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*—The Protein Structure Prediction (PSP) problem is one the modern most challenging problems from science. Hence, many heuristics strategies have been applied in order to find protein structures that minimize its free energy. However, these strategies have difficulties on 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. The hyper-heuristics framework are usually useful in this kind of context since they try to combine different heuristics strengths into a single framework. However, there is lack of works whose aim the automated design of hyper-heuristics components. This paper proposes the GEHyPSP which aims to the generation, through grammatical evolution, of selection mechanisms and acceptance criteria for a hyper-heuristic framework applied to PSP problem.**

I. Introduction

Proteins execute an essential role in nature, they are responsible of many important functions of the living cells. Proteins can be seen as amino-acids structures that guarantee the correct operation of many areas 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.

The protein structure prediction (PSP) has a broad range of medical and biotechnology applications. For instance: synthesis of new proteins and folds [29], structure based synthesis of new drugs [8], refinement of theoretical models obtained by comparative modeling [21], [16], and obtaining experimental structures from incomplete nuclear magnetic resonance data [25].

The task of determining the native structures of proteins is challenging even for the modern super computers. The difficulty raises up due to the huge search space to test all possible conformation that a sequence of amino-acids can adopt. There are many models to represent proteins structures and can be used to simulate the folding process. Extremely detailed models exist however these representation are computational very expensive. Hence, many authors [6], [12], [18], [28], [24], [7], [11] have used simplified models to represent the protein structures. A very common model for this purpose is the Hydrophobic-Polar (HP) model presented by Lau and Dill [17]. This model abstract the amino-acids into just two types: hydrophobic (H) or polar (P). Two-dimensional (2D) or three-dimensional (3D) grids can be used to represent the conformations that a protein can adopt. Each conformation has an associated energy [28].

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 [28], ant colony optimization [26], [27], estimation distribution algorithms [24], and others [9]. Even though there are different strategies already proposed for the PSP, all face difficulties to reach the optimal conformations when the length of amino-acids sequences increases.

This fact motivates the study here presented. In this kind of context that hyper-heuristics are usually a good option in order to better explore a complex search space. In general the hyper-heuristics frameworks propose a strategy to select, from a pool, the heuristic which is more appropriate for a given moment. Combining the strengths and weakness of a set of heuristics. Also the hyper-heuristics frameworks define a move acceptance criterion, which is responsible to decide on whether accept or reject when worst solutions are generated by the heuristics. Theses strategies are high level heuristics from a hyper-heuristic framework.

Usually the high level heuristics are human-designed approaches although a study [23] propose the automate design of those components. This study presented promising results and so inspired the idea of automating the design of high level heuristics to a hyper-heuristic framework to 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 when design a hyper-heuristic strategy to the PSP. Another motivation of the present study is that the hyper-heuristics frameworks have been applied to other domains problems and achieving good results in the bin packing, personnel scheduling, flowshop, TSP, MAXSAT and VRP problems [23].

Burke et al. [5] 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 is not an in-depth investigation of the suitability of this approach to solve the PSP problem.

We introduce the GEHyPSP: an automated mechanism for generation of high level heuristics to a hyper-heuristic

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 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 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, 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 [23].

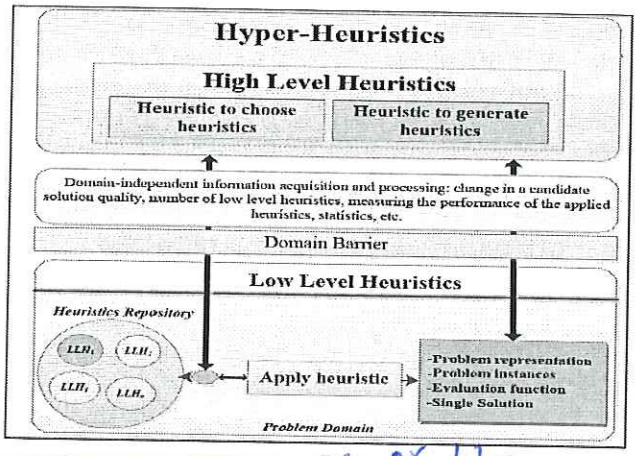

Fig. 2. General scheme of hyper-heuristic framework.

Based on the nature of the search space, hyper-heuristics can be classified [5] as 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 define high level heuristics to select low level heuristics. 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 [3]. In contrast, the choice function [3] 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 subcomponents from 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 [23].

A study presented by Sabar et al. [23], proposes an on-line approach to generate selection mechanisms and acceptance criteria (high level heuristics) to a hyper-heuristic framework, using GEP [9] (gene expression programming, a variant of GP). The experiments presented by Sabar et al. [23], using the 6 problem domains provided by the HyFlex hyper-heuristic framework [20], showed impressive results in comparison with other human-designed hyper-heuristic strategies from the state-of-art. This encouraging result inspired the present study which has the objective of off-line generate selection mechanisms and acceptance criteria, through a grammatical evolution process, for a hyper-heuristic framework to solve PSP problem.

B. Genetic Programming (GP)

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

Grammatical Evolution (GE) is a relatively new technique from the evolutionary computing, introduced by Ryan el al. [22] and it is a type of GP where programs are evolved using a genetic algorithm. The chromossome encode 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 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, 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.

TABLE I
PRODUCTION RULES AND THE NUMBER OF CHOICES ALLOWED

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

Ryan et al. [22] 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 utilized. An individual (chromosome) consists in variable-length integer vector and it is called genotype. Algorithm 1 presents the pseudocode of a grammatical evolution program. Lines 3 to 6 represents the initialization of the algorithm: a randomly generated population is created and then mapped to

heuristics use information related with 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 with the information that can be extracted during the search progress. The selection terminals are 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 solution generated 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 to 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 favors 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 favors heuristics that made big improvements in average.
- 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.
- TI: Total number of iterations.

Using these statistics as terminals and a function set containing the following arithmetic 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.

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

A. Fitness Function

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

⟨*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

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 [22] 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. [15] the relative representation has a better potential to achieve superior results. The relative representation defines that each gene of the chromosome represents a direction that in the grid will represent a movement that an conformation could adapt. The directions in the grid for each amino-acid are represented always based on the previous. 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 $\{0, 1, 2\}$.

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	7	13	19	20	33	32	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

each sequence. It is possible to note analyzing the last two 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.

B. Results from GEHyPSP-2

The best individual found in the GEHyPSP-2 was the following acceptance criterion: $((TI/Delta)/((Delta * ((TI/Delta)/CI) * Delta/Delta * TI) - CI))$ 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 10 minutes. Table IV presents the results for the best generated acceptance criterion in GEHyPSP-2 and also using the best selection mechanism generated in GEHyPSP-1. Again, looking to the two last lines 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 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 "better or equal" acceptance criterion.

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 10 minutes. Once again the values presented did not achieve good results when analyzing the most difficult instances. Furthermore, 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.

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	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

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

In order to compare the generated high level heuristics with a already proposed hyper-heuristic strategy a state-of-art human-designed hyper-heuristic framework [19] was selected. This framework produced the best results applied to the 6 domain problems from HyFlex framework [20]. Misir et al. [19] 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 [19] and finally the best know results (O_x(*)). The GEHyPSP and the GIHH were executed 10 minutes. It is possible to note that the GEHyPSP found worse 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 achieved good results in the six domains problems provided by HyFlex. Despite the good performance obtained by the GIHH in the domains from HyFlex, it is possible to notice that the GIHH was also unable to achieve the best known results in many instances.

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 analyzed. These individuals were re-executed in 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 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 it is required to execute arithmetical functions and in the other hand only a simple if was evaluated. But thinking in the behavior they were exactly the same. It was also noticed that 2 individuals, the worst of the group,

ming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, 2015.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.