Automated Design of Hyper-Heuristics Components Applied to the PSP Problem

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 Hermida
University of the Basque Country
Barrio Sarriena s/n 48940 Leioa Bizkaia
Email: roberto.santana@ehu.es

***Abstract*—Proteins are structures, composed by amino-acids, and have an important role in nature. These structures are formed by a process called protein folding. This process is not completely understood and it is considered one of the most challenging modern problems. This problem is usually called as Protein Structure Prediction (PSP) and can be considered as a minimization problem. Hence, many heuristics strategies have been applied in order to find protein structures that minimizes its free energy. However, these strategies have difficulties on finding the optimal solutions to the longer sequences of amino-acids, due, the complexity of the problem and the huge amount of local optima. The hyper-heuristics framework are usually useful in this kind of context since they try to combine different heuristics strengths into a single framework. Moreover, there is lack of works whose aim the automated design of hyper-heuristics components. Sabar et al. [20] proposed the automated design of high level heuristics to a hyper-heuristic framework and evaluated its performance by applying in the 6 domain problems from HyFlex [16] framework. The work of Sabar et al. [20] influenced this paper to chase the generation of high level components to a hyper-heuristic framework to the PSP problem.**

I. Introduction

Proteins plays a fundamental role in nature, they are involved in many important functions for the living cells. Proteins are structures, composed by amino-acids that guarantees the correct operation of wide range of biological entities. These structures are result from a process so-called protein folding, in which initially an unfolded chain of amino-acids will be transformed into its final/native structure.

The protein structure prediction (PSP) has a wide range of biotechnology and medical applications. For instance: synthesis of new proteins and folds [26], [18], structure based synthesis of new drugs [7], refinement of theoretical models obtained by comparative modeling [17], [12], and obtaining experimental structures from incomplete nuclear magnetic resonance data [22].

The determination of the native structure of a protein is a hard task even for the modern super computers. The difficulty is due to the huge search space to test all possible configurations that a given protein can adopt. Many models to represent the proteins structures exists and can be utilized to simulated the folding process. Although there exists extremely detailed models, these representations are computational very expensive. Hence, many authors [5], [10], [14], [25], [21], [6], [9] used simplified models to represent the protein structures. A useful model for this purpose is the Hydrophobic-Polar (HP) model presented by Lau and Dill [13]. This model generalizes the amino-acids that composes the proteins into just two types H or P. And, a grid (2D) or cube (3D) can be used to represent the conformations that a protein can adopt. Each conformation has an energy associated [25].

Many heuristics strategies were developed to find conformations of minimum energy, using the HP model. These approaches use genetic algorithms [25], ant colony optimization [23], [24], estimation distribution algorithms [**?**], among others[8]

Although, there are different strategies already proposed for the PSP, all of them face difficulties to reach the optimal configurations when the length of amino-acids sequences increases. This motivate the study here presented. The goal is to use a pool of heuristics proposed in the literature managed by a hyper-heuristic strategy [2]. Hence, different heuristics have different strengths and weakness. It makes sense to merge them into one framework and let a high-level strategy to manage the application of them.

Burke et al. [4] recently defined hyper-heuristics as ”an automated methodology for selecting or generating heuristics to solve hard computational search problems”. Over the years, these methodologies have demonstrated success in solving a wide range of real world problems. However, there are a lack of works that use this approach to solve the PSP problem.

The paper describes the automated generation of hyper-heuristics to solve the PSP problem. The hyper-heuristic includes two main components: a selection mechanisms and acceptance criteria. Both components are generated using grammatical evolution(GE), which is a kind of genetic programming (PG). A set of experiment is made to evaluate the approach using eleven PSP instances. The results are compared with the results of the literature.

This paper is organized ?

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.

Sequences are formed by combining amino acids and are considered the primary structure of proteins. The secondary structure is the locally ordered structure brought via hydrogen bounding mainly within the peptide backbone. The most common secondary structure elements in proteins are the alpha helix and the beta sheet. The global folding of a polypeptide chain is also called as the tertiary structure.

Under specific conditions, the protein sequence folds into a unique native 3-D structure. Each possible protein fold 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 [5], [10], [11], [14], [25] 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

The protein structures are very complex. Detailed representations of proteins exists and can be used to model the protein folding, these representations are computationally very costly. Having this in mind, Lau and Dill [13] 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 two types of residues: hydrophobic (H) residues and hydrophilic or polar (P) residues. A protein is considered a sequence of these two types of residues, which are located in regular lattice models forming self-avoided paths. 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).

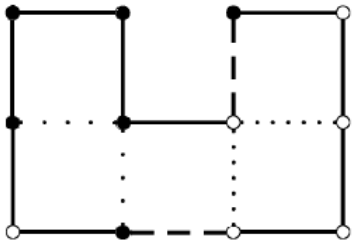

Fig. 1. One possible configuration of sequence $PPPPHPHHHHPH$ in the HP model. There is one HH (represented by a dotted line with wide spaces), one HP (represented by a dashed line) and two PP (represented by dotted lines) contacts.

For the HP model, an energy function that measures the interaction between topological neighbor residues is defined as $\epsilon_{HH} = -1$ and $\epsilon_{HP} = \epsilon_{PP} = 0$. The HP problem consists of finding the solution 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. Figure 1 shows the graphical representation of a possible configuration for the sequence $PPPPHPHHHHPH$ in a 2D space. The energy that the HP model associates with this configuration is -2 TODO: fixme because there is only two HH contact, arisen between the second and fifth residues.

Among many works that explores the PSP problem, here are some examples of the approaches that have been used to solve it.

A very influential paper from Unger and Moult [25], introduces a evolutionary algorithm (EA) which uses heuristic-based crossover and mutation operators for the HP model. The algorithm outperformed many variants of Monte Carlo methods for different instances. However, good results were obtained, the EA was unable to find the global optimal for the longest instances considered.

A multimeme algorithm (MMA) presented in [11] is a EA combined with a group of local search methods. For each individual in the population the MMA, selects a local search method that best fits. 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 [10], the author uses pruned-enriched Rosenbluth method (PERM) also know 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) was applied to the PSP problem using the HP-2D model in [23], [24]. This strategy, uses artificial ants to build conformations for a given HP instance. A local search method is then applied to further improve the solutions and also maintain the quality of the solutions.

The work of [21] utilizes Estimation of distribution algorithms (EDAs) as an efficient evolutionary algorithm that learns and exploits the search space in the form of probabilistic dependencies. New ideas were introduced for the application of EDAs to 2D and 3D simplified protein folding problem. The relationship between this proposal and other population-based approaches for the PSP was analyzed. The obtained results showed that EDAs can achieve superior solutions compared with other well-known population based optimization algorithms.

The present paper proposes the use of a grammatical evolution GE to generate high level heuristics for a hyper heuristic framework that will be applied to the PSP problem.

III. Background

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

A. Hyper-Heuristics

Burke et al. [4] recently defined hyper-heuristics as "an automated methodology for selecting or generating heuristics to solve hard computational search problems". Mainly, a generic hyper-heuristic framework is composed of two main components known as high-level and low-level heuristics. Figure III-A shows a general scheme of hyper-heuristic frameworks. The high and low level 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, to apply a hyper-heuristic framework to a different problem domain: it is possible to do it only by replacing the low-level component no changes should be required at the high-level. It is responsibility from the high-level component to 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 [20].

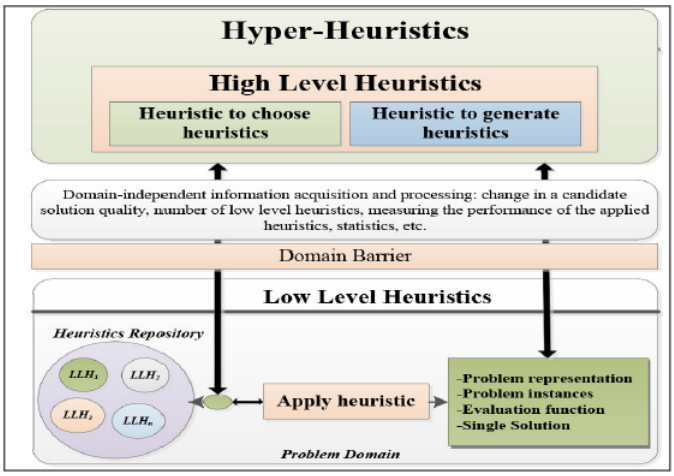

Fig. 2. General scheme of hyper-heuristic framework

Recently Burke et al. [4] classified the hyper-heuristics framework based on the nature of the search space: can be either selecting or generating heuristics for the underlying problem. Next will be analyzed both kinds of high-level heuristics.

- Selecting Heuristics: The majority of hyper-heuristic frameworks published define high level heuristics to select low level heuristics. In general, these frameworks operate using a set of human-designed low level heuristics, usually called pool of heuristics.
  The objective of the hyper-heuristic framework is to select the heuristic, from the pool, which most suits at a given moment. The main idea behind this: the strength of several heuristics can be combined into a single framework in order to better explore the search space.
- Generating Heuristics: In this case, the hyper-heuristic framework starts with a set subcomponents from low level heuristics and have the goal of fabricate new low level heuristics with it. Genetic programming is reported as a good strategy to combine and generated new heuristics for the SAT, scheduling and bin-packing problems [20].

A novel work presented by Sabar et al. [20], uses GEP (gene expression programming a kind of genetic programming) with the goal of generating the components (high level heuristics) to a hyper-heuristic framework. The experiments presented by Sabar et al. [20], using the 6 problem domains provided by the HyFlex hyper-heuristic framework, shown a impressive results comparing with other human-designed hyper-heuristic strategies from the state of art from the field of hyper-heuristics. This work inspired the present paper which has the objective of generating selection mechanism and acceptance criteria, through a grammatical evolution process, to a hyper-heuristic framework.

B. Genetic Programming

Genetic Programming (GP) [3] is a sub-field from the program synthesis which uses ideas from the evolution theory to produce programs.

Grammatical Evolution (GE) is a relatively new technique from the evolutionary computing, introduced by Ryan el al. [19] and it is a type of GP which utilizes a grammar to generate programs. Just like in GP the main goal is to find a executable program or piece of code, which have a good fitness value. Ryan el al. [19] proposes a technique to generate programs or fragments for any language using BNF (Backus Naur Form) definitions. This technique can be used to evolve programs though a evolution process. The GE uses a mapping mechanism between the genotype (coded individuals by integer vectors) and phenotype (generated programs to resolve a problem). The BNF notation is used to represent a grammar from a language in form of production rules. A BNF grammar consists in a set of terminals, which are items that are allowed to 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, S, where N is a set of non-terminals, T a set of terminals, P a set of production rules that maps the elements N to T; finally, S, a symbol to represent the initial point contained in N.

$$N = \langle expr \rangle, \langle op \rangle, \langle pre-op \rangle$$
$$T = Sin, Cos, Tan, Log, +, -, /, *, X$$
$$S = \langle expr \rangle$$

And P can be represented as:

TABLE I
PRODUCTION RULES AND THE NUMBER OF CHOICES ALLOWED

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

Ryan et al. [19] proposed the use of genetic algorithm (GA) to control which choices should be made, in this sense allowing the GA to select which production rules should be

$\langle expr \rangle$ ::= $\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$ (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)
| Cos (3)

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

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

utilized. An individual (chromosome) consists in variable-length integer vector and it is called genotype. Algorithm 1 presents the pseudo code of a grammatical evolution program. Except for lines 11, 12 and the introduction of grammar file to decode the integer vectors into programs, the GE looks very similar to a simple GA.

Algorithm 1: Pseudo code from the Grammatical Evolution

```
1  Input: GF – Grammar File
2  begin
3      population ← Create population;
4      programas ← Maps the population to programs using GF;
5      Execute the program;
6      Assign fitness value to 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 to crossover;
9          offspring ← Crossover(pais);
10         populacao ← Replacement;
11         Apply the Prune operator to the offspring;
12         Apply the Duplicate operator to the offspring;
13         Apply the Mutation operator to the offspring;
14         programs ← Maps offspring to programs using GF;
15         Execute programs;
16         Assign fitness value to solutions offspring according with
           the output obtained by the respective decoded program;
17         population ← Replacement;
18     end
19     return Best program from the population;
20 end
```

For the sake of comprehension the chromosome mapping process will be demonstrated using the Grammar 1. The Algorithm 2 presents the general template of the generated programs. The expression $\langle expr \rangle$ presented in line 2 is replaced by mathematical expressions coded by the chromosomes (integer vectors).

Algorithm 2: General template for the generated algorithms

```
1 float symb(float x)
2 a = ⟨expr⟩;
3 return a;
```

Now suppose the following integer vector:

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

This vector will be utilized to decode the chromosome (genotype) into a piece of code (phenotype) using the BNF grammar. 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 select which option, the first value from the vector should be used. The value is 220 and its mod 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 applies its mod by the amount of options. The mod from the next value from the vector is 203 mod 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 be 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 this 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 re-utilize the genes starting from the first one.

IV. Methodology

In this section will be presented the methodology defined for the grammatical evolution application for generating high level heuristics for a hyper-heuristic framework to the Protein Structure Prediction problem. The approach presented next is based on Sabar et al. [20] work.

As mentioned before a hyper-heuristic framework is divided in two levels: high and low level heuristics. In this paper, the high level heuristics are composed by a selection mechanism and an acceptance criterion. The low level heuristics consists in a set of heuristics, selected from previous works, a memory mechanism and a fitness function.

Two terminal sets (one for the selection mechanisms and another for the acceptance criteria) were defined accordingly

with the information that can be calculated within the history of the low level heuristics executions.

The selection terminals is presented next:

- 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 generated solution by the i_{th} heuristic.
- C_{best}: Number of times that the i_{th} heuristic updated the best known solution. This terminal is useful to systematically improve the current local minimum.
- $C_{current}$: Number of times that the i_{th} heuristic updated the current solution. This terminal is useful to keep the search near from the current solution.
- C_{accept}: Number of times that the generated solution by the i_{th} heuristic was accepted by the acceptance criterion. This terminal flavors heuristics that can escape from local minimum.
- C_{ava}: The average of previous improvements made by the i_{th} heuristic during the search progress. This terminal flavors heuristics that made big improvements in average.
- C_r: Number of times that the i_{th} heuristic was classified as the first.

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.
- TI: Total number of iterations.

The terminal set data was initialized executing all heuristic once and calculated the data to fill the values in the terminal sets. Every consecutive iteration will update the terminal set data and will be used during the search progress.

Using these statistic data and a function set of contained the following operations: addition, subtraction, multiplication and division, a grammar was designed to support the generation of the high level heuristics. The designed grammar to generate selection mechanisms and acceptance criteria is presented in the Grammar 2.

Using the Grammar 2 and integer vectors is possible to generate high level heuristics. The terminal sets from the grammar presents statistic information from the history of the low level heuristics application. This information is used as raw material for building the selection mechanisms and the acceptance criteria to a hyper-heuristic framework.

The next step consists of evolving a population of integer vectors, initially random generated, using the evolution process described in Section **??**.

A. Fitness Function

In order to evaluate the generated individuals during the search progress, a fitness function was designed. The fitness

⟨*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

function consisted on running the hyper-heuristic framework against three random selected instances from a set of eleven HP instances used in the previous works. Each run will be executed for one minute 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, it 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. The motivation behind executing the high level heuristic (individual) against three HP instances is that executing with only one it might not be sufficient to train a high level heuristic to obtain good results in various instances of the HP model.

B. Stopping Criterion

To stop the GE process a maximum number of evaluations was setup to 60000. This value was defined based on the work [19] where the grammatical evolution general process is first introduced.

C. Low Level Heuristics

The low level heuristics in this paper are composed by two main components: a set of low level heuristics, that were selected from previous studies using the PSP, a backtrack repair mechanism and a memory mechanism.

1) Low Level Heuristics Set:

- *Two Points Crossover* (2X): This operator selects, randomly, two crossing points splitting the individuals in 3 pieces. The genes between the selected positions are exchanged between the parents in order to generate the offspring. [1].
- *Multi Points Crossover* (MPX): Similar with 2X although with c points of crossing. The c is calculated using $c =$

$int(n * 0.1)$, where n is the sequence length. The MPX is useful to promote a wide structural diversity [20].

- *Segment Mutation* (SMUT): It changes a random number (5 to 7) of consecutive genes to distinct directions. This heuristic introduces large changes in the conformation, and it has a great probability of creating collisions.
- *Exhaustive Search Mutation* (EMUT): This heuristic selects a random gene and changes to all possible directions and it will keep the change that achieve improvement. This operator has a trade-off because it demands four fitness evaluations instead of just one. However, this heuristic has great potential of improving the input solution.
- *Local Move Operator* (LM): This heuristic exchanges directions between two consecutive random selected genes. There are some conditions in order to execute this heuristic, for instance, the new directions can not create redundant movements. This heuristic introduces a "corner" movement.
- *Loop Move Operator* (LPM): Similar with LM, this heuristic exchanges directions between two genes that are five genes of distance between each other, creating a loop movement.
- *Opposite Mutation* (OM): This heuristic exchanges directions, for the opposite, within a sequence of genes between two genes (i, j) randomly selected. The direction 0 (F) does not have a opposite, therefore it is maintained.

V. Experiments

In this section, we will discuss the conducted experiments in order to evaluate the methodology proposed by this paper. Three GE were executed and will be described next. All experiments were executed 30 times because of the stochastic behavior of the GE.

The first experiment (GExp-1) that was executed consisted on executing the GE only generating selection mechanisms. Just one piece of the high level heuristic from the hyper-heuristic frameworks was evolved, the acceptance criterion was fixed, within the hyper-heuristic framework HyPDP, in order to evaluate the ability of generating selection mechanisms isolated from acceptance criteria. An adaptation of the Improvement Only (OM) [2], which also accepts generated solutions with the same fitness value, was used as the acceptance criterion. The second GE experiment (GExp-2) consisted on generating only acceptance criteria. Fixing the selection mechanism, within the hyper-heuristic framework, with the better (the one with higher fitness from the 30 executions) selection mechanism generated by the GExp-1. Finally, the third GE experiment (GExp-3) was executed generating both selection mechanisms and acceptance criteria. Table II summarizes the average, standard deviations, max and min fitness values from 30 executions from the tree experiments.

Considering that the range of the fitness can vary from 0 to 3, as described in Sub-section IV-A it is possible to see looking Table II that the GExp-1 and GExp-2 obtained higher average values in relation to GExp-3. The Kruskal and Wallis

TABLE II
Average, Std Dev, Max and Min values of the 30 executions of the GE experiments

	GExp-1	GExp-2	GExp-3
Average	2,248	2,281	1,997
Std Dev	0,117	0,160	0,152
Max	2,433	2,453	2,310
Min	2,001	1,894	1,847

TABLE III
Results from the best individual found in GExp-1
$RC * Ccurrent * Cava - Cr$

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	7	13	19	20	33	32	46	40	41
O(x*)	9	9	8	14	23	21	36	42	53	48	50

showed statistical difference between GExp-1 x GExp-3 and GExp-2 x GExp-3 but no for the GExp-1 x GExp-2.

This indicated that generating the high level heuristics separately was more effective instead of generating both. The GExp-1, GExp-2 and GExp-3 was executed to evolve the high level heuristics using only three instances from the PSP problem as training phase. The next step consisted on selecting the best individual found, in the 30 executions of the 3 experiments, and executing against all instances presented in the section IV.

The best individual found in the GExp-1 was the following selection mechanism: $RC * Ccurrent * Cava - Cr$ and it was executed 30 times with a time limit of 10 minutes against the eleven HP instances. Table III presents the average,standard deviation ,minimum and maximum of 30 executions of the best individual found in the GExp-1 experiment. The last row denoted with O(x*) is the best known value for each sequence. It is possible to note analyzing the two last lines that only for the smaller instances the generated selection mechanism achieved good results. For the sequences 8,9,10 and 11 the results obtained are very far from the best known results.

The best individual found in the GExp-2 was the following acceptance criterion: $((TI/Delta)/((Delta * ((TI/Delta)/CI) * Delta/Delta * TI) - CI))$ and combined with the best individual from the GExp-1 the hyper-heuristic framework was executed 30 times for each one of the eleven instances with a time limit of 10 minutes. Table IV presents the results for the best generated acceptance criterion in GExp-2 and also using the best selection mechanism generated in GExp-1. Again, looking to the 2 last lines it is possible to realize that the hyper-heuristic framework, using the best generated selection mechanisms and acceptance criterion, was not able to reach the best known results for the larger sequences. And comparing Tables III and IV it is possible to visualize that there is a slight difference between the results, favoring the generated selection mechanism using the IO adaptation as acceptance criterion.

The best individual generated by the experiment GExp-3 was both a selection mechanism and an acceptance criterion.

TABLE IV
RESULTS FROM THE BEST INDIVIDUAL FOUND IN GEXP-2
$((TI/Delta)/((Delta*((TI/Delta)/CI)*Delta/Delta*TI)-CI))$

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 GEXP-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

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 GExp-3, of 30 executions with a time limit of 10 minutes. Once again the values presented did not achieve good results when analyzing the bigger instances. Furthermore, when comparing Table V with Tables III and IV it is possible to see that generating both selection mechanisms and acceptance criteria it is worst than generating them separately.

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 GExp-2 10 random individuals were selected to be further analyzed in detail. This 10 individuals were re-executed in debugging mode in order to check its behavior. Note that only the acceptance criterion was different between then because the selection mechanism was fixed, using the best generated in the previous experiment GExp-1. From 10 individuals 7 were accepting only better or equal solutions just like the fixed acceptance criterion (adapted IO to accept equal solutions) that was used in the GExp-1. The difference between the individuals and a fixed acceptance criterion was that individuals were slower than the fixed, because it is required to execute arithmetical functions and in the other hand only a simple if was needed to be evaluated. But thinking in the behavior they were exact the same. It was also noticed that 2 individuals, the worst of the group, were always accepting just like the all moves described by Burke et al. [2]. Finally, 1 individual was never accepting any solution. This experiments showed that the EG managed, several times, to find different acceptance criteria with the same behavior of human-designed strategies.

In order to compare the generated high level heuristics with a already proposed hyper-heuristic strategy it was selected a state-of-art human-designed hyper-heuristic framework [15] which presented the best results applied to the 6 domain problems from HyFlex framework [16]. Misir et al. [15] provided us the source code from the Generic Intelligent Hyper-heuristic (GIHH) and then it was executed against the PSP problem that we implemented using the HyFlex framework.

TABLE VI
BEST RESULTS FOUND BY THE BEST GENERATED HIGH LEVEL HEURISTIC B_HLH, BEST RESULTS FOUND BY GIHH AND THE BEST KNOW RESULTS O(X*)

Inst	1	2	3	4	5	6	7	8	9	10	11
B_HLH	9	9	7	13	19	20	33	32	46	40	41
GIHH (max)	9	9	8	13	22	21	35	37	49	43	45
O(x*)	9	9	8	14	23	21	36	42	53	48	50

Table VI presents the best results, for each instance, found by the best generated high level heuristic (B_HLH), the best results found by the GIHH [15] and finally the best know results (O_x(*)). The B_HLH and the GIHH were executed 10 minutes. It is possible to note that the B_HLH found worst results than the GIHH as the sequences length increase. However, the GIHH is a human-designed hyper-heuristic which demanded several years of research in order to be completed and achieve good results in the 6 domains problems provided by HyFlex. Although, the good performance obtained by the GIHH in the domains from HyFlex, it is possible to realize that the GIHH was also unable to achieve the best known results for the PSP domain.

VI. CONCLUSION

In this work, the automatic design of high level heuristics to a hyper-heuristic framework to the PSP problem was explored. The PSP is very challenging problem with lots of local optima and a very complex landscape. Many authors explored the PSP problem with heuristic methods. However, the majority have difficult to achieve the best known results when executing against longer sequences. Usually the hyper-heuristic frameworks fits well in this kind 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 within a set of eleven HP instances. The GE was 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 again in using the eleven HP instances.

Three GE experiments were designed: 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. Unfortunately when analyzing the behavior of the generated high level heuristics against the eleven instances it was possible to see that they were not able to achieve the best known results for the longer sequences. However, when comparing with a good state-of-art human-designed hyper-heuristic framework (GIHH) [15] the results are similar. This fact shows that it is possible to automate the creation of

high level heuristics and obtain results close to the state-of-art hyper-heuristics frameworks.

Another finding of this work was the behavior of the bests generated acceptance criteria. It was noticed that the bests of them behave just like "a better or equal" human-designed move acceptance strategy. Also some of the generated acceptance criterion were always accepting worst solution and it was noticed that this decreased the fitness of the individual whom coded the genotype. Other generated acceptance criteria were never accepting any worst solutions. This demonstrates that the GE is able generate at least three types of acceptance criterion.

REFERENCES

[1] César Manuel Vargas Benítez. Um algoritmo genético paralelo para o problema de dobramento de proteínas utilizando o modelo 3dhp com cadeia lateral. 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] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R Woodward. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*, pages 177–201. Springer, 2009.

[4] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[5] Fábio L Custódio, Hélio JC Barbosa, and Laurent E Dardenne. Investigation of the three-dimensional lattice HP protein folding model using a genetic algorithm. *Genetics and Molecular Biology*, 27(4):611–615, 2004.

[6] 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.

[7] Ian W Davis and David Baker. Rosettaligand docking with full ligand and receptor flexibility. *Journal of molecular biology*, 385(2):381–392, 2009.

[8] Paulo HR Gabriel, Vinícius V de Melo, and Alexandre CB Delbem. Algoritmos evolutivos e modelo hp para predição de estruturas de proteínas. *Revista de Controle e Automação*, 23(1):25–37, 2012.

[9] Mario Garza-Fabre, Gregorio Toscano-Pulido, and Eduardo Rodriguez-Tello. Locality-based multiobjectivization for the hp model of protein structure prediction. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 473–480. ACM, 2012.

[10] 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.

[11] 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.

[12] Elmar Krieger, Keehyoung Joo, Jinwoo Lee, Jooyoung Lee, Srivatsan Raman, James Thompson, Mike Tyka, David Baker, and Kevin Karplus. Improving physical realism, stereochemistry, and side-chain accuracy in homology modeling: four approaches that performed well in casp8. *Proteins: Structure, Function, and Bioinformatics*, 77(S9):114–122, 2009.

[13] 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.

[14] Cheng-Jian Lin and Shih-Chieh Su. Protein 3 d hp model folding simulation using a hybrid of genetic algorithm and particle swarm optimization. *International Journal of Fuzzy Systems*, 13(2):140–147, 2011.

[15] Mustafa Misir. Intelligent hyper-heuristics: a tool for solving generic optimisation problems. 2012.

[16] 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.

[17] Bin Qian, Angel R Ortiz, and David Baker. Improvement of comparative model accuracy by free-energy optimization along principal components of natural structural variation. *Proceedings of the National Academy of Sciences of the United States of America*, 101(43):15346–15351, 2004.

[18] Daniela Röthlisberger, Olga Khersonsky, Andrew M Wollacott, Lin Jiang, Jason DeChancie, Jamie Betker, Jasmine L Gallaher, Eric A Althoff, Alexandre Zanghellini, Orly Dym, et al. Kemp elimination catalysts by computational enzyme design. *Nature*, 453(7192):190–195, 2008.

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

[20] 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.

[21] Roberto Santana, Pedro Larrañaga, Jose Lozano, et al. Protein folding in simplified models with estimation of distribution algorithms. *Evolutionary Computation, IEEE Transactions on*, 12(4):418–438, 2008.

[22] Yang Shen, Robert Vernon, David Baker, and Ad Bax. De novo protein structure generation from incomplete chemical shift assignments. *Journal of biomolecular NMR*, 43(2):63–78, 2009.

[23] Alena Shmygelska, Rosalia Aguirre-Hernandez, and Holger H Hoos. An ant colony optimization algorithm for the 2d hp protein folding problem. In *Ant Algorithms*, pages 40–52. Springer, 2002.

[24] 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.

[25] Ron Unger and John Moult. Genetic algorithms for protein folding simulations. *Journal of molecular biology*, 231(1):75–81, 1993.

[26] Ling Wang, Eric A Althoff, Jill Bolduc, Lin Jiang, James Moody, Jonathan K Lassila, Lars Giger, Donald Hilvert, Barry Stoddard, and David Baker. Structural analyses of covalent enzyme–substrate analog complexes reveal strengths and limitations of de novo enzyme design. *Journal of molecular biology*, 415(3):615–625, 2012.