A Composite Function for Understanding Bin-Packing Problem and Tabu Search: Towards Self-adaptive Algorithms

V. Landero[1(✉)], David Ríos[1], O. Joaquín Pérez[2], and Carlos Andrés Collazos-Morales[3]

[1] Universidad Politécnica de Apodaca, Ciudad Apodaca, Nuevo León, México
{vlandero,drios}@upapnl.edu.mx
[2] Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), Departamento de Ciencias Computacionales, AP 5-164, 62490 Cuernavaca, México
[3] Universidad Manuela Beltrán, Bogotá, Colombia

Abstract. Different research problems (optimization, classification, ordering) have shown that some problem instances are better solved by a certain solution algorithm in comparison to any other. A literature review indicated implicitly that this phenomenon has been identified, formulated, and analyzed in understanding levels descriptive and predictive without obtaining a deep understanding. In this paper a formulation of phenomenon as problem in the explanatory understanding level and a composite function to solve it are proposed. Case studies for Tabu Search and One Dimension Bin Packing were conducted over set P. Features that describe problem instance (structure, space) and algorithm behavior (searching, operative) were proposed. Three algorithm logical areas were analyzed. Knowledge acquired by the composite function allowed designing of self-adaptive algorithms, which adapt the algorithm logic according to the problem instance description in execution time. The new, self-adaptive algorithms have a statistically significant advantage to the original algorithm in an average 91% of problem instances; other results (set P') indicate that when they obtain a best solution quality, it is significant and when they obtain the same or less solution quality, they finish significantly faster than original algorithm. The composite function can be a viable methodology toward the search of theories that permit the design of self-adaptive algorithms, solving real problems optimally.

1 Introduction

It has been seen for sorting problems, depending on the length and order of the sequence, there are algorithms that perform better than the rest [1]. Also, for NP-hard combinatorial optimization problems, the deterministic algorithms are considered adequate for smaller instances of such problems [2]. Intuition says that the difficulty of a problem instance varies with its size: large instances are usually more difficult to solve than smaller ones. However, in practice, recognizing the measuring difficulty only in terms of the instance size implies overlooking any structural property or feature of the instance, which could

O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 592–608, 2021.
https://doi.org/10.1007/978-3-030-86653-2_43

affect the problem complexity [3] and the algorithm performance. For example, classification problems show that some learning algorithms perform very well on certain problem instances depending on a set of specific features [4]. The scientific community of different disciplines, such as combinatorial optimization, machine learning, artificial intelligence and other fields of knowledge has worked for describing and analyzing the experimental relation between problem-algorithm, with the objective of solving a real problem optimally. However, its analysis has been conducted, in most cases, in descriptive and predictive understanding levels (for a major compression of this concept [5]) and it is necessary deepening more in this objective, in the explanatory understanding level, answering why the relation of certain problem instances and an algorithm produces best solutions (algorithm is very good) and why it is not good with other problem instances. Under the scope of this research (combinatorial optimization area and search algorithms), this paper contains: previous questions and formulations of reviewed literature, so too, the phenomenon is formulated as a question and a formal problem statement in the explanatory level, using and supplementing the Rice's formal nomenclature (Sect. 2); a proposed composite function to solve the stated problem (Sect. 3); a framework for the performance of the proposed composite function over the One Dimension Bin-Packing problem and Tabu Search algorithm (Sect. 4); Development of the proposed composite function, using a instances set $\boldsymbol{P}$ (Sect. 5), the discovered knowledge is used for answering how and why certain problem instances (describing structure and problem solutions space), algorithms features (describing operative and searching behavior); and the algorithm logical design contribute toward a better relation between problem-algorithm (in majority cases of reviewed literature, not all information are taken into account at the same time, problem, algorithm, logical area). A new self-adaptive search algorithm is designed for each case of study. Section 6 describes the results of self-adaptive search algorithms over instances sets $\boldsymbol{P}$ and $\boldsymbol{P}$' ($\boldsymbol{P}' \neq \boldsymbol{P}$)). Conclusions and future works are drawn in Sect. 7.

2 Reviewing State of Art and Setting the Problem Statement

Using and supplementing the Rice's formal Nomenclature,

$\boldsymbol{P}$= $\{x_1, x_2,\ldots, x_m\}$ a set of problem instances or space for analysis.

$\boldsymbol{F}$= the problem features space generated by a description process applied to $\boldsymbol{P}$.

$\boldsymbol{A}$= $\{a_1, a_2,\ldots, a_n\}$ a set of algorithms.

$\boldsymbol{Y}$= the performance space, it represents the mapping of each algorithm to a set to a set of performance metrics.

$\boldsymbol{C}$= $\{C_1, C_2,\ldots,C_n\}$ a partition of $\boldsymbol{P}$, where $|\boldsymbol{A}|=|\boldsymbol{C}|$.

$\boldsymbol{W} = \{(a_q \in \boldsymbol{A}, C_q \in \boldsymbol{C}) \big| Y_{aq,x} > Y_{\alpha,x} \cdot \forall \cdot \alpha \in (A - \{a_q\}), \forall \cdot x \in C_q\}$ is a set of ordered pairs (a_q, C_q), where each dominant algorithm $a_q \in \boldsymbol{A}$ is associated with one element C_q of partition $\boldsymbol{C}$, because this gives the best solution to partition C_q, considering a set of performance metrics mapped in set $\boldsymbol{Y}$.

$\boldsymbol{L}$= the algorithm features space.

In descriptive traditional level, the next first research question arises from experimental relation between problem-algorithm:

1. What is the performance of algorithm $a_q \in \boldsymbol{A}$ to solve the problem $\boldsymbol{P}$?

For this, a set of algorithms $\boldsymbol{A}$ is run over a set of instances $\boldsymbol{P}$. The general performance of each algorithm is measured by some performance metric $y \in \boldsymbol{Y}$ in order to obtain a quantifiable value that could be used for performing a comparison of algorithms by means statistical analysis (statistical tests The Sign, Wilcoxon and Friedman tests, among others) or tabular analysis or graphical analysis; after that performing a results interpretation. One classic example of related work in this understanding level is [6]. Nevertheless, the results of the solution algorithms on an instance set of a problem could be incorrectly interpreted, this is, one might expect that there are pairs of search algorithms a_q and α such that a_q performs better than α on average, even if outperforms a_q at times. Such expectation could be incorrect. Wolpert describes two No Free Lunch (NFL) theorems in [7]. In general terms, it establishes that for any algorithm, a high performance over a set of problems is paid in performance over another set. The existence of instances subsets for specific problem and an algorithm for each subset is suggested by NFL theorems. For the purpose of exemplification in the reviewed literature, other classic examples of related works identified different performances of algorithms on different problem instance sets, based on the construction of an association table, where each set had one or several similar features in the context of the problem structure description [8, 9]. A few of other related works are [1, 2, 4, 10, 11]. The above indicates that one algorithm can be associated to a problem instances subset, where it is the one that solves these instances in the best way possible, for a specific problem domain. The phenomenon observed could be described by means of some set $\boldsymbol{W}$ that includes pairs in the form (a_q, C_q), where instances subset C_q better correspond to a_q for instances set $\boldsymbol{P}$ for a specific problem (see nomenclature for a major description).

In predictive level, a general research question arises:

2. What is the best way to learn W (pairs (a_q, C_q)) in order to predict algorithm a_q that will give the best solution for a new instance from a problem?

The above question needs to consider information from the experimental relation between problem-algorithm, which is significant and has a predictive value. If this question is answered, the solution to the algorithm selection problem can be found. The algorithm selection problem (ASP) is originally formulated in [12], which is stated as:

> For a given problem instance $x \in \boldsymbol{P}$, with features $f(x) \in \boldsymbol{F}$, find the selection mapping $\boldsymbol{S}(f(x))$ into algorithm space $\boldsymbol{A}$, such that the selected algorithm $\propto \in \boldsymbol{A}$ maximizes the performance mapping $y(\propto(x)) \in \boldsymbol{Y}$.

It can be said that the phenomenon mentioned in this paper, can be analyzed and learned in the predictive understanding level of an implicitly manner when problem ASP is being solved. ASP was generalized through different research disciplines in [13], where its solution is important. Two known approaches that are utilized by related works in solving problem ASP are algorithm portfolio and supervised learning. In the case of algorithm portfolio, the algorithm performance is characterized and adjusted to a model (model-based portfolio) either by a regression model [14] or a probability distribution model [15, 16]; other related works have also shown that some problem features are considered in the building of a model (feature-based portfolio), for example, applying

supervised learning [17, 18]. In this case, many related works exhibit learning patterns (corresponding to set W in the context of this paper) from data by means of a supervised algorithm and use them for predicting the best algorithm for an unseen problem instance. Examples: Case-base reasoning [19, 20], decision trees [21, 22], Neural Networks [23] and Random Forest [24–26]. A discussion of machine learning methods applied to ASP problem can be found in [27]. Nevertheless, the principal disadvantage is that the phenomenon identified in the predictive understanding level is analyzed and studied without being fully understood. Two principal reasons: the information considered in analyses is only derived from a problem, or an algorithm, or a logical design (initializing parameter); and the built models are found to be difficult in interpreting the acquired knowledge by nature of its own structure; these are used for predictions.

In the explanatory understanding level, there are different guidelines. For example, some related works focused their analysis on problem difficulty, considering the problem structure, significant features or parameters, identifying important values from them to determine when problem is difficult and when is easy (known as Transition Phase Analysis) through the use of graphical or statistical analysis or unsupervised learning [23–25, 28, 29]. Other related works focused on algorithm performance, some deeply on the searching behavior, considering metrics that measures the trajectory, identifying when it is flat or rugged (there are fluctuations), determining whether the problem is difficult or easy for algorithm (Known as Landscape Analysis) [30–32]; others deeply focused on algorithm logical design [33, 34]; some focused on both [30, 35], for example, obtain important explanations which permit to configure the algorithm in a way that it can produce the better results. However, these explanations are not very clear, in the regard which kind of problem instances will produce better results. What are the specific features of the problem structure that help the configured algorithm better adjust to these problem instances? In [36], this information is considered important in order to adapt the algorithm logical design to the problem structure. A few related works focused on problem and algorithm [37–40], developing some visual tool or performing a graphical, or statistical, or data exploratory, or causal analysis. The reviewing of this literature indicates much effort has been taken to characterize and analyze the experimental relation of problem-algorithm under different understanding levels. However, we believe that it is necessary to pave the way in the comprehension of explanatory understanding level with a starting point, something very essential, simple, and important. Therefore, considering past efforts of the reviewed literature, the experience for previous works [41, 42], and continuing with observed in descriptive and predictive understanding levels, a simple research question arises for a domain specific:

3. Why does a problem instance subset C_q correspond better to an algorithm a_q than other instances in a specific problem domain?

In order to formulate this question as a formal problem, should be considered: as first instance, all significant information from problem (structure, solutions space) and algorithm (operative and searching behaviour), during and after execution, limitations of explanatory and predictive levels; as second instance, a methodology as guide to help obtain a formal model that can discover latent knowledge and explain the phenomenon in question for a specific problem. Following a step beyond to algorithm selection problem

(ASP), considering the above, and continuing in the improving of previous works, the phenomenon could be formulated as the next statement.

> For a set of algorithms $\boldsymbol{A}$ applied to a set of problem instances $\boldsymbol{P}$, with problem features $\boldsymbol{F}$, algorithm features $\boldsymbol{L}$, the algorithms performance space $\boldsymbol{Y}$, the algorithms performance partitions $\boldsymbol{W}$, according to $\boldsymbol{Y}$ and an ordered pair $(a_q, C_q) \in \boldsymbol{W}$; find the composite function $\boldsymbol{E}(a_q, P, \boldsymbol{A}, \boldsymbol{F}, \boldsymbol{L}, \boldsymbol{Y})$ that discovers an explanation formal model $\boldsymbol{M}$, such that $\boldsymbol{M}$, represents the latent knowledge from relations between features that describe: the problem features $\boldsymbol{F}$; interest algorithm $\boldsymbol{L}$; and provides solid foundations to explain, why certain problem instances, being the partition C_q correspond better to interest algorithm a_q, according to performance space $\boldsymbol{Y}$, and why other partitions $(C_q)^c$ do not correspond to algorithm a_q.

3 Proposed Solution

The solution to the above problem statement, is to discover a formal model that can acquire latent knowledge, structured in some way as cause-effect relations from problem, algorithm features, which can help explain such formulation. The process is known as the discovery of causal structure of data [43] (causal model). A causal model can be defined as a causal Bayesian network [43]. It is described by expression 1.

$$M = (V, G, Z) \tag{1}$$

Specifically,

- $V = \{v_1, v_2, .., v_n\}$ is a set of observed features.
- G is a directed acyclic graph with nodes corresponding to the elements of V that represents a causal structure (V, EC); i.e.,
  $EC = \{EC_1, EC_2, \ldots, EC_n\}$, where each $EC_i \in EC$ is a set of ordered pairs,
  $EC_i = \{(v_i, y_1), (v_i, y_2), \ldots, (v_i, y_n)\}$,it is
  $EC_i = \{(v_i \in V, y_k \in V) | v_i \neq y_k, y_k$ is a direct cause of v_i relative to V and there is a directed edge from y_k to v_i in $G\}$
  $Pa(v_i) = \{y_k \in V \mid (v_i, y_k) \in EC_i\}$ is a set of all direct causes of v_i.
- $Z = \mathrm{P}(v_i = j \mid y_1 = \alpha, y_2 = \beta, \ldots, y_p = \gamma)$, is a function of conditional probability of v_i in the range of values j given the direct causes of v_i $\{y_1, y_2, \ldots, y_p\} \in Pa(v_i)$, which are in the ranges of values $y_1 = \alpha, y_2 = \beta, \ldots, y_p = \gamma$.

3.1 Composite Function *E*

In general terms, the composite function $\boldsymbol{E}$ (see Fig. 1) consists of analyzing the experimental relation between the problem (instances set $\boldsymbol{P}$) – algorithm (interest algorithm a_q, $a_q \in \boldsymbol{A}$) considering the space of features from problem $\boldsymbol{F}$, the space of features $\boldsymbol{L}$ and performance $\boldsymbol{Y}$ from algorithms during execution, for discovering latent knowledge about this relation (represented by explanation model $\boldsymbol{M}$).

The domain of proposed composite function $\boldsymbol{E}$ (Expression 2), are parameters: the interest algorithm a_q, a set of algorithms $\boldsymbol{A}$, applied to a set of problem instances $\boldsymbol{P}$, the

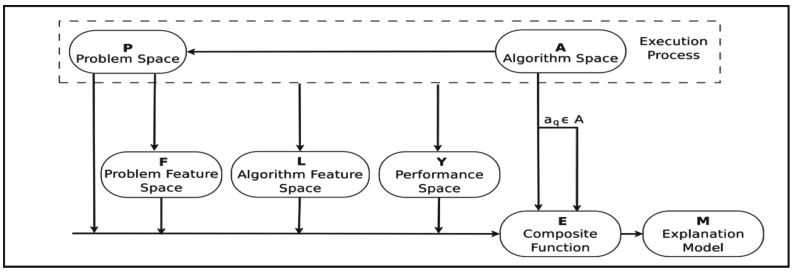

Fig. 1. General diagram of the composite function

set $\boldsymbol{F}$ obtained by f ($\boldsymbol{P}$), the set $\boldsymbol{L}$ obtained by $l(\boldsymbol{A})$, the performance space $\boldsymbol{Y}$, obtained by $y(\boldsymbol{A})$. The functions f and l perform a description process, before and during execution of algorithms, to obtain features that represent information about set $\boldsymbol{P}$ and $\boldsymbol{A}$.

$$(\boldsymbol{E} \cdot fz \cdot fg \cdot fv \cdot fd \cdot fc)(a_q, P, A, F, L, Y) \equiv E(fz(fg(fv(fd(fc(a_q, P, A, F, L, Y))))))) \tag{2}$$

The function y evaluates the algorithms $\boldsymbol{A}$, by means a set of performance metrics. The acquisition of latent knowledge from experimental relation between problem-algorithm is formalized by means of the proposed composite function $\boldsymbol{E}$, which, evaluated in each iteration, obtains the values of a set of significant features, causal relations between these, and estimations, represented by the sets V, G, Z (codomain-causal model $\boldsymbol{M}$).

4 Framework for Proposed Composite Function *E*

The instance sets $\boldsymbol{P}$ and $\boldsymbol{P'}$ ($\boldsymbol{P} \neq \boldsymbol{P'}$) were randomly selected (324) from Beasley's OR-Library [44], the Operational Research Library [45]. The objective of each case of study is to explain the description process and the composite function of the latent relation between problem (One Dimension Bin-Packing - BPP) and algorithm (Tabu Search); an insight into the problem structure, problem solution space, the algorithm logical design, its operative behaviour, and behaviour during its searching and performance. Due to the above, four versions of Tabu Search algorithm were implemented, where each one had a specific logical design (Table 1). The methodology for initializing control parameter (PM) is applied to size of Tabu list (*nLTabu*). The static procedure is to set it as 7 [46]. The dynamic procedure is to set it as $\sqrt{n}$, where n represents the number of the objects or items of the problem instance. The methodology for the generation of an initial solution (IM) can be conducted by a random or deterministic procedure. The methodology for building the neighbourhood of a solution (NM) can achieved through one or several methods, which were proposed in [47]. The candidate list (LCANDI) size was fixed to 4*(*nLTabu* * 0.25) for all the study cases. As well as, the methodology to stop the algorithm execution (SM) was the same; after 4000 iterations or there was no improvement in the solution. Table 2 shows the study cases.

Table 1. Set of algorithms $\boldsymbol{A}$

Variants	PM		IM		NM	
	Static	Dynamic	Random	Deterministic	One	Several
a_1	✓		✓		✓	
a_2	✓		✓			✓
a_3		✓	✓		✓	
a_4	✓			✓	✓	

Table 2. Cases of study

Case of study	Algorithms	Methodology
1	a_1, a_2	Building Neighborhood (NM)
2	a_1, a_3	Initializing Control Parameter (IM)
3	a_1, a_4	Generating Initial Solution (PM)

5 Performing Composite Function E

The function $f(\boldsymbol{P})$ performs description process for the instances set $\boldsymbol{P}$, where the set $\boldsymbol{F}$ is obtained. After that, the set of algorithms $\boldsymbol{A}(a_1, a_2, a_3, a_4)$ is applied to solve the problem instances set $\boldsymbol{P}$. During the search and solution process, the functions $l(\boldsymbol{A})$ and $y(\boldsymbol{A})$ perform description process to obtain the algorithm features space, set $\boldsymbol{L}$ and set performance space $\boldsymbol{Y}$. This process ($f(\boldsymbol{P})$, $l(\boldsymbol{A})$, $y(\boldsymbol{A})$), for all cases of study, is described in greater detail only for those features that were significant in next sections.

5.1 Problem Instances Structure Description: Function $f(\boldsymbol{P})$

There are three features, (b, d) [21] and cu proposed in this paper; b describes the proportion of the total weights of the objects that can be assigned to one container; d describes the dispersion of the quotient between the object weight and the container capacity; cu is the kurtosis of object weights (w weights and de standard deviation).

$$cu = \frac{\sum_{i=1}^{n} (w_i - \overline{w})^2}{de^4} \tag{3}$$

The problem solution space for each instance, os, is described in past works [38, 41, 42]. It is the variability of ms randomly generated solutions ($ms = 100$ produced better results). The codomain of function f is the set $\boldsymbol{F}$ (expression 4), where rows represent the problem instances and columns are the values of these features.

$$F = \{\{b_1, d_1, cu_1, os_1\}, \{b_2, d_2, cu_2, os_2\}, \ldots, \{b_m, d_m, cu_m, os_m\}\} \tag{4}$$

5.2 Algorithm Behavior Description: Function $l(A)$

The algorithm operative behaviour is described by features (*nn*, *fn, vf*), proposed in past works [38, 41, 42]. The number of neighbours built by algorithm during its search process per instance is given by *nn*. The number and variability of feasible solutions are given by *fn*, *vf*. The algorithm searching behaviour is described by features *pn* (number of inflection points), *vn* (number of valleys), *vs* (size of valleys) proposed in past works [38, 41, 42] and *vd* proposed in this paper. These features are obtained from the algorithm searching path. The searching inflections are the changes in the direction of fitness function from two consecutive solutions during one algorithm run; *pn* is the average of these for all algorithm runs (16); *vn* is concerned if there exists a searching pattern that refers to our concept, Valley. It is considered when there is a sequence major to *sm* solutions, where their fitness function values keep on decreasing ($sm = 6$ indicated be significant in past works). Then, the feature *vn* is the average of Valleys identified from algorithm runs. The inflection point located in an identified Valley is considered as the location point, for example one run has location points p_1, p_2, p_3 and p_4; the distance between each point is calculated, dd_1, dd_2 and dd_3. The standard deviation of these is calculated (expression 5). The average of Valleys dispersion for all algorithm runs is calculated, *vd*. The set $\boldsymbol{L}$ is built with the specific order as Expression, 6. Here, $L_{1,1}$ means algorithm a_1 for problem instance x_1 has the elements, nn_{11}, fn_{11}, vf_{11}, pn_{11}, vn_{11}, $vs_{11,}$ vd_{11} (algorithm behaviour features) and so on.

$$vd_{run} = \sqrt{\frac{\sum_{i=1}^{pn-1}\left(dd_i - \overline{dd}\right)^2}{pn-2}} \tag{5}$$

$$L = \left\{ \begin{matrix} \{nn_{11}, vf_{11}, pn_{11}, vn_{11}, vs_{11}, vd_{11}\}, \ldots, \{nn_{1m}, \ldots, vd_{1m}\} \\ \{nn_{21}, vf_{21}, pn_{21}, vn_{21}, vs_{21}, vd_{21}\}, \ldots, \{nn_{2m}, \ldots, vd_{2m}\} \\ \{nn_{n1}, vf_{n1}, pn_{n1}, vn_{n1}, vs_{n1}, vd_{n1}\}, \ldots, \{nn_{nm}, \ldots, vd_{nm}\} \end{matrix} \right\} \tag{6}$$

5.3 Performance Space Description: Function $Y(A)$

The function, *y*, evaluates the algorithm performance according to metrics *time* and *quality*. The metric *time* is the total of feasible and infeasible solutions built during algorithm execution. The metric *quality* is the ratio between found solution and theoretical solution [41, 42]. The codomain of the function, *y*, is the set, *Y* (performance space) which is built with the specific order as Expression 7. Here, $Y_{1,1}$ means algorithm a_1 for problem instance x_1 has the elements, $quality_{11}$ and $time_{11}$, $Y_{1,m}$ means algorithm a_1 for problem instance x_m has the elements *quality*; *time*, and so on.

$$Y = \left\{ \begin{matrix} \{quality_{11}, time_{11}\}, \ldots, \{quality_{1m}, time_{1m}\} \\ \{quality_{21}, time_{21}\}, \ldots, \{quality_{2m}, time_{2m}\} \\ \{quality_{n1}, time_{n1}\}, \ldots, \{quality_{nm}, time_{nm}\} \end{matrix} \right\} \tag{7}$$

5.4 Discovering Knowledge: Functions *fc*, *fd* (*TC*), *fv*(*D*), *fz*(*G*)

The proposed composite function, $\boldsymbol{E}$, is applied to one algorithm of interest a_q for each case of study (1, 2, 3). The algorithms, a_2, a_3, a_1 were randomly selected. The function, *fc* identifies the set $\boldsymbol{W}$, considering performance space $\boldsymbol{Y}$ (based on *time* and *quality* metrics). After that, the performance scope of interest algorithm a_q is obtained, considering set $\boldsymbol{W}$. The codomain of function *fc* is described by set $\boldsymbol{S}$. Each value indicates the scope of interest algorithm for each problem instance (set $\boldsymbol{P}$), 1 was the best, 0 otherwise. For example, $\boldsymbol{S} = \{0, 1,\ldots,1\}$ means that interest algorithm had scope: 0 for instance 1, 1 for instance 2, 1 for instance *m* and so on. A sets family $SF = \{\boldsymbol{F}, \boldsymbol{L}, \boldsymbol{Y}, \boldsymbol{S}\}$ is built for interest algorithm a_q and is represented by dataset *TC*, where $TC = \cup SF$; the tuples are instances and columns are features that describe: the structure and problem solutions space $\boldsymbol{F} = f(\boldsymbol{P})$; the operative and searching behaviours of interest algorithm, obtained from set $\boldsymbol{L}$; performance space of the interest algorithm, obtained from set $\boldsymbol{Y}$ (*time* and *quality*); performance scope from set *W*, value from set *S*. The function, *fd*, first normalizes the values of each by means of method min-max; values that lie within the closed interval, [0, 1]. After that, the method, MDL [48], is performed to discretize the values. The codomain of function *fd* is the discretized dataset, *D*. The function, *fv*, performs general, graphical and variance analyses of the features from dataset *D* for selecting the most significant. The general analysis creates bar plots, where the frequencies of the values of the features are analysed with respect to the scope (value *s*) of *aq*. The metric *quality* from dataset *TC* did not assume a normal distribution; therefore, for the graphical and variance analyses, it was transformed using methods of logarithm or Box-Cox, using values 2 or −2 for λ. The graphical analysis creates box plots for each feature (identified in general analysis) with respect to metric *quality*, in order to identify features that in influence it in terms of variation and locality. Finally, the function, *fv*, performs an analysis one variance (ANOVA), with a confidence level of 95% for each feature (identified in graphical analysis) with respect to the *quality* metric. The function, *fv*, did not find significant features in the study case 3; for other cases of study, 1 and 2, the codomain of function *fv* is the significant dataset $\boldsymbol{V}_1$ with proposed features. For example, in case study 1, the dataset, $\boldsymbol{V}_1$, is formed by features *b*, *os*, *nn*, *vf*, *pn*, *vn*, *vs*, and scope of interest algorithm a_2 (*S*). So too, with the objective of considering metrics known and used by the scientific community (describing the algorithm searching behaviour), the auto-correlation coefficient, (*ca*), the auto-correlation length, (*al*) ([49]), and highlight the utility of our proposed features (*pn*, *vn* and *vs*), another dataset, $\boldsymbol{V}_2$, is built with features *b*, *os*, *nn*, *vf*, metrics *ac*, *al*, and scope of interest algorithm a_2 (*S*). The datasets $\boldsymbol{V}_1$ (by means function *fv*) and $\boldsymbol{V}_2$ are built in case of studies 1 and 2. The function *fg* performs the process of learning a causal structure (algorithm PC [43]) with a confidence level of 95% for datasets $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ in case studies; the causal inference software, HUGIN (Hugin Expert, www.hugin.com) was used.

Figure 2 shows the causal structures: *a*) $fg \rightarrow G_1$ and *b*) $fg \rightarrow G_2$ from datasets $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ for these cases of study. It is important to emphasize that the causal structure, G_1, in these cases represents clearly the direct causes (problem and algorithm significant features) of performance scope for interest algorithm a_q in performance space $\boldsymbol{Y}$, in terms of set $\boldsymbol{W}$. For case study 1, it is evident that the causal structure, G_2, did not yield relevant information about direct causes. In case study 2, G_2 did not yield relevant information

about direct causes for algorithm performance scope with respect to algorithm behavior during its searching. These structures (G_1) were considered for the next analyses. Also, Fig. 2 shows the intervals of direct causes, obtained previously by function *fd*. Continuing with the composite function, ***E***, the function, *fz*, referring to parameter learning algorithm Counting [43], estimates the intensity of causal relations (identified in structures G_1), see Table 3, the codomain is set *Z*. The codomain of composite function ***E*** for each study case are the sets V_1, G_1 and *Z* (causal model ***M***). The problem instance set, ***P'***, was used as an input for each causal structure G_1 to obtain the prediction accuracy percentage, using another causal inference software NETICA (Norsys Corporation), where the obtained percentages were %78.04 and %70.37, respectively.

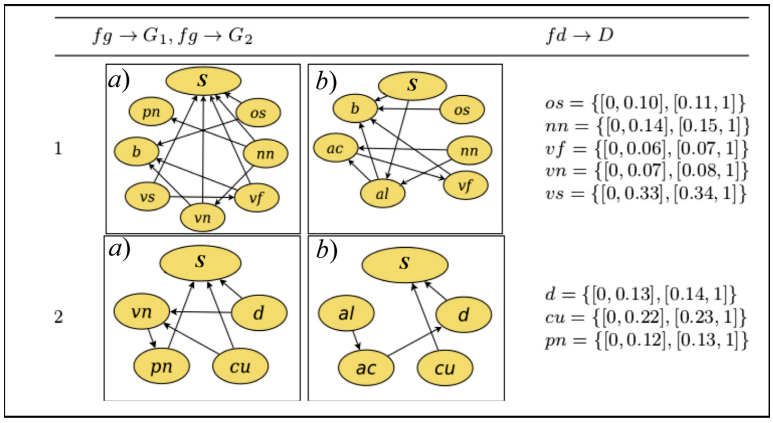

Fig. 2. Discovering knowledge

Table 3. Significant features, casual relations and estimations

	Functions	
1	$fv \rightarrow \boldsymbol{V_1}$	V_1={*b*, *os*, *nn*, *vf*, *vn*, *vs*, *pn*, *S*}
	$fg \rightarrow \boldsymbol{G_1}$	$EC = \left\{ \begin{matrix} \{\{S, os\}, \{S, nn\}, \{S, vf\}, \{S, vn\}, \{vs\}\}, \\ \{\{b, os\}, \{b, fv\}, \{b, vn\}\}, \{(pn, nn)\}, \\ \{(vf, vs)\}, \{(vn, nn)\} \end{matrix} \right\}$
	$fz \rightarrow \boldsymbol{Z}$	$P(S = 1 \mid os = 2, nn = 2, vf = 2, vn = 2, vs = 2) = 92\%$ $P(S = 0 \mid os = 1, nn = 1, vf = 1, vn = 1, vs = 1) = 99\%$
2	$fv \rightarrow \boldsymbol{V_1}$	V_1={*d*, *cu*, *pn*, *vn*, *S*}
	$fg \rightarrow \boldsymbol{G_1}$	$EC = \left\{ \begin{matrix} \{(S, d), (S, cu), (S, pn)\}, \{(vn, d), (vn, cu)\}, \\ \{(pn, vn)\} \end{matrix} \right\}$
	$fz \rightarrow \boldsymbol{Z}$	$P(S = 1 \mid d = 2, cu = 1, pn = 2) = 99\%$ $P(S = 0 \mid d = 1, cu = 2, pn = 1) = 76\%$

5.5 Analyzing Acquired Knowledge and Self-adapting Algorithms

The results from case studies 1 and 2 are reviewed deeply in Fig. 3 (Analysis 1 and 2). So too, the logical design of the new Tabu Search Self-Adapting Algorithms is shown. On the other hand, another interesting result was obtained in case study 3, where the Tabu Search algorithm was distinguished by methodology in order to generate the initial solution. Though the function, *fv*, of composite function ***E*** could not discover significant features to build a causal explanation model, it is important to highlight the fact that a knowledge was obtained as well. One possible interpretation of this result may be that the method used to generate the initial solution does not impact the algorithm performance, according to set ***Y***, in solving problem instances. One similar result was observed in [31] for another optimization problem and Tabu Search algorithm.

Tabu Search Self-adaptive Algorithms (a_{g1} and a_{g2})

```
   Begin
1  Calculates feature os from input parameters;                          a_g1
2  nLTabu = 7 // Methodology for initializing input control parameter (PM)
3  Calculates d and cu from problem parameters;                          a_g2
4  // Methodology for initializing input control parameter (PM)
5  If (value of d belongs to interval 2 and value of cu belongs to interval 1)
6  Then nLTabu = √n; Else nLTabu = 7; // Size of Tabu list
7  tn = nLTabu; //Tenency of one solution in Tabu list
8  x*=Generate randomly a feasible solution initial; x=x*; LCANDI={∅};
9  LTabu={∅};
10 Repeat
11   Begin                                                               a_g1
12   // Methodology for building neighborhood (NM)
13   If (value of os belongs to interval 1) Then
14     LCANDI = N(x, LTabu); // Building with one method
15   Else LCANDI = N(x, LTabu); // Building with several methods
16   LCANDI = N(x, LTabu); // Building with one method                   a_g2
17   y = the best solution of LCANDI and y ∉ LTabu;
18   LTabu = LTabu ∪ {(y, tn)};
19   For each solution e ∈ LTabu do
20     Tenency of e is decremented;
21     If the tenency of e has expired Then LTabu = LTabu − {(e, tn)};
22   End
23   If fo(y) > fo(y*) Then x = y; y*=y;
24   End
25 Until 4000 iterations or there is no improvement
26 return y*;
27 End
```

Analysis 1. The logical design of algorithm a_2 adjusts better ($S = 1$) to the problem instances description (partition C_2) in terms of $os = 2$. Its searching behavior was flexible to find a solution close to theoretical, as it could enter and leave from an identified valleys number $vn = 2$ with a valley average size in the second interval ($vs = 2$). Algorithm a_2 wins in *quality* in the majority of instances (214 from 216) and wins in *time* 2 instances (when quality is the same). The logical design of a_2 (several methods for searching) is responsible for its high cost in its performance, in terms of *time*; it lost in 93 out of 108 problem instances, as it is not necessary to intensify the search a lot for instances with a description in terms of $os = 1$. The logical design of a_2 does not adjust ($S = 0$) to this problem description (partition C_2)[c]. The algorithm, a_1, had an advantage in this situation because of its logical design, one method is considered for the searching. It is limited to finding solutions and corresponds better to this kind of problems. This knowledge gave guidelines to design a new Tabu Search self-adaptive algorithm a_{g1}.

Analysis 2. The logical design of a_3 adjusts better ($S = 1$) to problem instances description (partition C_3) in terms of $d = 2$ and $cu = 1$. This permitted to the logical design of algorithm, a_3, store more Tabu solutions. It is more restrictive in accepting a solution compared to algorithm a_1 (Tabu list size - 7); it has a searching behavior with inflection in the second interval ($pn = 2$), ranging between solutions that cannot be in the Tabu list; and it gives a solution more close to theoretical. It is found that algorithm a_3 wins in *quality* in more of half instances (164 from 278) and wins in *time* 114 instances (when *quality* is the same). However, for instances whose problem instances description (in terms $d = 1$ and $cu = 2$), the solutions that could be generated from these will be not very different; the searching behavior of algorithm a_3 had an inflection ($pn = 1$) in the first interval. Therefore, these solutions can be Tabu because the Tabu list of algorithm a_3 is long; it loses ($S = 0$) in terms of *time* in more of half instances (27 out of 46). Unlike the algorithm, a_1, which has an advantage in this situation, it is not very restrictive with the solutions generated, as it handles a small Tabu list. The acquired knowledge allowed the designing for a new self-adaptive algorithm a_{g2}.

Fig. 3. Analysis of case studies 1, 2 and Tabu Search Self-Adaptive Algorithms

6 Results Analysis

In the reviewed literature, there was no related work with the same circumstances for comparing performance of individual results. An example of this can be an improved Tabu Search algorithm for BPP. Therefore, the performance for the self-adaptive algorithms (a_{a1}, a_{a2}) was compared against that of the original analyzed algorithms (a_2, a_3), it is to say $a_{a1}-a_{2,}$ $a_{a2}-a_{3.}$ The comparative analysis of performance results from the algorithms was performed in two consecutive phases of analysis. The first analysis phase consisted on determining the total scope of new self-adaptive algorithms a_{a1} (287) a_{a2} (301); thereafter determining their total scope percentage, example $287/324 = 89\%$ for a_{a1}, a_{a2} (93%). Analysing the partitions $C_{a_{a1}}$ and $C_{a_{a2}}$ in more detail, a_{a1} wins 159 and a_{a2} wins 163 instances in *quality*, 128 and 138 in *time*, where the *quality* is the same for both algorithms (see Fig. 4).

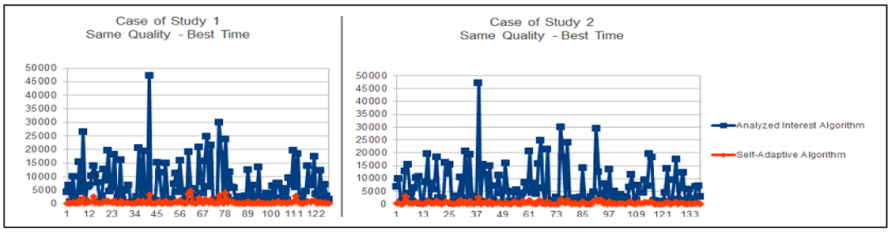

Fig. 4. Times of analyzed algorithm and self-adaptive algorithm (same quality)

The time differences are too big, it is not necessary to apply a statistical test. The self-adaptive algorithms finishing faster than the interest algorithm (a_q) analyzed in study cases. Due the above, the objective of second analysis phase is to verify the values of *quality* metric, specifically when self-adaptive algorithm has the best *quality* (159-case 1, 163-case 2). In this sense, for study cases 1 and 2, a_{a1} had best time in 80 out of 159 problem instances and a_{a2} had best time in 95 out of 163 problem instances.

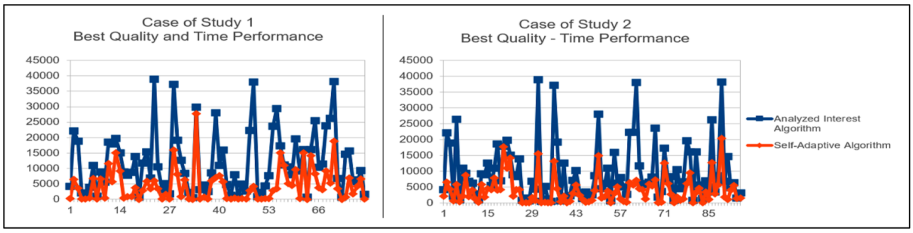

Fig. 5. Times of analyzed algorithm and self-adaptive algorithm (best quality)

Figure 5 shows these times. The metric values *time* for each one of algorithms do not assume a normal distribution. Thus, a nonparametric statistical test of two dependent samples is applied (the two sample two-side wilcoxon signed rank test) for significance

levels 95% and 99%. The Dataplot statistical software (www.itl.nist.gov) was used for this test. The test statistic was 7.67 and 8.46 for study cases. The null hypothesis (equal means) is rejected (critical values 1.96, 2.57) and it means that there is a significant difference between times. The self-adaptive algorithms (μ_2) finishing faster than the original algorithm (μ_1) analyzed in study cases. Continuing with the analysis, it is necessary to verify if there is a statistically significant difference between the means of metric *quality*. The values of this metric do not assume a normal distribution. Thus, the same statistical test for significance levels was applied (test statistic 10.15 and 9.92). The null hypothesis is rejected, there is a significant difference in terms *quality*.

6.1 Other Results

The analysed original algorithm and self-adaptive algorithm from cases of study 1 and 2, were executed over set $\boldsymbol{P}$' ($\boldsymbol{P}$' $\neq \boldsymbol{P}$). The self-adaptive algorithms had better *quality* or *time* in 166 and 156 problem instances of set $\boldsymbol{P}$' than analysed interest algorithm, respectively. Figure 6 shows the *time* of algorithms when they have the same quality. The same wilcoxon statistical test was applied to *time* differences. The test statistic was 9.27 and 8.37 for both cases. The null hypothesis is rejected for significance levels 95% and 99% (critical values 1.96, 2.57). The self-adaptive algorithms finishing faster than the analysed interest algorithm. Also, Fig. 7 shows the times when self-adaptive algorithm has less *quality* than analysed interest algorithm. The *quality* difference average was 0.025 (very small) for 158 problem instances out of 324 and the self-adaptive algorithms finishing faster than the interest algorithm in most cases over set $\boldsymbol{P}$'.

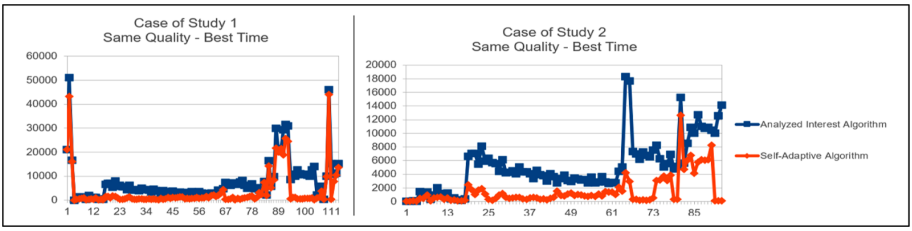

Fig. 6. Same quality of both algorithms and best times of self-adaptive algorithm

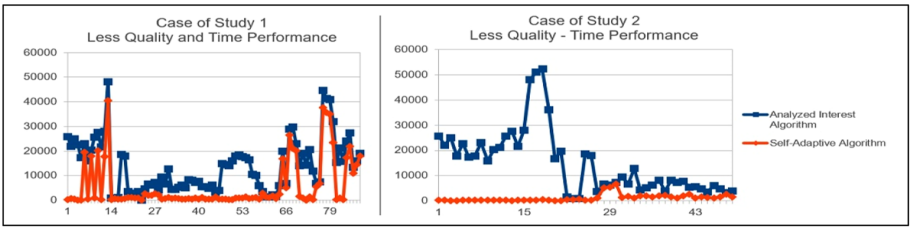

Fig. 7. Less quality and best times of self-adaptive algorithm

7 Conclusions

Most of the literature consulted from combinatorial optimization area has been focusing only on problem information or algorithm information or, rarely, on both in order to describe the experimental relation between problem-algorithm. Furthermore, in the analysis of this relation, has been identified in the descriptive understanding level that certain problem instances correspond better to a certain algorithm than other. This phenomenon has been analyzed and learned implicitly in the predictive understanding level (algorithm selection problem). However, it has not been understood at all for a specific problem. This paper goes beyond the predictive level, covering limitations identified. The phenomenon is formally formulated as question and problem at the explanatory understanding level. The composite function, ***E***, is proposed to solve such formulation and performed for a specific domain (One Dimension Bin-Packing problem and Tabu Search algorithm) over an instance set ***P***. A description process for problem structure and space, for algorithm performance and the operative and searching behaviors of the algorithm during execution (significant features) was proposed to build the domain of the composite function. Also, the metrics known by scientific community were used, but these were not significant in the analyses. Three important logical areas were contemplated. The knowledge acquired by the proposed composite function allowed the solving of the stated problem, understanding the phenomenon, answering the "how" and "why" for certain problem instances, algorithm significant features and the algorithm logical design contribute toward a better relation between problem-algorithm. It was applied to design self-adaptive algorithms and improve the performance considering the causal relations according to problem structure or space. On average, a 91% significant advantage of Tabu Search self-adaptive algorithms was obtained over analyzed original algorithms. Other results over set ***P'*** showed that when they obtain the same or less quality of solution, they finish significantly faster than analyzed interest algorithm and the quality difference is very small. As future work is to explore generalized features. The proposed composite function ***E*** could act as a guideline to find latent knowledge from relation problem-algorithm of other problem domains and search algorithms; this permits the adapting of logical design as well as operative and searching behaviors of an algorithm (self-adaptive algorithm) to problem structure, solutions space and behavior of algorithms (operative and searching) during execution for providing the best solution to problems.

References

1. Lagoudakis, M., Littman, M.: Learning to select branching rules in the DPLL procedure for satisfiability. Electron. Notes Discrete Math. **9**, 344–359 (2001)
2. Chr, P., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity (1982)
3. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-29953-X
4. Rendell, L., Cho, H.: Empirical learning as a function of concept character. Mach. Learn. **5**, 267–298 (1990)
5. Cohen, P.: Empirical Methods for Artificial Intelligence. The MIT Press, Cambridge (1995)

6. Barr, R., Golden, B., Kelly, J., Resende, M.: Designing and reporting on computational experiments with heuristic methods. J. Heuristics **1**(1), 9–32 (1995)
7. Wolpert, D., Macready, W.: No free lunch theorems for optimizations. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1996)
8. Frost, D., Dechter, R.: In search of the best constraint satisfaction search. In: Proceedings of the National Conference on Artificial Intelligence, Seattle, vol. 94, pp. 301–306 (1994)
9. Tsang, E., Borrett, J., Kwan, A. An attempt to map the performance of a range of algorithm and heuristic combinations. In: Hallam, J., et al. (eds.) Hybrid Problems, Hybrid Solutions. Proceedings of AISB-95, vol. 27, pp. 203–216. IOS Press, Amsterdam (1995)
10. Frost, D., Rish, I., Vila, L.: Summarizing CSP hardness with continuous probability distributions. In: Proceedings of the 14th National Conference on AI, American Association for Artificial Intelligence, pp. 327–333 (1997)
11. Vanchipura, R., Sridharan, R.: Development and analysis of constructive heuristic algorithms for flow shop scheduling problems with sequence-dependent setup times. Int. J. Adv. Manufact. Technol. **67**, 1337–1353 (2013)
12. Rice, J.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
13. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1), 1–25 (2009)
14. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 556–572. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3_37
15. Silverthorn, B., Miikkulainen, R.: Latent class models for algorithm portfolio methods. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, Georgia, USA (2010)
16. Yuen, S., Zhang, X.: Multiobjective evolutionary algorithm portfolio: choosing suitable algorithm for multiobjective optimization problem. In: 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, pp. 1967–1973 (2014)
17. Guerri, A., Milano, M.: Learning techniques for automatic algorithm portfolio selection. In: Proceedings of the 16th Biennial European Conference on Artificial Intelligence, Valencia, Spain, pp. 475–479. IOS Press, Burke (2004)
18. Xu, L., Hoos, H., Leyton-Brown, K.: Hydra: automatically configuring algorithms for portfolio-based selection. In: Proceedings of the 25th National Conference on Artificial Intelligence (AAAI 2010), pp. 210–216 (2010)
19. Pavón, R., Díaz, F., Laza, R., Luzón, M.: Experimental evaluation of an automatic parameter setting system. Expert Syst. Appl. **37**, 5224–5238 (2010)
20. Yeguas, E., Luzón, M., Pavón, R., Laza, R., Arroyo, G., Díaz, F.: Automatic parameter tuning for evolutionary algorithms using a Bayesian case-based reasoning system. Appl. Soft Comput. **18**, 185–195 (2014)
21. Pérez, J., Pazos, R.A., Frausto, J., Rodríguez, G., Romero, D., Cruz, L.: A statistical approach for algorithm selection. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 417–431. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24838-5_31
22. Ries, J., Beullens, P.: A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction. J. Oper. Res. Soc. **66**(5), 782–793 (2015)
23. Smith-Miles, K., van Hemert, J., Lim, X.Y.: Understanding TSP difficulty by learning from evolved instances. In: Blum, C., Battiti, R. (eds.) LION 2010. LNCS, vol. 6073, pp. 266–280. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13800-3_29
24. Hutter, F., Xu, L., Hoos, H., Leyton-Brown, K.: Algorithm runtime prediction: methods & evaluation. Artif. Intell. **206**, 79–111 (2014)
25. Leyton-Brown, K., Hoos, H., Hutter, F., Xu, L.: Understanding the empirical hardness of NP-complete problems. Mag. Commun. ACM **57**(5), 98–107 (2014)

26. Munoz, M., Kirley, M., Halgamuge, S.: Exploratory landscape analysis of continuous space optimization problems using information content. IEEE Trans. Evol. Comput. **19**(1), 74–87 (2015)
27. Kottho, L., Gent, I.P., Miguel, I.: An evaluation of machine learning in algorithm selection for search problems. AI Commun. **25**(3), 257–270 (2012)
28. Lopez, T.T., Schaeer, E., Domiguez-Diaz, D., Dominguez-Carrillo, G.: Structural effects in algorithm performance: a framework and a case study on graph coloring. In: Computing Conference, 2017, pp. 101–112. IEEE (2017)
29. Fu, H., Xu, Y., Chen, S., Liu, J.: Improving WalkSAT for random 3-SAT problems. J. Univ. Comput. Sci. **26**(2), 220–243 (2020)
30. Tavares, J.: Multidimensional knapsack problem: a fitness landscape analysis. IEEE Trans. Syst. Man Cybern. Part B: Cynern. **38**(3), 604–616 (2008)
31. Watson, J., Darrell, W., Adele, E.: Linking search space structure, run-time dynamics, and problem difficulty: a step toward demystifying tabu search. J. Artif. Intell. Res. **24**, 221–261 (2005)
32. Watson, J.: An introduction to fitness landscape analysis and cost models for local search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, pp. 599–623. Springer, Boston (2010). https://doi.org/10.1007/978-1-4419-1665-5_20
33. Chevalier, R.: Balancing the effects of parameter settings on a genetic algorithm for multiple fault diagnosis. Artificial Intelligence, University of Georgia (2006)
34. Cayci, A., Menasalvas, E., Saygin, Y., Eibe, S.: Self-configuring data mining for ubiquitous computing. Inf. Sci. **246**, 83–99 (2013)
35. Le, M., Ong, Y., Jin, Y.: Lamarckian memetic algorithms: local optimum and connectivity structure analysis. Memetic Comput. **1**, 175–190 (2009)
36. Montero, E., Riff, M.: On-the-fly calibrating strategies for evolutionary algorithms. Inf. Sci. **181**, 552–566 (2011)
37. Pérez, J., Cruz, L., Landero, V.: Explaining performance of the threshold accepting algorithm for the bin packing problem: a causal approach. Pol. J. Environ. Stud. **16**(5B), 72–76 (2007)
38. Pérez, J., et al.: An application of causality for representing and providing formal explanations about the behavior of the threshold accepting algorithm. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 1087–1098. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69731-2_102
39. Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H.J.F., Alvim, A.C.: A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Comput. Oper. Res. **55**, 52–64 (2015)
40. Taghavi, T., Pimentel, A., Sabeghi, M.: VMODEX: a novel visualization tool for rapid analysis of heuristic-based multi-objective design space exploration of heterogeneous MPSoC architectures. Simul. Model. Pract. Theory **22**, 166–196 (2012)
41. Landero, V., Pérez, J., Cruz, L., Turrubiates, T., Ríos, D.: Effects in the algorithm performance from problem structure, searching behavior and temperature: a causal study case for threshold accepting and bin-packing. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11619, pp. 152–166. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24289-3_13
42. Landero, V., Ríos, D., Pérez, J., Cruz, L., Collazos-Morales, C.: Characterizing and analyzing the relation between bin-packing problem and tabu search algorithm. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12249, pp. 149–164. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58799-4_11
43. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edn. The MIT Press, Cambridge (2001)
44. Beasley, J.E.: OR-Library. Brunel University (2006). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/binpackinfo.html

45. Scholl, A., Klein, R. (2003). http://www.wiwi.uni-jena.de/Entscheidung/binpp/
46. Glover, F.: Tabu search - Part I, first comprehensive description of tabu search. ORSA-J. Comput. **1**(3), 190–206 (1989)
47. Fleszar, K., Hindi, K.S.: New heuristics for one-dimensional bin packing. Comput. Oper. Res. **29**, 821–839 (2002)
48. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: IJCAI, pp. 1022–1029 (1993)
49. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In: New Ideas in Optimization, pp. 245–260. McGraw-Hill Ltd., UK (1999)