MATHEMATICAL MODELING AND COMPUTING, Vol. 8, No. 4, pp. 736–746 (2021)

A modified choice function hyper-heuristic with Boltzmann function

Mellouli O., Hafidi I., Metrane A.

LIPIM, ENSA Khouribga, University Sultan Moulay Slimane,
Bd Béni Amir, B.P. 77, Khouribga, Morocco

(Received 23 May 2021; Accepted 7 June 2021)

Hyper-heuristics are a subclass of high-level research methods that function in a low-level heuristic research space. Their aim objective is to improve the level of generality for solving combinatorial optimization problems using two main components: a methodology for the heuristic selection and a move acceptance criterion, to ensure intensification and diversification [1]. Thus, rather than working directly on the problem's solutions and selecting one of them to proceed to the next step at each stage, hyper-heuristics operates on a low-level heuristic research space. The choice function is one of the hyper-heuristics that have proven their efficiency in solving combinatorial optimization problems [2–4]. At each iteration, the selection of heuristics is dependent on a score calculated by combining three different measures to guarantee both intensification and diversification for the heuristics choice process. The heuristic with the highest score is therefore chosen to be applied to the problem. Therefore, the key to the success of the choice function is to choose the correct weight parameters of its three measures. In this study, we make a state of the art in hyper-heuristic research and propose a new method that automatically controls these weight parameters based on the Boltzmann function. The results obtained from its application on five problem domains are compared with those of the standard, modified choice function proposed by Drake et al. [2,3].

Keywords: *hyper-heuristics, combinatorial optimization, choice function, modified choice function, heuristic selection, heuristic generation, Boltzmann function.*

2010 MSC: 35Q20, 90C27, 90C59 **DOI:** 10.23939/mmc2021.04.736

1. Introduction

For several years, heuristic methods were a success in the field of combinatorial research, they showed efficiency in solving difficult real problems [1, 5]. However, they presented several limitations and difficulties where new problems come up, so it is necessary to find how to adapt them to its structure or even to the resolution of several instances of the same problem. And here came the motivation behind hyper-heuristics, which main purpose is to automate the design and adaptation of heuristics for solving difficult combinatorial research problems by raising the level of abstraction and generality in the research process. Hyper-heuristics as a promising field was considered in several research studies [1]. Hyper-heuristics take a set of heuristics as a search space in order to select or generate in each iteration the best heuristic to apply to converge to solutions with acceptable quality. So, given one or more problems and one or more different instances, hyper-heuristics generates an suitable combination of these components based on a variety of low-Level heuristics to efficiently solve the given problem [1–3].

The next sections will discuss the evolution of hyper-heuristics over the years, so section 2 will present a state-of-the-art of hyper-heuristic research by presenting their new definition as well as their classification. Section 3 and 4 will present the standard modified choice function and the new proposed version including Boltzmann function. And the last section will be devoted to a presentation of the experimental results.

2. Hyper-heuristic research state of the art

In combinatorial optimization world, the word 'hyper-heuristic' was first used in 2001 as 'a heuristic to choose heuristics' [6]. At each decision point, the high-level heuristic selects the low-level heuristics to be used based on a predefined move acceptance criterion and a selection methodology. However, a tracking in the history of heuristic automation shows that the idea goes back to the 1960s [7–9]. Over the years, research studies in hyper-heuristics passed to a new phase which is the automatic generation of heuristics, through a high-level heuristic and by combining the components of many low-level heuristics, a new heuristic suited to the search problem is automatically generated.

In [1], Edmund K. and Burke et al. classified hyper-heuristics according to two dimensions: "the nature of the heuristic research space and the source of feedback during learning" (Edmund K. and Burke et al., 2010), Fig. 1. So instead of "heuristic to choose heuristics", the definition of the term "hyper-heuristic" has been generalized to include even automatic generation: "A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve difficult combinatorial research problems" (Edmund K. and Burke et al., 2010).

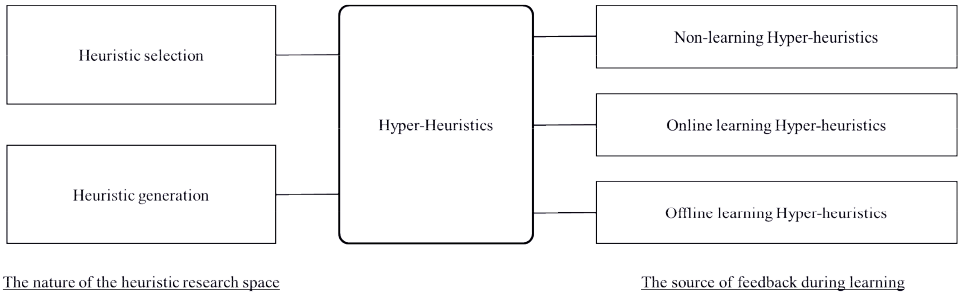

Fig. 1. Hyper-heuristic classification based on the nature of the search space of the heuristic and the source of feedback during learning.

This classification defines the dimension of "the nature of the heuristic research space" by giving two categories of hyper-heuristics [1]:

- **Heuristic Selection:** which picks and chooses a heuristic from a collection of pre-existing heuristics.
- **Heuristic generation:** which generates automatically a heuristic from the components of a collection of pre-existing heuristics.

The second dimension focuses on the learning side of hyper-heuristics. In this dimension, we have three categories [1]:

- **Hyper-heuristics with no learning:** are hyper-heuristics that do not use learning in the selection or generation process.
- **Hyper-heuristics with an online learning:** are hyper-heuristics that learn when they solve problem instance.
- **Hyper-heuristics with an Offline learning:** are hyper-heuristics who learn a mechanism to generalize to unseen instances via the learning from the execution of many instances

In the next sections, we will present some representative examples of classification based on the nature of the heuristic research space. It will not be an exhaustive survey, the aim of this section is to illustrate the implementation of hyper-heuristics

2.1. Heuristic selection methodologies

In selection hyper-heuristic, based on a selection process, with an approval judgement taken at each point based on the acceptance criteria, a low-level heuristic is selected and applied to the problem,

Algorithm 1 Hyper-heuristic Scheduling Algorithm (HHSA).

Configure the algorithm parameters
Input the problem domain
Generate the initial population of solution $Z = \{z_1, z_2, ..., z_N\}$
From the candidate pool H_i, select randomly a heuristic H
while the stop criterion is not reached
  Use the selected algorithm H_i to update Z
  $F_1 = \text{ImprovementDetection}(Z)$
  $F_2 = \text{DiversityDetection}(Z)$
  if $\psi(H_i, F_1, F_2)$ **then**
    Randomly select a new H_i
    $Z + \text{Perturb}(Z)$
Output the best solution meted.

A Hyper-heuristic Approach to Scheduling a Sales Summit [6]. The term 'Hyper-heuristic' appeared in 2001 with Colling et al. and has been applied to Scheduling a Sales Summit problem [6]. Based on two acceptance criteria, only improving that only accepts moves that improve the solution and all moves where improving and non-improving moves are all accepted, this work studied three distinct types of hyper-heuristic approaches: random approaches, greedy approaches, and choice function based approaches [6].

For the random approach, three approaches are considered [6]:

- **SimpleRandom:** at each iteration and while the stop criteria is not reached, the algorithm choose randomly a low-level heuristic to apply.
- **RandomDescent:** this approach looks like the first one, the only difference is that the randomly chosen heuristic will be executed until the met a local optimum.
- **RandomPermDescent:** this approach is identical to RandomDescent, except this time, at the beginning, we choose a random sequence of low-level heuristics and we execute a cycle to the next heuristic if the application of the current heuristic is not improving.

Greedy method executes all low-level heuristics to the current solution and chooses the one with the best solution. In this approach, the versions of the two acceptance criteria (Only Improving and All moves) are similar [6].

The third group of hyper-heuristic methods introduces a choice function F; this hyper-heuristic determines which low-level heuristic to call next. Given the current state of learning about the solution space region being explored, the choice function F calculates the probability that a particular low-level heuristic is relevant. This choice function records, for each low-level heuristic, its recent performances (f_1), about its effectiveness for a consecutive heuristic peer (f_2) the time period since the last call of the heuristic (f_3) [6].

A hyper-heuristic Scheduling Algorithm for Cloud [28]. This paper presents the Hyper-Heuristic Scheduling Algorithm (HHSA), a heuristic scheduling algorithm that guide cloud computing systems find better planning solutions. The principle behind this algorithm is to combine intensification and diversification in finding solutions during the convergence process, by dynamically determining which low-level heuristics should be used to identify better solution candidates using diversity detection and improvement detection operators, as shown in Fig. 2 [28].

At the beginning, in Line 1, the parameters of the algorithm are defined, the maximum number of iterations to run the low-level heuristic algorithm selected, and the number of iterations when the selected heuristic does not improve the solutions. Line 2 reads the search problem to be solved, Line 3 initializes the solution population $Z = \{z_1, z_2, \ldots, z_N\}$, where N is the size of the population. In line 4, a randomly low-level heuristic $H = \{H_1, H_2, \ldots, H_n\}$, simulated annealing, genetic algorithm, particle swarm optimization, and ant colony optimization are the candidate of the low-level heuristics

pool. The chosen heuristic, H_i, would then be run until the stop criterion is reached, as indicated in Lines 5–13. More precisely, to maintain a balance between the intensification and diversification of research directions, the H_i heuristic selected will evolve the Z solution for the maximum number of iterations, defined in the parameters, using the $\psi(H_i, F_1, F_2)$ determination function, as defined in equation 1 [10].

The intensification and diversification of research directions depend on the knowledge given by the improvement detection operator F_1 and the diversity detection operator F_2 (as shown in Lines 7 and 8) [10].

$$\psi(H_i, F_1, F_2) = \begin{cases} \text{False} & \text{if } H \in S \text{ and } F_2 = \text{true}, \\ \text{False} & \text{if } H \in S \text{ and } F_1 = \text{true and } F_2 = \text{true}, \\ \text{True} & \text{otherwise.} \end{cases} \quad (1)$$

Where S is the set of SSBHAs (Single-Solution-Based Heuristic Algorithms) and are only used in F_1; P is the set of PBHAs (Population-Based Heuristic Algorithms) and are used in F_1 and F_2. The proposed algorithm, when $\psi(H_i, F_1, F_2)$ returns true, will select a random Hi heuristic algorithm and return the Z solution to the perturbation operator in order to improve the output, as shown in lines 9–11 [10].

Parallel cooperative optimization through hyper- heuristics [11]. This article discusses a cooperative approach for solving optimization problems. The proposed algorithm is a hyper-heuristic that combines three well-known canonical metaheuristics of varying degrees of effectiveness. The Parallel Optimizer With Hyper-heuristics (POWH) algorithm is built on the master-slave model (Fig. 3) and consists of several metaheuristics that collaborate and function simultaneously in a distributed computing environments [11]. It is organized using an A-team architecture (Talukdar et al., 1998), with the following metaheuristics serving as autonomous agents: Simulated Annealing (SA), Genetic Algorithm (GA), and Ant Colony Optimization (ACO).

In the control scheme, a rating index (Ind), that evaluates each metaheuristic's output once a processor becomes idle, guides the selection of suitable metaheuristics. The strategy with the highest ranking will be considered for the next execution [11].

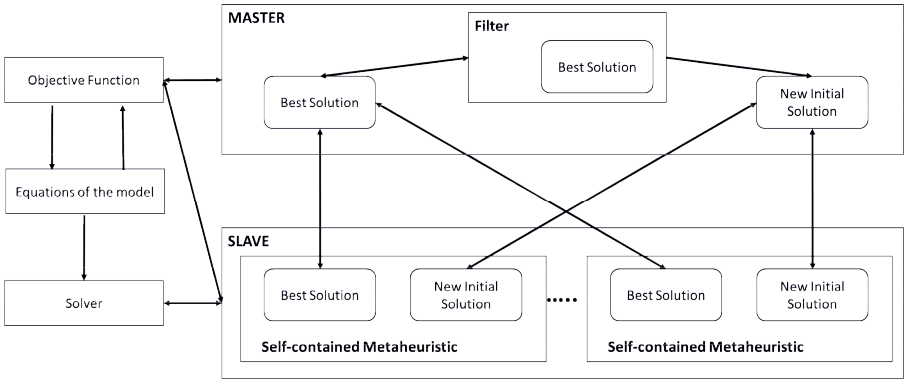

Fig. 2. Model of the Parallel Optimizer With Hyper-heuristic.

2.2. Heuristic generation methodologies.

The principle of automatic generation is to consider the determination of the algorithm as an optimization problem where we look for the algorithm that will give the maximum performance [12–16]. Genetic programming is an appropriate method of generation hyper-heuristics due to the syntax of an algorithm that can be interpreted as a tree.

Indeed, many studies in the literature use genetic programming to generate algorithms automatically [5, 17–22]. The concept is to interpret low-level heuristics using a tree. The nodes represent the algorithm's instructions, the inner nodes represent the usual high-level functions used in each com-

puter algorithm, and the leaf nodes represent functions that work on the problem's solution. As the genetic algorithm runs, the generation of heuristics evolves from one generation to another by applying mutation operators, crossover and reproduction to converge to a personalized heuristic generated automatically for the problem. This method enables hyper-heuristics to achieve a higher degree of generality and independence, as a customized heuristic is created automatically for each distinct problem, and for each distinct instance of the same problem. Fig. 4 explains the process of genetic programming. The implementation of mutation, crossover, and reproduction operators generates a new generation $P(t+1)$ from the previous generation of algorithms $P(t)$. To evaluate the output of the algorithm proposed on a group of instances of the problem, a fitness function must be determined. This method is repeated for many generations in order to converge by the end of the process to a suitable algorithm for solving the problem.

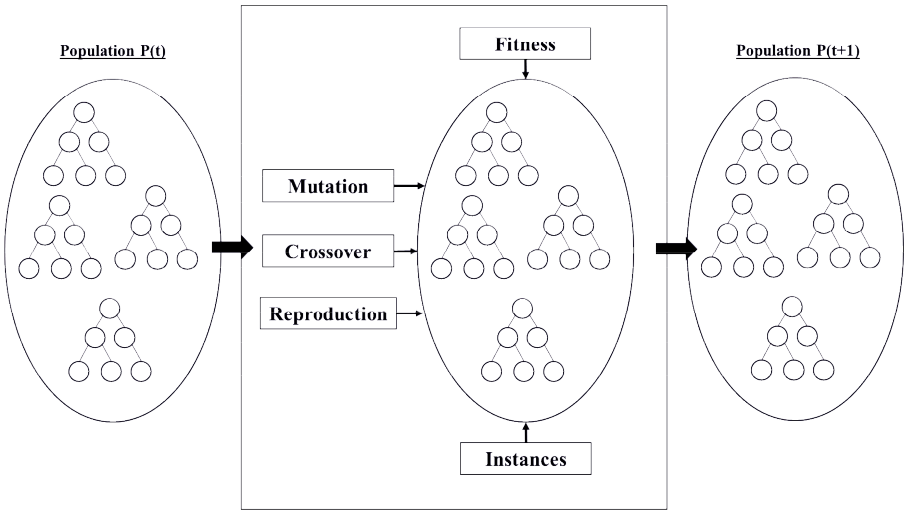

Fig. 3. Evolutionary process of the heuristic generation.

Automatic Design of Algorithms for the Travelling Salesman Problem (Nicolas Acevedo, 2020) [23]. The hyper-heuristic was used to solve the TSP problem in this work. The following elements must be defined before initiating the automatic generation phase:

- Definition of the data structure for the storage of TSP solutions:
  - Lists to stores the cities, their coordinates (x, y) and the circuit generated by the algorithm, ...
  - A matrix to store inter-city distances.
  - The central point coordinates.
- Design of the grammar of the automatic generation: in this step, two types of functions must be defined:
  - The basic functions of programming languages [23]: they will have two variables $P1$ and $P2$, considering that 1 corresponds to the true value and 0 to the false value, so any function will return true or false: While$(P1, P2)$, while $P1$ returns true, $P2$ will be executed; AND$(P1, P2)$, if $P1$ and $P2$ returns true, both of them will be executed and the function AND will returns true; IfThen$(P1, P2)$ performs $P2$ only if $P1$ is evaluated as true, ...
  - The functions of the TSP problem: the functions of this part are defined exclusively for the TSP problem. Loyola and Ai have chosen to use two types of functions: Constructive and circuit modification functions. Although the tour is incomplete, the constructive terminals insert cities to it. It returns "1" when a city is inserted to the circuit and "0" when the tour is completed. The tour modification terminals alter the tour that has been created up to this stage. If the tour has been modified, it returns "1", otherwise, it returns "0."

- Design of the fitness function [23]: Loyola and Ai chose tree metrics to measure an algorithm's performance: the algorithm's efficiency, feasibility, and size. The cost of the produced circuit is used to determine its efficiency/quality, which reflects the relative error when compared to the optimal solution of each instance. An algorithm is considered as feasible if it produces feasible solutions (acceptable tours) for the TSP. The variation of the number of cities in the created circuit is used to determine feasibility. The third metric is the algorithm's size; this metric is critical for decoding the algorithm proposed by automatic generation. The algorithm's size is then determined by the deviation of the number of nodes in comparison to a target number of nodes.
- The definition of the set of instances that will be used to evaluate the algorithm's suggested solutions.
- The definition of the parameters of the Genetic programming: Population size, probability of selection, mutation and crossover, etc.

3. The modified choice function

The modified choice function is a famous selection heuristic that ranks low-heuristics according to three distinct metrics, emphasizing the intensification and diversification parameters [2,3]. To rate heuristics at each iteration of a search, a rank is assigned to each low-heuristic using the modified choice function F calculated as:

$$f(h_i) = \Phi_t f_1(h_i) + \Phi_t f_2(h_k, h_i) + \delta_t f_3(h_i), \tag{2}$$

where t is the current iteration.

The first metric (f_1) represents the historical achievements of a low-level heuristic. For a low-level heuristic h_j, the value of f_1, described as:

$$f_1(h_j) = \sum_n \phi^{n-1} \frac{I_n(h_j)}{T_n(h_j)}, \tag{3}$$

where Φ is a value between 0 and 1 that places more focus on historical performance, $I_n(h_j)$ represents the solution value variation and $T_n(h_j)$ represents the time required to call all the past n invocations of the heuristic h_j.

The second metric (f_2) recognizes heuristics that work well when used in sequence. f_2 is defined as follows:

$$f_2(h_k, h_j) = \sum_n \phi^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)}, \tag{4}$$

where Φ has similar meaning as in f_1, $I_n(h_k, h_j)$ represents the solution value variation and $T_n(h_k, h_j)$ is the time required to call all the past n invocations of the heuristic h_j after h_k.

The time ($\tau(hj)$) since the Choice Function last selected each heuristic is the third measure (f_3). This ensures that all heuristics have a probability of being chosen.

$$f_3(h_j) = \tau(h_j). \tag{5}$$

If the performance of the solution increases at each stage, ϕ is awarded generously while δ is severely punished [2, 3]. If the solution performance degrades by using a low-level heuristic, the diversification of the research process is guaranteed by linearly decreasing ϕ and improving δ. The aim of this strategy is to consider the intensification variable major determinant in calculating F, with the parameters ϕ_t and δ_t defined as:

$$\phi_t = \begin{cases} 0.99, & \text{if quality improves,} \\ \max\{\phi_t - 0.01, 0.01\}, & \text{if quality deteriorates,} \end{cases}$$

$$\delta_t = 1 - \Phi_t, \tag{6}$$

ϕ will always have some effect on the F value since the 0.01 is used as the minimum weight.

4. The Modified Choice Function with Boltzmann Function

In the standard Modified Choice Function proposed by Drake et al. [2,3], ϕ and δ increase and decrease according to the quality of the solution with a constant equal to 0.01. In our work, we propose a new method to control these weight parameters based on Boltzmann function. Boltzmann function has shown efficiency with the Simulate Annealing algorithm [24–30] as an acceptance probability function which allows to avoid getting stuck at a local minimum. The temperature T gradually reduces so that the probability of accepting an up-hill move also gradually reduces.

With the same principle, in our work, the parameters ϕ_t and δ_t are defined as shown in the equations (12) and (13) so that they will increase and decrease gradually according to the quality of the solution and the temperature T, the temperature T gradually reduces so that the probability of accepting a non-improving heuristic also gradually reduces. This new version of the Modified Choice Function by the inclusion of Boltzmann Function allows the diversification at the beginning of the research process by starting with δ_t bigger than ϕ_t so the third measure will have the higher weight, and rewards the intensification by the end of the research process as the ϕ_t increase gradually. So the Modified Choice function will be defined as follow:

$$f(h_i) = \Phi_t f_1(h_i) + \Phi_t f_2(h_k, h_i) + \delta_t f_3(h_i). \tag{7}$$

With

$$f_1(h_j) = \sum_n \phi^{n-1} \frac{I_n(h_j)}{T_n(h_j)}, \tag{8}$$

$$f_2(h_k, h_j) = \sum_n \phi^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)}, \tag{9}$$

$$f_3(h_j) = \tau(h_j). \tag{10}$$

And

$$\delta_t = e^{\frac{-I(h_i)}{T}}, \tag{11}$$

$$\Phi_t = 1 - \delta_t, \tag{12}$$

where t is the current iteration and T is the temperature.

5. Experimental results

The suggested Modified Choice Function Hyper-heuristic with Boltzmann function (MCF-F) and the standard Modified Choice Function Hyper-heuristic (MCF-S) of Drake et al. [2,3] are compared in this section. All tests were performed over a collection of 5 known problem domains (SAT, Flow Shop, Bin Packing, Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP)), in order to judge the performance of the proposed hyper-heuristic. The results of the tests represent, for each hyper-heuristic, the average of five runs. Table 1 shows formula one score of the two hyper-heuristics, the hyper-heuristic with the best performance gets 10 points and the second gets 8 points. An evaluation of hyper-heuristics scores shows that integrating Boltzmann Function into the Modified Choice Function Hyper-heuristic leads to a significant performance when compared to Drake et al.'s Modified Choice Function hyper-heuristic [2, 3].

Using the Formula One scoring system as a to compare the two hyper-heuristics gives a good indication about their performances over all five problem domains, it allows us to see that one method excels some different domains over another. Using the Formula One scoring framework as a comparative tool provides a clear indicator of performance in all five problem domains, it also allows us to see if one method outperforms the other one in specific domains. As shown, the proposed Modified Choice Function Hyper-heuristic with Boltzmann Function outperforms the standard Modified Choice Function Hyper-heuristic in three of the five problem domains evaluated: Bin Packing, Flow Shop, and SAT.

Table 1. Formula one score of the standard and the proposed modified choice function in each problem domain.

	MCF-S	MCF-B
Bin Packing	40	**50**
Flow Shop	42	**48**
TSP	**50**	40
SAT	40	**50**
VRP	**50**	40

Table 2. Comparison of the objective function values performance of MCF-S and MCF-B.

	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5
Bin Packing	=+	s+	=+	=+	s+
Flow Shop	=+	=–	=+	=+	=+
TSP	=–	=–	=–	=–	=–
SAT	s+	s+	s+	s+	s+
VRP	s–	s–	s–	s–	s–

Table 3. The ratio of performance of the objective function value of the MCF-B when compared to the MCF-S.

	Instance 1		Instance 2		Instance 3		Instance 4		Instance 5	
	Average	Best Solution	Average	Best Solution	Average	Best Solution	Average	Best Solution	Average	Best Solution
Bin Packing	**3%**	**8%**	**13%**	**44%**	**3%**	**2%**	**1%**	**7%**	**7%**	**20%**
Flow Shop	**0.25%**	**1%**	**–0.06%**	**1%**	**0.36%**	**0.32%**	**0.53%**	**0.41%**	**0.33%**	**0.15%**
TSP	–3%	–7%	–2%	–8%	–1%	–6%	–1%	–3%	–1%	–3%
SAT	**12%**	**67%**	**16%**	**70%**	**16%**	**74%**	**15%**	**52%**	**32%**	**25%**
VRP	–9%	–56%	–13%	–40%	–20%	–61%	–18%	–66%	–20%	–22%

Every cell of Table 2 tests the difference in performance between the two hyperheuristics on the obtained objective values: s+ (s–) indicates that MCF-B (MCF-S) performs statistically considerably better than MCF-S (MCF-B), while =+ (=–) indicates there was no statistically meaningful difference in results between MCF-B and MCF-S, but MCF-B performs slightly higher (lower) on average.

Table 3 shows the ratio of performance of MCF-B when compared with the MCF-S. When the objective function values are compared, it is clear that the Modified Choice Function Hyper-heuristic with Boltzmann Function outperforms in all five instances of the SAT, with a ratio of performance that can achieve 32% on average and 74% on the best solution found by the two hyper-heuristics. Conversely, in the VRP, the standard Modified Choice Function Hyper-heuristic gives better performance in all 5 instances, with a ratio of performance that can achieve 20% on average and 66% on the best solution found by the two hyper-heuristics.

For the Bin Packing problem, the difference is statistically significant in 2 of 5 instances for the MCF-B with a ratio of performance $> 5\%$ and a slight better performance for the other 3 instances. For the other two problem domains, there is no significant performance variation, however, the MCF-B performs slightly better than MCF-S for the Flow Shop and slightly worse for TSP.

This shows that the performance of a method not only vary when the problem domain changes but also for each instance of the same problem domain. Fig. 4 shows how many times each hyper-heuristic can do better (a. on average and b. on the best solution found by the hyper-heuristic). In terms of the cumulative number of instances, where each hyper-heuristic outperformed one another, the modified choice function with Boltzmann Function outperformed the standard Modified Choice Function, in 14 cases on average and 15 cases for the best solution. Whereas, the standard Modified Choice Function outperformed the modified choice function in 11 cases on average and 10 cases for the best solution.

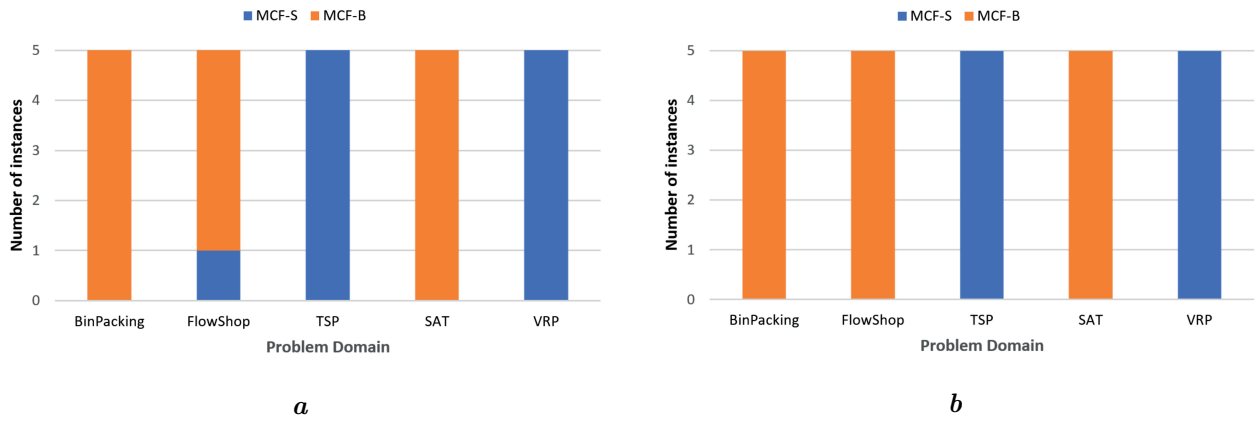

Fig. 4. Number of instances in which each Hyper-Heuristic (MCF-S and MCF-B) performs best (***a***) on average of the 5 runs and (***b***) on the best solution found by the Hyper-heuristic.

6. Conclusion

In this work, we have presented a state of the art of hyper-heuristics research by describing how their definition has evolved from a "heuristic to choose heuristics" to "an automated methodology for selecting or generating heuristics to solve difficult combinatorial search problems", and we have presented how they are classified based on this definition. We have described a new version of the modified choice function proposed by Drake et al. [2, 3] which manages in a different way the weight parameters of intensification and diversification components of the choice function by the inclusion of Boltzmann Function. The Modified Choice Function of Drake et al. [2, 3] aggressively rewards the intensification weight and heavily punishes the diversification component each time an improvement is made. This proposed Modified Choice Function by the integration of Boltzmann Function allows the diversification at the beginning of the research process and rewards the intensification by the end of the research process. We have shown that the key of success of the choice function is therefore to choose the correct weight parameters for its three measures. As future work, we plan to present different ways to manage those weight parameters to improve even more the problems resolution and to analyze which method suits better to each problem domain.

[1] Burke E. K., Hyde M., Kendall G., Ochoa G., Özcan E., Woodward J. R. A classification of hyper-heuristic approaches in Handbook of Metaheuristics. Springer US. 449–468 (2010).

[2] Drake J. H., Özcan E., Burke E. K. An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search. PPSN 2012: Parallel Problem Solving from Nature – PPSN XII. 307–316 (2012).

[3] Drake J. H., Özcan E., Burke E. K. A Modified Choice Function hyper-heuristic controlling unary and binary operators. 2015 IEEE Congress on Evolutionary Computation (CEC). 3389–3396 (2015).

[4] Tay J. C., Ho N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering. **54** (3), 453–473 (2008).

[5] Lyaqini S., Nachaoui M., Quafafou M. Non-smooth classification model based on new smoothing technique. Journal of Physics: Conference Series. **1743** (1), 012025 (2021).

[6] Cowling P. I., Kendall G., Soubeiga E. A Hyperheuristic Approach to Scheduling a Sales Summit. PATAT 2000: Practice and Theory of Automated Timetabling III. **2079**, 176–190 (2001).

[7] Crowston W. B., Glover F., Thompson G. L., Trawick J. D. Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR research memorandum, Carnegie-Mellon University, Pittsburgh (1963).

[8] Fisher H., Thompson G. L. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth J. F., Thompson G. L. (eds). Industrial Scheduling. 225–251 (1963).

[9] Fisher H., Thompson G. L. Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference, Carnegie Institue of Technology. May 10–12 (1961).

[10] Nachaoui M., Chakib A., Nachaoui A. An efficient evolutionary algorithm for a shape optimization problem. Applied and Computational Mathematics. **19** (2), 220–244 (2020).

[11] Oteiza P. P., Rodriguez D. A., Brignole N. B. Parallel cooperative optimization through hyper-heuristics. Computer Aided Chemical Engineering. **44**, 805–810 (2018).

[12] Burke E. K., Hyde M., Kendall G., Ochoa G., Ozcan E., Woodward J. Exploring hyper-heuristic methodologies with genetic programming. Computational Intelligence. 177–201 (2009).

[13] Burke E. K., Hyde M. R., Kendall G., Woodward J. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. 1559–1565 (2007).

[14] Burke E. K., Hyde M. R., Kendall G., Woodward J. R. The scalability of evolved on line bin packing heuristics. 2007 IEEE Congress on Evolutionary Computation. 2530–2537 (2007).

[15] Burke E. K., Hyde M. R., Kendall G. Evolving bin packing heuristics with genetic programming. Parallel Problem Solving from Nature – PPSN IX. 860–869 (2006).

[16] Dimopoulos C., Zalzala A. M. S. Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software. **32** (6), 489–498 (2001).

[17] Fukunaga A. Automated discovery of composite SAT variable selection heuristics. Proceedings of the National Conference on Artificial Intelligence (AAAI). 641–648 (2002).

[18] Fukunaga A. S. Automated discovery of local search heuristics for satisfiability testing. Evol. Comput. **16** (1), 31–61 (2008).

[19] Fukunaga A. S. Evolving local search heuristics for SAT using genetic programming. Genetic and Evolutionary Computation – GECCO-2004. 483–494 (2004).

[20] Geiger C. D., Uzsoy R., Aytug H. Rapid modeling and dis- covery of priority dispatching rules: an autonomous learning approach. Journal of Scheduling. **9**, 7–34 (2006).

[21] Keller R. E., Poli R. Cost-benefit investigation of a genetic-programming hyperheuristic. EA 2007: Artificial Evolution. 13–24 (2007).

[22] Keller R. E., Poli R. Linear genetic programming of parsimonious metaheuristics. 2007 IEEE Congress on Evolutionary Computation. 4508–4515 (2007).

[23] Acevedo N., Barra C. R., Bolton C. C., Parada V. Automatic design of specialized algorithms for the binary knapsack problem. Expert Systems with Applications. **141**, 112908 (2020).

[24] Eglese R. W. Simulated annealing: A tool for operational research. European Journal of Operational Research. **46** (3), 271–281 (1990).

[25] Fleischer M. A. Simulated annealing: Past, present, and future. Proceedings of the 1995 Winter Simulation Conference. 155–161 (1995).

[26] Koulamas C., Antony S. R., Jaen R. A survey of simulated annealing applications to operations research problems. Omega. **22** (1), 41–56 (1994).

[27] Kirkpatrick S., Gelatt Jr, C. D., Vecchi M. P. Optimization by Simulated Annealing. Science. **220** (4598), 671–680 (1983).

[28] Romeo F., Sangiovanni-Vincentelli A. A theoretical frame-work for simulated annealing. Algorithmica. **6**, Article number: 302 (1991).

[29] Suman B., Kumar P. A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the Operational Research Society. **57** (10), 1143–1160 (2006).

[30] Alahyane M., Hakim A., Laghrib A., Raghay S. A lattice Boltzmann method applied to the fluid image registration. Applied Mathematics and Computation. **349**, 421–438 (2019).

Гіпер-евристика модифікованої функції вибору за функцією Больцмана

Меллулі О., Гафіді І., Метране А.

LIPIM, ENSA Хурібга, Університет Султана Мулая Слімана, Bd Béni Amir, B.P. 77, Хурібга, Марокко

Гіпер-евристика — це підклас методів дослідження високого рівня, які функціонують у просторі евристичних досліджень низького рівня. Їхня мета — покращити рівень загальності для розв'язування задач комбінаторної оптимізації за допомогою двох основних компонентів: методології евристичного вибору та критерію прийнятності ходу для забезпечення інтенсифікації та диверсифікації [1]. Таким чином, замість того, щоб безпосередньо працювати над розв'язками задачі та обирати один з них, щоб перейти до наступного кроку на кожному етапі, гіпер-евристика діє у просторі евристичного дослідження низького рівня. Функція вибору є однією з гіпер-евристик, які довели свою ефективність у розв'язанні задач комбінаторної оптимізації [2–4]. На кожній ітерації вибір евристики залежить від оцінки, обчисленої шляхом поєднання трьох різних показників, щоб гарантувати як інтенсифікацію, так і диверсифікацію процесу вибору евристики. Тому для розв'язуванн задачі вибирається евристика з найвищим балом. Отже, ключем до успіху в виборі функції є вибір правильних вагових параметрів для трьох її мір. У цій роботі виконано сучасне гіперевристичне дослідження та запропоновано новий метод, який автоматично керує цими ваговими параметрами на основі функції Больцмана. Проведено порівняння результатів, отриманих внаслідок його застосування до п'яти предметних областей, з результатами методу стандартної модифікованої функції вибору, які запропоновані Дрейком та ін. [2, 3].

Ключові слова: *гіпер-евристика, комбінаторна оптимізація, функція вибору, модифікована функція вибору, евристичний вибір, евристична генерація, функція Больцмана.*