Automated Design of Hyper-Heuristics Components to Solve the PSP Problem with HP Model

Vidal D. Fontoura
Federal University of Parana
(DInf-UFPR), Curitiba - PR, Brazil
Email: vdfontoura@inf.ufpr.br

Aurora T. R. Pozo
Federal University of Parana
(DInf-UFPR), Curitiba - PR, Brazil
Email: aurora@inf.ufpr.br

Roberto Santana
University of the Basque Country
San Sebastian, Spain
Email: roberto.santana@ehu.es

***Abstract*—The Protein Structure Prediction (PSP) problem is one of the modern most challenging problems from science. Simplified protein models are usually applied to simulate and study some characteristics of the protein folding process. Hence, many heuristic strategies have been applied in order to find simplified protein structures in which the protein configuration has the minimal energy. However, these strategies have difficulties in finding the optimal solutions to the longer sequences of amino-acids, due to the complexity of the problem and the huge amount of local optima. Hyper heuristics have proved to be useful in this type of context since they try to combine different heuristics strengths into a single framework. However, there is lack of work addressing the automated design of hyper-heuristics components. This paper proposes GEHyPSP, an approach which aims to achieve generation, through grammatical evolution, of selection mechanisms and acceptance criteria for a hyper-heuristic framework applied to PSP problem. We investigate the strengths and weaknesses of our approach on a benchmark of simplified protein models. GEHyPSP was able to reach the best known results for 7 instances from 11 that composed the benchmark set used to evaluate the approach.**

I. Introduction

Proteins execute an essential role in nature, they are responsible of many important functions for the living cells. Proteins can be seen as amino-acids structures that guarantee the correct operation of many process from the biological entities. These structures are product of the so-called protein folding process where an unfolded chain of amino-acids is transformed into its final/native structure.

Many authors [4], [13], [3] have used the Hydrophobic-Polar (HP) [7] simplified model to represent the protein structures. This model simplifies the amino-acids into just two types: hydrophobic (H) or polar (P). Two-dimensional (2D) and three-dimensional (3D) grids can be used to represent the conformations that a protein can adopt. Each conformation in the grid has an associated energy.

A wide range of heuristics strategies were proposed to search conformations of minimum energy in the HP model. Among the approaches that have been applied are genetic algorithms [3], ant colony optimization [14], estimation distribution algorithms [13], and others [9]. Even though there are different strategies already proposed for the PSP, all face difficulties due to the variability of the characteristics of the fitness landscape for different HP instances. Some heuristics may work well for some instances but fail to produce good results on instances with different characteristics.

This fact motivates the study here presented. It is in this type of context that hyper-heuristics are usually a good option. In general the hyper-heuristics frameworks propose a strategy to select, from a pool, the heuristic which is more appropriate for a given stage of the search. Also the hyper-heuristics frameworks define a move acceptance criterion, which is responsible to decide on whether to accept or reject solutions generated by the heuristics. These strategies are high level heuristics from a hyper-heuristic framework.

Usually the high level heuristics are human-designed approaches although two recent and novelty studies [11], [12] proposes the automated design of those components. The results obtained on both studies inspired the idea of automating the design of high level heuristics for a hyper-heuristic framework to solve the PSP problem. Since the PSP with the HP-2D model presents a very complex landscape, automating the design of hyper-heuristic framework could save time, money and human effort. Another motivation of the present study is that the hyper-heuristics frameworks have been applied to other domains and achieved good results. Some examples are the bin packing, personnel scheduling, flowshop, TSP, MAXSAT and CVRP problems [12].

We introduce GEHyPSP: an automated mechanism for the generation of high level heuristics for a hyper-heuristic framework to solve the PSP problem. The high level heuristic includes two main components: a selection mechanism and an acceptance criterion. Both components are generated using grammatical evolution (GE) [10], which is a type of genetic programming (GP) [5]. A set of experiments was designed to evaluate the approach using eleven PSP instances. The results are compared with a the best known results and other human-designed hyper-heuristic approach.

This paper is organized as follows: in the next section we briefly introduce the Protein Structure Prediction problem and a review of the related works is also presented. Section III presents a background of hyper-heuristics, GP and the GE algorithm. Thereafter, in Section IV, the GEHyPSP is introduced. In Section V, the experimental benchmark and numerical results of the conducted experiments are presented. Finally, in Section VI, the conclusions of the research are given, and further work is discussed.

978-1-5090-4601-0/17/$31.00

II. Protein Structure Prediction

Proteins are macromolecules composed by an alphabet of twenty different amino-acids, also referred to as residues. An amino-acid is formed by a peptide backbone and a distinctive side chain group. The peptide bond is defined by an amino group and a carboxyl group connected to an alpha carbon to which a hydrogen and side chain group are attached.

The protein sequence folds, under particular conditions, into a unique native 3-D structure. Each possible protein conformation/structure has an associated energy value. The thermodynamic hypothesis states that the native structure of a protein is the one for which the free energy achieves the global minimum. Based on this hypothesis, many methods [3], [4], [6] that search for the protein native structure define an approximation of the protein energy and use optimization algorithms that look for the protein fold that minimizes the energy. These approaches mainly differ in the type of energy approximation employed and in the characteristics of the protein modeling.

A. The HP Model

Lau and Dill [7] created a model called *Hydrophobic-Hydrophilic* Model (HP Model), to represent the proteins using simplifications. The model can be used either to represent proteins in a 2D space or 3D space.

The HP model considers only two types of residues: hydrophobic (H) and hydrophilic or polar (P). A protein is considered a sequence of these two types of residues, which are located in regular lattice models and must form a self-avoided walk (SAW). If a structure is not a SAW it means that it contains collisions between the amino-acids and this structure is considered to be invalid. Given a pair of residues, they are considered neighbors if they are adjacent either in the chain (connected neighbors) or in the lattice but not connected in the chain (topological neighbors). In the HP context the amino-acid sequences are used as instances for the problem.

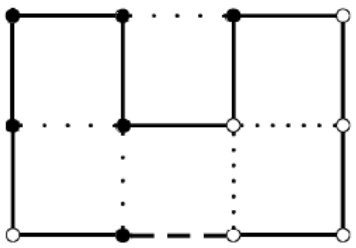

Fig. 1. One possible configuration of sequence $PPPPHPHHHHPH$ in the HP model. The white circles represents the P amino-acids and the black circles describes the H amino-acids. The sequence starts at the third white circle on the bottom of the figure. The next amino acids are positioned counter-clockwise. There are two $H-H$ (represented by a dotted line with wide spaces), one $H-P$ (represented by a dashed line) and two $P-P$ (represented by dotted lines) contacts.

For the HP model, an energy function that measures the amount of topological neighbor residues is defined as $\epsilon_{HH} = -1$ and $\epsilon_{HP} = \epsilon_{PP} = 0$. The HP problem consists of finding the protein configuration that minimizes the total energy. In the linear representation of the sequence, hydrophobic residues are represented with the letter H and polar ones, with P. In the graphical representation, hydrophobic proteins are represented by black beads and polar proteins, by white beads. Fig 1 presents a possible graphical conformation of a sequence in the 2D lattice. The energy associated with this conformation is -3.

We briefly review some of the previous studies on the optimization of the simplified PSP, relevant for our research.

A study presented in [6] introduces the MMA (Multimeme algorithm) which consisted on an EA combined with a group of local search methods. For each individual in the population, the MMA selects the local search method that is more suitable with the individual. Used first to find solutions for the functional model protein, the strategy was later improved with fuzzy-logic-based local searches, leading the algorithm to achieve improved results in the PSP problem.

In [4], Hsu et al. present the pruned-enriched Rosenbluth method (PERM), also known as chain growth algorithm, that is based on growing the sequence conformation by adding particle by particle, aiming to increase good configurations and eliminating bad ones.

The ant colony optimization (ACO), in [14], was applied to the PSP problem using the HP-2D model. This strategy, uses artificial ants to build conformations for a given HP instance (sequence of amino-acids). A local search method is then applied to further improve the solutions and also maintain the quality of the solutions.

The study of Santana et al. [13] applies EDAs as an efficient evolutionary algorithm that learns and exploits the regularities of the search space in the form of probabilistic dependencies. They introduced new probabilistic models tailored for the application of EDAs to 2D and 3D simplified protein folding problem. The obtained results showed that EDAs can achieve superior solutions compared with other well-known population based optimization algorithms.

III. Background

This section will present the background context in order to provide the readers with the necessary information, for a good comprehension of the concepts and techniques used in this paper.

A. Hyper-Heuristics

Mainly, a generic hyper-heuristic framework is composed of two components known as high-level and low-level heuristics. Fig 2 shows a general scheme of hyper-heuristic frameworks. The high and low levels are separated by a domain barrier which means that the low-level component is problem dependent while the high-level component does not require any knowledge of the problem domain. Ideally it is possible to apply a hyper-heuristic framework to a different problem domain by only replacing the low level component. The responsibility of a high-level component is manage the selection or generation of which heuristic should be applied at each

decision point. The low-level component corresponds to a pool of heuristics or heuristic components [11].

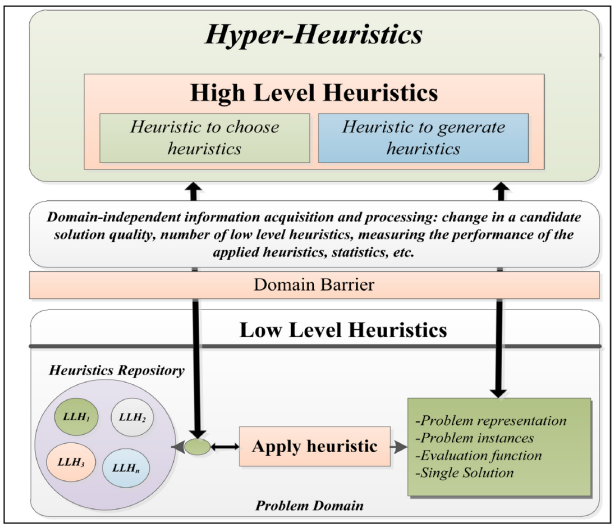

Fig. 2. General scheme of a hyper-heuristic framework [11].

Based on the nature of the search space, hyper-heuristics can be classified as either selecting or generating heuristics for the underlying problem [2]. Next we analyse both types of high-level heuristics.

- Selecting Heuristics: The majority of hyper-heuristic frameworks define high level heuristics to select low level heuristics [2]. In general, these frameworks define a selection mechanism and also an acceptance criterion. These strategies use information from the statistical history of low level heuristics applications. For instance, a greedy selection considers only the number of improvements that a low level heuristic obtained. In contrast, the choice function [2] is a reward-based strategy, which considers both the number of improvements and the elapsed time since the last application.
- Generating Heuristics: In this case, the hyper-heuristic framework starts with a set of subcomponents from the low level heuristics and has the goal of building new low level heuristics with it. GP is reported as a good strategy to combine and generate new heuristics for the SAT, scheduling, and bin-packing problems [2].

Two studies presented by Sabar et al. [11], [12], proposes an on-line approach to generate selection mechanisms and acceptance criteria (high level heuristics) to a hyper-heuristic framework. The results in both studies outperforms some bespoke strategies, which have reported best known results in some instances of the domain problems explored. This encouraging results inspired the present study which has the objective of generating selection mechanisms and acceptance criteria, through a grammatical evolution process in a off-line manner, for a hyper-heuristic framework to solve the PSP problem using the simplified HP-2D model.

B. Genetic Programming (GP)

It is a sub-field from program synthesis which uses ideas from evolution theory to produce programs [5].

Grammatical Evolution (GE) is a relatively new technique from the evolutionary computing, introduced by Ryan el al. [10] and it is a type of GP where programs are evolved using a genetic algorithm. The chromosome encodes production rules of the grammar. GE uses a mapping mechanism between the genotype (coded individuals by integer vectors) and phenotype (generated programs to solve a problem). The BNF notation is used to represent a grammar from a language in the form of production rules. A BNF grammar consists in a set of terminals, which are items that are allowed in the language, for instance: +, -, *, /, etc and non-terminals, which can be expanded into one or more terminals. The Grammar can be expressed as a tuple N, T, P, where N is a set of non-terminals presented in the Grammar 1 as $\langle expr \rangle$, $\langle op \rangle$, $\langle pre-op \rangle$ and $\langle var \rangle$, T a set of terminals and is presented in the Grammar 1 as $+$, $-$, $/$, $*$, Sin, Cos and Tan. Finally, P is the set of production rules that maps the elements N to T and is presented in Table I.

$\langle expr \rangle$:= $\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$ (0)
| ($\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$) (1)
| $\langle pre\text{-}op \rangle$ ($\langle expr \rangle$) (2)
| $\langle var \rangle$ (3)

$\langle op \rangle$:= + (0)
| - (1)
| / (2)
| * (3)

$\langle pre\text{-}op \rangle$:= Sin (0)
| Cos (1)
| Tan (2)

$\langle var \rangle$:= X (0)

Grammar 1. Sample grammar to demonstrate how to decode integer vectors in computer programs

TABLE I
EXAMPLE OF POSSIBLE PRODUCTION RULES AND THE NUMBER OF CHOICES ALLOWED

Production Rules	Number of choices
$\langle expr \rangle$	4
$\langle op \rangle$	4
$\langle pre - op \rangle$	3
$\langle var \rangle$	1

Ryan et al. [10] proposed the use of a GA to control which choices should be made, in this sense allowing the GA to select which production rules should be utilized. Algorithm 1 presents the pseudocode of a grammatical evolution program. Lines 3 to 6 represent the initialization of the algorithm: a

randomly generated population is created and then mapped to programs using the GF (Grammar File). Those programs are evaluated and a fitness value is assigned for each individual in the population. Line 7 indicates the start of the main loop of the algorithm and the evolution process occurs within this loop. Line 8 invokes the parent selection that will be used in the crossover in Line 9 which will generate an offspring (integer vectors). The offspring is then used as input to prune and duplication operators in Lines 10 and 11. The mutation operator is applied to the offspring in Line 12. The mapping between the integer vectors to programs is accomplished in Line 13. Next the execution and fitness assignment to the individuals from the population are implemented in Line 14 and 15. The population replacement with the fittest individuals is done in Line 16. Finally, Line 18 returns the program with the best fitness found.

Algorithm 1: Pseudo code from the Grammatical Evolution

```
1  Input: GF – Grammar File
2  begin
3      population ← Create population;
4      programs ← Maps the population to programs using GF;
5      Execute the program;
6      Assign fitness value with the solutions of population according with the output obtained
         by the respective decoded program;
7      while Stop condition not reached do
8          parents ← Select individuals for crossover;
9          offspring ← Crossover(parents);
10         Apply the Prune operator to the offspring;
11         Apply the Duplicate operator to the offspring;
12         Apply the Mutation operator to the offspring;
13         programs ← Maps offspring to programs using GF;
14         Execute programs;
15         Assign fitness value to offspring according with the output obtained by the
             respective decoded program;
16         population ← Replacement;
17     end
18     return Best program from the population;
19 end
```

Algorithm 2 presents the general template of the generated programs. In Line 3 the variable f receives a function returned by the genotype/phenotype mapping. The function is executed and the resulting value is returned in Lines 4 and 5.

Algorithm 2: General template for the generated algorithms

```
1  Input: int[] chromossome;
2  begin
3      Function f = mapGenotypeToPhenotype(chromossome);
4      double value = f.execute();
5      return value;
6  end
```

The mapping process to decode a genotype on its phenotype will be demonstrated using the following integer vector:

$$[220, 203, 17, 6, 108, 215, 104, 30]$$

This vector will be used to decode the chromosome (genotype) into a piece of code (phenotype) using the Grammar 1. Table I summarizes the number of choices associated within each production rule from Grammar 1.

There are 4 options of production rules that can be selected for the expression $\langle expr \rangle$. In order to determine which is the option to select, the first value from the vector should be used. The value is 220 and its remainder divided by 4 (four options that can be selected for the expression $\langle expr \rangle$) results in 0, which means that the first option should be selected $\langle expr \rangle \langle op \rangle \langle expr \rangle$. Note that the first expression is also $\langle expr \rangle$ and following the same logic we should expand using the next value from the integer vector and apply its remainder by the division of the number of options. The remainder of 203 $\%$ 4 = 3, which indicates that we should select the fourth option: $\langle var \rangle$ which is a terminal. The $\langle var \rangle$ has just one option associated with it and is the value X. Placing this selection in the original expression we have $X \langle op \rangle \langle expr \rangle$.

Next, it is necessary to decode the non-terminal expression $\langle op \rangle$. The next value from the integer vector is 17 and again we have 4 options $(+|-|/|*)$. The result is equal to 1, and indicates to select the: $-$. Re-writing the expression we got: $X - \langle expr \rangle$. This process should continue until all the non-terminals have been expanded to terminals. In this example the resulting expression (phenotype) is: $X - sin(X)$. Note that not all of the genes were necessary to obtain the phenotype. In these cases, the genes that were not used are discarded. Moreover, the opposite case can occur: if a chromosome does not contain the necessary number of genes to map to a program. In this scenario the strategy is to re-use the genes starting from the first one.

IV. GEHyPSP

This section presents GEHyPSP: an off-line grammatical evolution application for generating the high level heuristics of a hyper-heuristics framework for the Protein Structure Prediction problem. Our approach is based on the study by Sabar et al. [11]

The high level heuristics are composed of a selection mechanism and an acceptance criterion. These high level heuristics use information related of the history of low level heuristics applications. Data about the improvements obtained by the low level heuristics, number of times since the last application of a low level heuristic and the fitness difference between the current solution and the generated one are examples of information used by selection mechanisms and acceptance criteria. Furthermore, two terminal sets (one for the selection mechanisms and another for the acceptance criteria) were defined according to the information that can be extracted during the search progress. The selection terminals are:

- RC (*Reward Credit*): The reward that a given heuristic should receive based on its performance. The improvement is calculated, for the i_{th} heuristic, using $M(i) = (|f1 - f2|/f1) * 100$ if $f2 < f1$, where $f1$ is the current fitness and $f2$ is the fitness of the solution generated by the i_{th} heuristic.
- C_{best}: Number of times that the i_{th} heuristic updated the best known solution.
- $C_{current}$: Number of times that the i_{th} heuristic updated the current solution.

- C_{accept}: Number of times that the solution generated by the i_{th} heuristic was accepted by the acceptance criterion.
- C_{ava}: The average of previous improvements made by the i_{th} heuristic during the search progress.
- C_r: Number of times that the i_{th} heuristic was selected to be applied.

A specific terminal set for generating acceptance criteria was also defined.

- Delta: The difference between the quality of the current solution and the generated solution.
- PF: The quality of the previous solution.
- CF: The quality of the current solution.
- CI: Current iteration of the search progress.
- TI: Total number of iterations of the search progress.

Using these statistics as terminals and a function set containing the following arithmetic operations: addition, subtraction, multiplication and division, a grammar was designed. The Grammar 2 presents an BNF notation to represent the grammatical rules. Within these rules it is possible to generate arithmetic functions. These functions can be applied to sort the low level heuristics set. When a low level heuristic is ranked first, this heuristic is selected to be applied generating the next solution. In the same way, the acceptance criterion will accept solutions only if the outcome obtained by the acceptance arithmetical function is higher than 0.5.

For the initialization of the terminal set data: all heuristics were executed once and the data for each terminal was computed. Every consecutive iteration will update the terminal set data and this information will be used during the search.

⟨*hh-selection*⟩ ::= ⟨*selection-mechanism*⟩ ⟨*acceptance-criterion*⟩

⟨*selection-mechanism*⟩ :== ⟨*selection-terminal*⟩
| ⟨*selection-mechanism*⟩ ⟨*math-function*⟩ ⟨*selection-mechanism*⟩
| (⟨*selection-mechanism*⟩ ⟨*math-function*⟩ ⟨*selection-mechanism*⟩)

⟨*selection-terminal*⟩ :== RC | Cbest | Ccurrent | Caccept | Cava | Cr

⟨*math-function*⟩ :== + | - | * | %

⟨*acceptance-criterion*⟩ ::== ⟨*acceptance-terminal*⟩
| ⟨*acceptance-criterion*⟩ ⟨*math-function*⟩ ⟨*acceptance-criterion*⟩
| (⟨*acceptance-criterion*⟩ ⟨*math-function*⟩ ⟨*acceptance-criterion*⟩)

⟨*acceptance-terminal*⟩ :== PF | CF | CI | TI

Grammar 2. Designed grammar to generate high level heuristics

A. Fitness Function to evaluate the High Level Heuristics

In order to evaluate the individuals generated during the search, a fitness function was designed. The fitness function consisted of running the generated high level heuristics, within a hyper-heuristic framework, on three random selected instances from a set of eleven instances. The motivation behind executing the high level heuristic (individual) against three HP instances is that executing with only one might not be sufficient to train a high level heuristic to obtain good results for various instances of the HP model.

Each run will be executed for 30 minutes and will return the best HP solution found. The fitness value associated with the returned solution is then normalized between 0 and 1. The fitness of an individual, of the GE, is the sum of the three outcomes from each execution of the three randomly selected instances. Hence, the best possible fitness value is 3 and the worst is 0.

B. Stopping Criterion

To stop the GE process, a maximum number of evaluations was setup to 60000. This value was defined based on previous work [10] where the grammatical evolution general process was introduced for the first time.

C. Low Level Heuristics

1) Representation of the problem: There are many ways of representing a protein conformation within the HP-2D model. According to Krasnogor et al. [6] the relative representation has a better potential to achieve superior results. The relative representation defines that each gene of the chromosome represents a direction in the grid. Each gene will encode an conformation that a protein could adopt. The directions in the grid for each amino-acid are represented always based on the previous one. There are 3 possible directions within the HP-2D model: forward (F), left (L) and right (R). Hence, the following integer codification was used $F \rightarrow 0$, $L \rightarrow 1$ and $R \rightarrow 2$. Thereby, the allowed alphabet can be represented as $\{0, 1, 2\}$.

2) Low Level Heuristics Set: The low level heuristics set was selected from previous studies [1], [3] that explore the PSP. The implemented low level heuristic set have 6 heuristics with different characteristics. The majority of the heuristics are applied to a single solution. However, there are cases that the heuristics require two individuals. In this cases the current solution and a second solution, randomly selected from the memory mechanism (See subsection IV-C4), are used.

The low level heuristic set consisted of the following:

- *Two Points Exchange* (2X): This heuristic selects, randomly, two exchange points from two individuals. The genes between the selected positions are exchanged between the input solutions in order to generate a new one [1].
- *Multiple Points Exchange* (MPX): Similar to 2X although with c points of exchange. We use $c = int(n{*}0.1)$, where n is the instance length. The MPX is useful to promote a wide structural diversity [12].
- *Segment Change* (SG): Given an individual it changes multiple consecutive genes, between 5 to 7, to distinct values. This heuristic introduces large changes in the protein conformation, and it has a great probability of

creating collisions (cases in which the protein conformation are not self avoiding).

- *Exhaustive Search Change* (ESC): This heuristic selects a random gene of an individual and modifies it to other possible value. All possible values are tried and the one that achieved the higher improvement will be kept.
- *Local Move Operator* (LM): This heuristic exchanges directions between two consecutive random selected genes of an individual. This heuristic introduces small local changes that can reduce the distance between pairs of H-H.
- *Loop Move Operator* (LPM): Similar with LM, this heuristic exchanges directions between two genes that are five genes of distance between each other.
- *Opposite Change* (OC): This heuristic exchanges multiple genes between two points (i, j), of an individual, to the respective opposite. In the HP-2D this means that genes that code L are changed to R or vice and versa. The F has no opposite thus, the operator is not applied.

This low level heuristic set will be used by the generated high level heuristics. In other words the selection mechanisms have the responsibility of selecting the most suitable heuristic for each iteration.

3) Backtrack Repair: The low level heuristics have a great potential of creating solutions with collisions [1]. Those solutions are submitted to a repair procedure that uses a backtrack strategy. The solutions that can be repaired are kept and the others are penalized.

4) Memory Mechanism: Sabar et al. [11] suggested that the use of a memory of solutions to the PSP would be more effective. Relying with a single solution. may restrict the ability of dealing with large and heavily constrained search spaces.

V. Experiments

In this section, we will present and discuss the conducted experiments in order to evaluate GEHyPSP performance. Since GEHyPSP is an offline strategy the experiments are divided in two phases: training and validation. The training phase consisted o executing the GEHyPSP to search for high level heuristics that can achieve good results for the fitness function described in IV-A. The validation phase consisted on executing the best individuals found in the training phase and execute again but now against all instances. Unfortunately, the results of the training phase were suppressed from the paper because of space limitations. Only the results of the best individuals found are presented.

Three groups of experiments were designed/executed. The first group was only concerned on generating selection mechanisms. The second group was designed only to generate acceptance criteria. The third group was developed to generate both selection and acceptance mechanisms. All experiments were executed 30 times because of the stochastic behavior of the GEHyPSP.

The results obtained by the groups were first compared with each other. Next, the results were compared with the best known results. Also a final comparison was made with a state-of-art hyper-heuristic. The proposed strategy was compared with the Generic Intelligent Hyper-heuristic (GIHH) presented by Misir el al [8].

TABLE II
Instances and the respective size of each one

Inst	1	2	3	4	5	6	7	8	9	10	11
Size	20	24	25	36	48	50	60	64	85	100	100

In the first group of experiments (GEHyPSP-1), only selection mechanisms are generated. The acceptance criterion was fixed with a "better or equal" acceptance [2]. This strategy only accepts (replace the current solution) the generated solution if its fitness is better or equal than the current solution.

The goal of this group is to evaluate the ability of our approach to generate selection mechanisms using a fixed and human-designed acceptance criterion.

The second group of experiments, GEHyPSP-2, consisted on generating only acceptance criteria. The selection mechanism was fixed using the best selection mechanism found in the first group. Consequently, this group of experiments depends on the output from the first group. The goal of this experiment was to evaluate the generation of acceptance criteria separately from the selection mechanism using a fixed one.

The third group of experiments GEHyPSP-3 was designed to generate both selection mechanisms and acceptance criteria. The goal of this group was to evaluate the ability of generating selection mechanisms along with acceptance criteria. Differently from GEHyPSP-2, this group of experiments does not depend on any output from previous experiments since it generates both mechanisms without fixing any component.

For each group of experiments, eleven instances, selected from the literature, were used. The sizes (number of residues) of the HP instances used in the experiments are presented in Table II. For the sake of space, the sequences are not reproduced here. They can be obtained in [13].

A. Results from GEHyPSP-1

The best individual found in the GEHyPSP-1 was the following selection mechanism: $RC * Ccurrent * Cava - Cr$ and it was executed 30 times with a time limit of 30 minutes against the eleven instances. Table III presents the fitness average, standard deviation, minimum and maximum of 30 executions of the best individual found in the GEHyPSP-1 experiment. The last row denoted with O(x*) is the best known value for each instance. Analysing the last two rows it is possible to notice that only in 7 instances the selection mechanism generated achieved the best known results. However in case of instances 7, 9, 10 and 11 the results obtained are very far from the best known results.

B. Results from GEHyPSP-2

The best individual found in the GEHyPSP-2 was the following acceptance criterion: $((TI/Delta)/((Delta *$

TABLE III
RESULTS FROM THE BEST INDIVIDUAL FOUND IN GEHYPSP-1

Inst	1	2	3	4	5	6	7	8	9	10	11
Avg	8.1	7.6	6.7	11.9	17.4	16	30	28.3	40.1	35.6	35.9
St Dv	0.3	0.5	0.5	0.7	1	1.4	1.7	2	2.7	2.1	2.9
Min	8	7	5	11	15	13	25	23	34	32	27
Max	**9**	**9**	**8**	**14**	**23**	**21**	33	**42**	46	40	41
O(x*)	**9**	**9**	**8**	**14**	**23**	**21**	**36**	**42**	**53**	**48**	**50**

TABLE IV
RESULTS FROM THE BEST INDIVIDUAL FOUND IN GEHYPSP-2

Inst	1	2	3	4	5	6	7	8	9	10	11
Avg	8	7.6	6.7	10.1	14.8	14.7	27	25.7	38.3	32.8	30.
St Dv	0.6	0.4	0.6	0.7	1.5	1.4	2.0	2.5	3.3	3.7	3.4
Min	7	7	5	8	12	12	23	22	31	26	24
Max	**9**	8	**8**	11	17	18	31	31	44	40	37
O(x*)	9	9	8	14	23	21	36	42	53	48	50

TABLE V
RESULTS FROM THE BEST INDIVIDUAL FOUND IN GEHYPSP-3

Inst	1	2	3	4	5	6	7	8	9	10	11
Avg	7.6	7.0	5.7	9.7	13.8	12.7	24	24.2	31.6	27	26.4
St Dv	0.6	0.7	0.8	0.9	1.3	0.9	1.1	1.6	1.7	1.7	2
Min	7	6	4	7	12	11	21	21	29	24	24
Max	**9**	8	**8**	11	18	15	26	28	37	31	31
O(x*)	9	9	8	14	23	21	36	42	53	48	50

TABLE VI
BEST RESULTS FOUND BY GEHYPSP, BEST RESULTS FOUND BY GIHH AND THE BEST KNOW RESULTS O(X*)

Inst	1	2	3	4	5	6	7	8	9	10	11
GEHyPSP	**9**	**9**	**8**	**14**	**23**	**21**	33	**42**	46	40	41
GIHH (max)	**9**	**9**	**8**	**14**	**23**	**21**	35	40	49	43	45
O(x*)	9	9	8	14	23	21	36	42	53	48	50

$((TI/Delta)/CI) * Delta/Delta * TI) - CI))$ and combined with the best individual from the GEHyPSP-1 the hyper-heuristic framework was executed 30 times for each one of the eleven instances with a time limit of 30 minutes. Table IV presents the results of the best generated acceptance criterion by GEHyPSP-2 with the best selection mechanism generated in GEHyPSP-1. Again, looking at the two last rows it is possible to notice that the hyper-heuristic framework, using the best generated selection mechanisms and acceptance criterion, was not able to reach the best known results for many cases. And comparing Tables III and IV it is possible to visualize that the results obtained by the GEHyPSP-2 are very far from the ones obtained by the first GEHyPSP-1. This means that the acceptance criterion was not able to achieve superior results than the acceptance criterion used in the GEHyPSP-1. The results achieved in the first experiment are very superior.

C. Results from GEHyPSP-3

The best individual generated by the experiment GEHyPSP-3 was both a selection mechanism and an acceptance criterion. And they are presented below:
Selection Mechanism: $(((((Caccept/RC) * Cr/Caccept)/RC * Cr)/Caccept/RC) * Cr)/Caccept$
Acceptance Criterion: $(((((CI/PF) * Delta/CI)/PF * Delta)/CI/PF) * Delta)/CI$

Table V presents the results of the best individual generated by the experiment GEHyPSP-3, of 30 executions with a time limit of 30 minutes. Once again the values presented did not achieve good results for most instances. Only for 2 very simple instances the best known results were achieved. Consequently, when comparing Table V with Tables III and IV it is possible to see that generating both selection mechanisms and acceptance criteria produces worse results than generating them separately.

D. Comparison with a state-of-art hyper-heuristic

In order to compare the generated high level heuristics with an already proposed hyper-heuristic strategy a state-of-art human-designed hyper-heuristic framework [8] was selected. This framework produced the best results applied at the 6 domain problems from HyFlex framework [9]. Misir et al. [8] provided us the source code from the GIHH and then it was executed against the PSP problem.

Table VI presents the best results, for each instance, found by the best generated high level heuristic with GEHyPSP, the best results found by the GIHH [8] and finally the best know results (O_x(*)). The GEHyPSP and the GIHH were executed in 30 minutes. It is possible to note that the GEHyPSP achieved similar results compared with GIHH. In 6 cases GIHH found the best known results and on the other hand GEHyPSP was able to achieve the best known results in 7 cases. Although in the case of instance 7 both approaches did not achieved the best known. However, it is possible to see that the GIHH obtained a higher value. The same is true for instances 9,10 and 11 which are the more complex from the benchmark set. Thus, the GIHH obtained better results than the GEHyPSP in the cases that the instances were more complex. However, the GIHH is a human-designed hyper-heuristic which demanded several years of research in order to be completed and achieved good results in the six domains problems provided by HyFlex.

E. Discussion

In order to better investigate the relationship between the generated selection mechanisms and acceptance criteria another experiment was designed. From the 30 executions of the GEHyPSP-2 ten random individuals were selected to be further analysed. These high level heuristics were re-executed on debugging mode in order to check its behavior. Note that only the acceptance criterion was different between them because the selection mechanism was fixed, using the best one generated in the previous experiment GEHyPSP-1. From 10 individuals, 7 were accepting only better or equal solutions just like the fixed acceptance criterion that was used in the GEHyPSP-1. The difference between the individuals and a fixed acceptance criterion was that individuals were slower than the fixed, because on one hand, it is required to execute arithmetical functions and on the other hand only a simple if was evaluated. But with respect to their behavior they were exactly the same. It was also noticed that 2 individuals, the worst of the group, were always accepting worst solutions just

like the "all moves" described by Burke et al. [2]. Finally, one individual was never accepting any solution.

These experiments showed that the GEHyPSP managed, several times, to find different acceptance criteria with the same behavior of human-designed strategies.

VI. Conclusion

In this work GEHyPSP, an automatic way of generating high level heuristics for a hyper-heuristic framework for the PSP problem, was presented and evaluated. The PSP is a very challenging problem with a high number of local optima and a very complex landscape. Many authors explored the PSP problem with heuristic methods. However, very often the proposed heuristic approaches are unable to find the best known results because of the wide variability of the characteristics between the fitness landscapes from different HP instances. Usually, the hyper-heuristic frameworks fits well in this type of complex scenario. Hence, the goal of this paper was to generate, using a grammatical evolution strategy, selection mechanisms and acceptance criteria to a hyper-heuristic framework and evaluate its performance and behavior with a benchmark set containing eleven HP instances. Three groups of experiments were executed, using three randomly selected HP instances, in order to generate the high level heuristics and later the best individuals found in the experiments were executed using all the HP instances from the benchmark set.

Three groups of experiments were executed: first generating only selection mechanisms within a fixed acceptance criterion; second generating only acceptance criteria using the best selection mechanism found in the first; finally both high level heuristics were generated in parallel. The results showed that better high level heuristics were found when generating them separately.

Combined with the "better or equal" human-designed acceptance criterion the best generated selection mechanism is able to achieve the best known results for 7 instances from a total of 11. However, in the remaining instances which are the most challenging and complex, only PERM [4], from the previous studies considered by this paper, obtained the best known results. Also a comparison with a good state-of-art human-designed hyper-heuristic framework (GIHH) [8] was done. The results from both approaches are slightly close. This fact suggested that it is possible to automate the creation of high level heuristics and obtain results close to the state-of-art hyper-heuristics frameworks. The investigation of this paper, where a new problem never explored by hyper-heuristics approaches, empathizes the ability of hyper-heuristics to solve a set of problems without a huge amount of specific knowledge.

Another finding of this work was the behavior of the best generated acceptance criteria. It was noticed that it behaves just like "a better or equal" human-designed acceptance criterion. Also some of the generated acceptance criteria were always accepting worst solution and this fact impacted in the individual fitness. This fact demonstrates that GEHyPSP was able to generate acceptance criteria with the same behavior of simple human-designed move acceptance strategies. However, in order to obtain better results it might be necessary to improve GEHyPSP to generate more complex selection mechanisms and acceptance criteria to couple with the landscape complexity of the PSP problem.

Acknowledgements

This work ask you received support from CNPq (Productivity Grant Nos.: 306103/2015-0 and Program Science Without Border Nos.: 400125/2014-5), from CAPES (Brazil Government), from IT609-13 program (Basque Government) and TIN2016-78365-R (Spanish Ministry of Economy, Industry and Competitiveness).

References

[1] César Manuel Vargas Benítez and Heitor Silvério Lopes. Protein structure prediction with the 3D-HP side-chain model using a master–slave parallel genetic algorithm. *Journal of the Brazilian Computer Society*, 16(1):69–78, 2010.

[2] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[3] Fábio Lima Custódio, Helio JC Barbosa, and Laurent Emmanuel Dardenne. A multiple minima genetic algorithm for protein structure prediction. *Applied Soft Computing*, 15:88–99, 2014.

[4] Hsiao-Ping Hsu, Vishal Mehra, Walter Nadler, and Peter Grassberger. Growth algorithms for lattice heteropolymers at low temperatures. *The Journal of chemical physics*, 118(1):444–451, 2003.

[5] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[6] Natalio Krasnogor, BP Blackburne, Edmund K Burke, and Jonathan D Hirst. Multimeme algorithms for protein structure prediction. In *Parallel Problem Solving from NaturePPSN VII*, pages 769–778. Springer, 2002.

[7] Kit Fun Lau and Ken A Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22(10):3986–3997, 1989.

[8] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. Intelligent hyper-heuristics framework for chesc 2011. pages 461–466. Springer, 2012.

[9] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al. Hyflex: A benchmark framework for cross-domain heuristic search. In *Evolutionary computation in combinatorial optimization*, pages 136–147. Springer, 2012.

[10] Conor Ryan, JJ Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, pages 83–96. Springer, 1998.

[11] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *Trans. Evol. Comp*, 17.

[12] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, 2015.

[13] Roberto Santana, Pedro Larrañaga, and Jose A Lozano. Component weighting functions for adaptive search with EDAs. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 4066–4073. IEEE, 2008.

[14] Alena Shmygelska and Holger H Hoos. An improved ant colony optimisation algorithm for the 2D HP protein folding problem. In *Advances in Artificial Intelligence*, pages 400–417. Springer, 2003.