design." *Information Sciences* 414 (November): 340–54. https://doi.org/10.1016/j.ins.2017.05.029.

Nelder, J. A., and R. Mead. 1965. "A Simplex Method for Function Minimization." *The Computer Journal* 7 (4): 308–13. https://doi.org/10.1093/comjnl/7.4.308.

Nickabadi, Ahmad, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. 2011. "A Novel Particle Swarm Optimization Algorithm with Adaptive Inertia Weight." *Applied Soft Computing* 11 (4): 3658–70. https://doi.org/10.1016/j.asoc.2011.01.037.

Ochoa, Gabriela, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, et al. 2012. "HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search." In *Evolutionary Computation in Combinatorial Optimization*, edited by Jin-Kao Hao and Martin Middendorf, 7245:136–47. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29124-1_12.

Pant, Millie, Radha Thangaraj, and Ajith Abraham. 2007. "A New PSO Algorithm with Crossover Operator for Global Optimization Problems." In *Innovations in Hybrid Intelligent Systems*, edited by Emilio Corchado, Juan M. Corchado, and Ajith Abraham, 44:215–22. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-74972-1_29.

Rothlauf, Franz, and Marie Oetzel. 2006. "On the Locality of Grammatical Evolution." In *Genetic Programming*, edited by Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, 3905:320–30. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/11729976_29.

Ryan, Conor, Jj Collins, and Michael O Neill. 1998. "Grammatical Evolution: Evolving Programs for an Arbitrary Language." In *Genetic Programming*, edited by Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, 1391:83–96. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/BFb0055930.

Sabar, Nasser R, Masri Ayob, and Graham Kendall. 2013. "Grammatical Evolution Hyper-Heuristic for Combinatorial Optimization Problems" 17 (6): 23.

Sabar, Nasser R., Masri Ayob, and Graham Kendall. 2014. "The Automatic Design of Hyper-Heuristic Framework with Gene Expression Programming for Combinatorial Optimization Problems" 19 (3): 18.

Sabar, Nasser R., Masri Ayob, Graham Kendall, and Rong Qu. 2015. "A Dynamic Multiarmed Bandit-Gene Expression Programming Hyper-Heuristic for

Combinatorial Optimization Problems." *IEEE Transactions on Cybernetics* 45 (2): 217–28. https://doi.org/10.1109/TCYB.2014.2323936.

Shi, X.H., Y.C. Liang, H.P. Lee, C. Lu, and L.M. Wang. 2005. "An Improved GA and a Novel PSO-GA-Based Hybrid Algorithm." *Information Processing Letters* 93 (5): 255–61. https://doi.org/10.1016/j.ipl.2004.11.003.

Shi, Y., and R. Eberhart. 1998. "A Modified Particle Swarm Optimizer." In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 69–73. https://doi.org/10.1109/ICEC.1998.699146.

Soria-Alcaraz, Jorge A., Gabriela Ochoa, Marco A. Sotelo-Figeroa, and Edmund K. Burke. 2017. "A Methodology for Determining an Effective Subset of Heuristics in Selection Hyper-Heuristics." *European Journal of Operational Research* 260 (3): 972–83. https://doi.org/10.1016/j.ejor.2017.01.042.

Swan, Jerry, Ender Özcan, and Graham Kendall. 2011. "Hyperion – A Recursive Hyper-Heuristic Framework." In *Learning and Intelligent Optimization*, edited by Carlos A. Coello Coello, 6683:616–30. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-25566-3_48.

Swan, Jerry, John Woodward, Ender Özcan, Graham Kendall, and Edmund Burke. 2014. "Searching the Hyper-Heuristic Design Space." *Cognitive Computation* 6 (1): 66–73. https://doi.org/10.1007/s12559-013-9201-8.

Tan, Boxiong, Hui Ma, and Yi Mei. 2018. "A Genetic Programming Hyper-Heuristic Approach for Online Resource Allocation in Container-Based Clouds," 12.

Tang, Ke, Xiaodong Li, P N Suganthan, Zhenyu Yang, and Thomas Weise. 2009. "Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization," November, 24.

Thangaraj, Radha, Millie Pant, Ajith Abraham, and Pascal Bouvry. 2011. "Particle Swarm Optimization: Hybridization Perspectives and Experimental Illustrations." *Applied Mathematics and Computation* 217 (12): 5208–26. https://doi.org/10.1016/j.amc.2010.12.053.

Vandenbergh, F, and A Engelbrecht. 2006. "A Study of Particle Swarm Optimization Particle Trajectories." *Information Sciences* 176 (8): 937–71. https://doi.org/10.1016/j.ins.2005.02.003.

Wang, Yu, Bin Li, Thomas Weise, Jianyu Wang, Bo Yuan, and Qiongjie Tian. 2011. "Self-Adaptive Learning Based Particle Swarm Optimization." *Information Sciences* 181 (20): 4515–38. https://doi.org/10.1016/j.ins.2010.07.013.

Zhang, Weigang. 2007. "Multi-Objective Optimization for Crash Safety Design of Vehicles Using Stepwise Regression Model." *Chinese Journal of Mechanical Engineering* 43 (08): 142. https://doi.org/10.3901/JME.2007.08.142.

ملخص

ان كتابة حل خوارزمي لحل مشكلة ما يتطلب معرفة عميقة بأحد الموجهات التجريدية ويتطلب الكثير من الوقت. سنقوم في هذا البحث بدراسة منهجية بحث عالية المستوى المسمّاة بمنهجية الموجّه الفائق والتي تُطبَّق على خوارزمايات مجال بحث الموجّه والموجّه التجريدي. تهدف هذه الدراسة لإيجاد الحل الخوارزمي الأكثر ملائمة لفئة معينة من المشاكل. الموجّه الفائق يقسم إلى فئتين رئيستين: موجّه فائق انتقائي وموجّه فائق مولِّد. ستركز هذه الدراسة على الموجّه الفائق المولِّد، وبشكل خاص على الموجّه الفائق المولِّد الذي يُطبَّق على مكونات خوارزمية سرب الجزيئات و خوارزمية الجينات. تستخدم هذه الدراسة قواعد معدلة ممثلة كبنية الشجرة المسمّاة البرمجة الجينية الموجهة من أجل توليد حالّان خوارزمية سرب الجزيئات وخوارزمية الجينات هجينتين لمشاكل التحسين الشاملة المستمرة. في هذه الدراسة قمنا بمقارنة خوارزميتنا مع خوارمييتين رائدتين والنتائج أظهرت ان خوارميتنا تتمتع بكفاءة عالية.