IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 2, 2017, accepted April 16, 2017, date of publication April 28, 2017, date of current version June 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2699426

Evolvability Metric Estimation by a Parallel Perceptron for On-Line Selection Hyper-Heuristics

JORGE A. SORIA-ALCARAZ, ANDRÉS ESPINAL, AND MARCO A. SOTELO-FIGUEROA
Departamento de Estudios Organizacionales, División de Ciencias Económico Administrativas, Universidad de Guanajuato, Guanajuato 36000, México
Corresponding author: Jorge A. Soria-Alcaraz (jorge.soria@ugto.mx)

ABSTRACT Online hyper-heuristic selection is a novel and powerful approach to solving complex problems. This approach dynamically selects, based on the state of a given solution, the most promising operator (from a pool of operators) to continue the search process. The dynamic selection is usually based on the analysis of the latest applications of a given operator during actual execution, estimating the potential success of the operator at the current solution state. The estimation can be made by evolvability metrics. Calculating an evolvability metric is computationally expensive since it requires the generation and evaluation of a neighborhood of solutions. This paper aims to estimate the potential success of an operator for a given solution state by using a pre-trained neural network; known as a parallel perceptron. The proposal accelerates the online selection process, allowing us to achieve better performance than hyper-heuristic models, which directly use evolvability functions.

INDEX TERMS Adaptive algorithm, optimization, artificial intelligence, artificial neural networks, parallel perceptron.

I. INTRODUCTION

Hyper-heuristics are high-level strategies that choose or generate a set of low-level heuristics to solve difficult search and optimization problems [7], [24]. Hyper-heuristics aim to replace bespoke approaches by using more general methodologies, thereby reducing the required expertise to construct individual heuristics as their main goal. [6], [9].

Selection hyper-heuristics operate at a high-level to choose the next operator, from a low-level heuristic pool, to guide the search [6], [7], [25], [30]. The objective is to apply the most effective low-level heuristic at each subsequent stage in order to solve a given problem instance. The result is an automated methodology that generates a sequence of heuristics to improve or construct a solution.

These methods need to identify the potential success of each specific heuristic at any step of the search process; this is not an easy task, because the potential of each heuristic may vary dynamically during the search process. Other approaches used when designing optimization heuristics in an autonomous way are automatic parameter tuning and algorithm configuration tools on a set of training instances [4], [13]. Eiben and Smit [10] specifically propose a framework for parameter tuning along with a survey of tunning methods. However, that framework was specialized for tunning parameters and not for operator selection.

This work considers adaptive operator selection (AOS), which requires two cooperating mechanisms: *operator selection* that defines how the next operator to be applied must be chosen based on its estimated potential success; and *credit assignment* that assigns a reward value based on the observed performance on last iterations. An initial approach of this assignment is to account for the fitness improvement brought by the operators. That is, the fitness difference between the offspring with respect to a reference value, usually the parent fitness [19], [20], [32].

Our proposal is to use metrics that are based on characteristics of the neighbourhood surrounding a current solution to evaluate the impact of operators, and incorporate these metrics into the credit assignment mechanism. Lourenço *et al.* [16] supports the idea to incorporate a training phase of a simple and less accurate metric than a mere fitness evaluation to avoid the overfitting of the data problem in a selection hyper-heuristic. In this work, we use evolvability metrics as an option to construct credit assignment methodologies that are based not only on the current fitness improvement but also on the neighbourhood (local landscape)

of a current solution. Evolvability is loosely defined as the capacity to evolve [1], [27], i.e. the potential of an individual to produce offspring with better fitness value. The idea is to reward an individual for its potential rather than its fitness value; two individuals with equal fitness could have different potentials for evolution [27].

Evolvability can be seen as a metric of *how much we may expect* from a given solution when using a specific operator, so different operators could be given different evolvability values when they are applied to the same solution. The key idea is to use the operator that maximizes the evolvability value from a given solution. We considered evolvability metrics as the *credit assignment* rule in AOS in a previous study with encouraging results [28].

Calculating evolvability requires a sampling process that consumes fitness evaluations on the fly; therefore, it may be computationally expensive. In this paper, evolvability is estimated by the implementation of a parallel perceptron. By using the parallel perceptron, we accelerate this process because we are only processing an incoming solution with its fitness value through a single layer of perceptrons, which have been pre-trained to behave like an evolvability metric.

This parallel perceptron neural network uses the parallel delta (p-delta) rule [2]. In contrast to the back-propagation learning rule for multi-layer perceptrons [26], the p-delta rule only has to tune a single layer of weights, and it does not require the computation and communication of analogue values with high precision [2]. The main idea is to train a parallel perceptron to use it as an evolvability metric without expending unnecessary fitness evaluations during execution.

To test and compare our proposal, we select the algorithmic framework (high-level search strategy and relevant parameter settings) used in recent works on adaptive operator selection [11], [28], [29]. Three benchmark problems with binary representation are used. Namely, OneMax problem (also used in [11]), Royal Staircase family functions and the Multiple Knapsack problem (also used in [28]). The next section describes important concepts like evolvability, the parallel perceptron and its $p-delta$ learning rule. Our proposal is described in Section 3. Section 4 details the empirical setup, while Section 5 reports and analyses our results. Finally, Section 6 summarizes our findings and suggests directions for future work.

II. RELEVANT CONCEPTS

A. EVOLVABILITY

The first formalisation of the Evolvability concept is attributed to Altenberg [1]. In this work, evolvability is defined as "the ability of the genetic operator/representation scheme to produce offspring that are fitter than their parents." This is desirable feature for adaptation in natural and artificial systems relevant to adaptive operator selection.

We are interested in measuring the potential of this operator/solution pair. The key idea is to evaluate the evolvability value of an incumbent solution along with an operator taken from a pool. The operator that gives us the best evolvability value using an incoming solution will have greater chances to be selected to guide the search at further iterations.

Several evolvability metrics have been proposed in the literature [18]. In this paper we focus on a simple and well-know evolvability metric named E_a, proposed in [27]. E_a is defined as the probability of the offspring's fitness being higher or equal to the parent's fitness (Equation 1). The use of this E_a evolvability metric as a *credit assignment* rule in adaptive operator selection has been studied previously in Soria-Alcaraz *et al.* 2014 with encouraging results [28]. Following notation and definitions by Smith *et al.* [27] we define the E_a metric as: let (V, E) be the fitness landscape with vertices V (solutions) connected by edges E. There is an edge between two solutions if one is generated from the other through a single application of a given operator/low level heuristic. Let the pair $< h, k >$ represents a solution with genotype h and fitness k. The set, G, of offspring $< h, k >$ is determined by the vertices connected to the parent solution/vertex: $G(< h, k >) = \{g \in V : E(< h, k >) = g\}$.

The fitness function, F, maps every solution to a single $\mathbb{R}$ value. Then, we can define a set of offspring with fitness $F(g)$ greater than a given value:

$$G_c^+(< h, k >) = \{g \in V : E(< h, k >) = g, F(g) \geq c\}$$

E_a can then be described as the probability of a descendant to have a higher (or at least the same) fitness value than that of its parents. More formally, the ratio between the number of offspring with fitness higher or equal to that of the parent. Equation 1 details this ratio.

$$E_a = \frac{|G_k^+(< h, k >)|}{|G(< h, k >)|} \tag{1}$$

A sampling methodology is required to give an approximation of this E_a value since it could be impractical to evaluate each possible offspring from a given *Operator*/*Solution* pair. This sampling method needs to follows a uniform distribution to guarantee a proper exploration of the neighbourhood space around a solution. However, several of the solutions or points contained in the neighboring space may not be useful [31]. Therefore, a desirable characteristic for our sampling process is the ability to give priority to solutions with higher fitness. As suggested in [31], we use the *Metropolis-Hastings* sampling algorithm to meet these requirements. The *Metropolis-Hastings* extends the standard Metropolis sampling to non-symmetric stationary probability distributions.

The Metropolis-Hasting sampling method is used for estimating the E_a metric collecting information about the offspring produced from the incumbent solution and a given operator. A sampling size of 15 (offspring) is used in our experiments. This empirically found value demonstrated good performance in terms of both: computation expenses and metric approximation quality.

B. THE PARALLEL PERCEPTRON

A perceptron neuron with a given z input of size d can be conceived as a function f from $\mathbb{R}^d$ into $\{-1, 1\}$ as seen in Equation 2.

$$f(z) = \begin{cases} 1 & \text{if } \alpha \cdot z \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha \in \mathbb{R}^d$ is the weight vector of the perceptron, and $\alpha \cdot z$ denotes the usual vector product (with one of the inputs as a constant bias input).

Auer *et al.* [2] defines a *parallel perceptron* as a single layer consisting of a finite number of n perceptrons (without lateral connections). Let $f_1, \ldots, f_n$ be the functions from $\mathbb{R}^d$ into $\{-1, 1\}$ that are represented by these perceptrons. For an input, z, the output of the parallel perceptron are the values seen in Equation 3,

$$x = \sum_{i=1}^{n} f_i(z) \in \{-n, \ldots, n\} \tag{3}$$

this value needs to be escalated using $s(\sum_{i=1}^{n} f_i(z))$, where $s : \mathbb{Z} \rightarrow \mathbb{R}$ is a squashing function that scales the output into the desired range. [2].

In this paper the squashing function used to scale the raw input from equation 3 is a simple linear transformation, this function can be seen in equation 4

$$s(x) = \frac{x + n}{2n} \tag{4}$$

where x is the output value from equation 3 and n is the number of perceptrons used in our parallel array.

C. PARALLEL DELTA RULE

The parallel array of perceptrons described in section II-B uses a simple but powerful learning rule defined by Auer *et al.* [2] as the *parallel delta (p-delta) rule*.

The *p-delta* rule has two components, a classical delta rule which is applied to a subset of perceptrons and a component which decides whether the classical delta rule should be applied to a specific perceptron.

For completeness, we replicate the *p-delta rule* in Figure 1, we invite interested readers to analyze Auer *et al.* [2] to fully understand the power of this rule.

The p-delta rule has a set of parameters (Figure 1) we now detail the values used in our implementation; $\hat{o}$ refers to the actual output of the parallel perceptron calculated by Equation 4, o refers to the desired output for that specific input z. α_i means the $i-th$ perceptron in our array, η represents the learning rate (we select this value as 0.0001 through preliminary experimentation), ε is the desired error (we fix this error as 0.01), μ means a margin used by the p-delta rule to reinforce the learning phase, following Auer *et al.* [2] we fix this value to 1, γ stabilizes the output of the perceptron array keeping $\alpha \cdot z$ away from 0, in our implementation γ has the value of 0.01. We use this parallel delta rule to train our parallel perceptron.

p-delta rule
For all $i = 1, \ldots, n$
(a)

$$\alpha_i \leftarrow \alpha_i + \eta \begin{cases} (-z) & \text{if } \hat{o} > o + \varepsilon \text{ and } \alpha \cdot z \geq 0 \\ (+z) & \text{if } \hat{o} < o - \varepsilon \text{ and } \alpha \cdot z < 0 \\ \mu(+z) & \text{if } \hat{o} \leq o + \varepsilon \text{ and } 0 \leq \alpha \cdot z < \gamma \\ \mu(-z) & \text{if } \hat{o} \geq o - \varepsilon \text{ and } -\gamma < \alpha \cdot z < 0 \\ 0 & \text{otherwise} \end{cases}$$

(b)

$$\alpha_i \leftarrow \alpha_i / ||\alpha_i||$$

FIGURE 1. ***p*-delta rule taken from Auer *et al.* [2].**

III. METHODOLOGY

We investigate the use of the parallel perceptron as evolvability metric estimator in a selection hyper-heuristic. An adaptive operator selection scheme consists of two components: (i) a *credit assignment* mechanism, which associates a reward with each operator and (ii) a *selection rule*, which makes the effective selection of the operator to be used on latter iterations. It is within the credit assignment mechanism that the parallel perceptron as evolvability metric is introduced as described in subsection III-A. Subsection III-B.2 describes the selection rules implemented, while in subsection III-B the high-level search strategy used.

A. THE PARALLEL PERCEPTRON AS EVOLVABILITY METRIC

As seen in section II-A, Equation 1, the E_a evolvability metric gives a value in the range [0, 1] that represents the probability for a potential offspring to achieve a better fitness than its parent using a specific heuristic. This value could be used as a *credit assignment operator* where an incoming solution is paired with each operator in order to assign a potential value of success for each pair. The idea is to select the operator that maximizes this evolvability value for an incoming solution. An important disadvantage of the usage of evolvability metrics in credit assignment is the necessity of investing function evaluations to reach a representative evolvability value. A properly pre-trained parallel perceptron could behave like a classical evolvability metric, making it possible to achieve similar evolvability values for each solution-operator pair at a lower computational cost.

The execution of a pre-trained parallel perceptron (subsection II-B) is straightforward, since it only requires: d inputs, α_i weights and a squashing function. In this paper we use as inputs the actual solution state along with its fitness evaluation. An example of this configuration can be seen in Figure 2, considering the OneMax Problem with a bit-string length of 5 (we use an extra input with the value of 1 as a bias). Our squashing function is the linear expression shown in equation 4. Finally, the set of α_i weights is obtained through a training process detailed in section III-A.1.

Input z					Output (o)
Binary State					Fitness	Target (E_a)
0	0	0	0	0	0	1
0	0	1	0	0	1	0.8
1	0	1	0	0	2	0.6
1	1	1	0	1	4	0.2

FIGURE 2. Training data example for 1-Flip Operator in the OneMax problem.

1) TRAINING PHASE

In order to achieve a set of appropriate α_i weights for our parallel perceptron we use the p-delta rule detailed in section II-C. This learning rule allows us to train our parallel perceptron to behave as the E_a evolvability metric. In order to execute the p-delta rule for each operator we build a training dataset. These datasets were obtained by means of the implementation of a standard $(1+\lambda)$ Evolutionary Algorithm (EA) without recombination, where $\lambda = 50$ offspring are created through the application of a given operator from a single parent (allowing us to calculate the E_a value of the last pair operator/parent) and the best individual among the current offspring and parent becomes the parent in the next generation, when a new state has a better fitness value than its parent we store it in a data structure as seen in Figure 2. Algorithm 1 details our data set construction.

Once we obtain a $dataset_{op}$ for each heuristic/operator, we train a parallel perceptron for each desired operator. We use the p-delta values detailed in section II-C during 5000 epochs. Finally, our parallel perceptron implementation contains 100 perceptrons.

B. SELECTION HYPER-HEURISTIC

A selection hyper-heuristic algorithm has three main components: the high-level search strategy, the pool of operators and the adaptive or control mechanism to dynamically select the operator to apply at each search step. This section describes the high-level strategy and adaptive operator selection mechanism used by the proposed approach. The pool of operators is normally problem-specific.

1) HIGH-LEVEL STRATEGY

In order to make a fair comparative between this and previous works, we implemented an Iterated Local Search algorithm(ILS) as high-level strategy [29]. Iterated Local Search is a simple and effective algorithm by Lourenço *et al.* [15]. This algorithms works iteratively alternating between an exploration move (pertubation) and a exploitation move (local search) from the perturbed solution. Variants of this algorithm when used along with selection hyper-heuristics has been reported previously with encouraging results [23], [29].

Our implementation is outlined in Algorithm 2. The adaptive control mechanism is applied to the improvement stage, in which a local search heuristic is selected (Selection rule, credit assignment) from the available pool and then applied to the incumbent solution (line 5). The perturbation stage uses a fixed randomized operator, and the acceptance condition simply accepts all improvements. This implementation differs from our previous ILS hyper-heuristic [29] in the operator control mechanism for selecting heuristics, as shown in Algorithm 2.

Algorithm 1 $(1+\lambda) - EA$ for Training Dataset Construction

Require: F : fitness function, *sample* : number of samples, size of parent pool (1), λ : size of the offspring pool, *op* : Heuristic or operator to sample.

1: $ParentPool_0 = generatePopulation(1)$
2: $currentSamples = 0, t = 0$
3: *initialize* $dataset_{op}$
4: **while** $dataset_{op}.size() < samples$ **do**
5: $t++$
6: initialize $OffspringPool_t = apply(op)\ \lambda$ *times to* $ParentPool_{t-1}$
7: *calculate* E_a *value for* $ParentPool_{t-1}$ *using* $OffspringPool_i$
8: $OffspringPool_t.add(ParentPool_{t-1})$
9: $ParentPool_t = Best(OffspringPool_t)$
10: **if** $F(ParentPool_t)$ *best than* $F(ParentPool_{t-1})$ **then**
11: *Store* $ParentPool_{t-1}$, $F(ParentPool_{t-1})$ *and* E_a *value*
12: **end if**
13: **end while**
14: **return** $dataset_{op}$

Algorithm 2 High-Level Strategy: Iterated Local Search

1: $s_0 = GenerateInitialSolution$
2: $s^* = ImprovementStage(s_0)$
3: **while** !$StopCriteria()$ **do**
4: $s' = SimpleRandomPerturbation(s^*)$
5: $s^{*'} = ImprovementStage(s')$
6: **if** $F(s^{*'})$ *better than* $F(s^*)$ **then**
7: $s^* = s^{*'}$
8: **end if**
9: **end while**
10: **return** s^*

2) OPERATOR SELECTION

Two components are required in this phase: A *selection rule*, which defines the operator to be used in latter iterations according to its estimated quality; and a *credit assignment* mechanism, which defines how to estimate the operator quality based on the performance of its most recent application. These mechanisms are described in detail below.

a: SELECTION RULE

We use dynamic multi-armed bandit (DMAB) [8] as selection rule, where each operator is viewed as an arm. Let $l_{i,t}$ denote the number of times the i^{th} arm has been played, and $\hat{r}_{i,t}$ the

average empirical reward it has received up to time t. At each time step t, from K alternative arms, the algorithm selects the arm maximising the quantity computed by expression 5.

$$\hat{r}_{j,t} + C\sqrt{\frac{2log(\sum_{i=1}^{K} l_{i,t})}{l_{j,t}}} \tag{5}$$

A scaling factor C is needed in order to achieve a balance between exploration and exploitation phases. Also, since the multi-armed bandit framework is combined with a *Page-Hinkley* statistical test, two additional parameters associated with the *Page-Hinkley* are introduced; they are: γ_{ph}, which controls the trade-off between false alarms and unnoticed changes; and δ, which enforces the robustness when dealing with slowly varying environments. Parameters C and γ_{ph} need to be tuned for every problem. We found in preliminary experiments that the values $C = 10$, $\gamma_{ph} = 100$ obtain encouraging results consistently. For the parameter δ, we used the value suggested in [11] ($\delta = 0.15$) for all our experiments.

b: CREDIT ASSIGNMENT

We use an *extreme value* criteria for determining the operator's credit [11], [12]. Rewards are updated as follows, when an operator *op* is selected, it is applied to the current solution. Afterwards we use our trained Parallel Perceptron (section III-A) to *estimate* its E_a value. This E_a value is added to a FIFO list of size W. A separate list is kept for each operator. Thereafter, the operator reward is updated to the maximal E_a in the list. More formally, let t be the current step and $E_a(t)$ the E_a evolvability value estimated by our parallel perceptron at time t for a specific heuristic *op*, the expected reward $\hat{r}_t$ for heuristic *op* is computed using equation 6.

$$\hat{r}_t = argmax\{E_a(t_i), i = 1 \dots W\} \tag{6}$$

IV. EXPERIMENTS

Our experimental setup is detailed in this section along to the parameters, algorithms and statistical tests used.

A. TEST PROBLEMS

For this work we select three binary string based domains, we have selected these domains in order to analyze in this first work the potential of use of a parallel perceptron as metric estimation for on-line selection hyper-heuristics. The domains are:

Onemax or counting one's problem used in many theoretical and proof of concept papers [11].

1) ROYAL STAIRCASE FUNCTIONS

This binary string based domain belongs to the *Royal Road* functions [21]. These binary string based functions are characterized by long periods of invariability of the fitness followed by occasional and abrupt changes [22]. Genotypes of this domain are binary strings divided in N_r blocks where each block holds K_r bits per block. The fitness value of a given string is the number of blocks corresponding to 1 plus the number of consecutive fully-set blocks starting from the left i.e. blocks that only hold values of 1. N_r and K_r values determine a particular instance of this domain.

2) MULTIPLE KNAPSACK PROBLEM

This domain comes from the single Knapsack problem where the main objective is to fill a knapsack of capacity C with a set of elements, each one with a profit p_i and a weigh w_i, looking for the assignation that produces the maximum profit.

The multiple version consists of m knapsacks of capacities $c_1, c_2, \dots, c_m$ and n objects with profits $p_1, p_2, \dots, p_n$. Each object has m possible weights: object i weighs w_{ij} when considered for inclusion in knapsack j ($1 \leq j \leq m$). Again, the objective is to find a packing that guarantees that no knapsack is overfilled: $\sum_{i=1}^{n} w_{ij}x_i \leq c_j$ for $j = 1, 2, \dots, m$; and that achieves the maximum profit $P(x) = \sum_{i=1}^{n} p_i x_i$ [14].

B. ALGORITHMS AND PARAMETER SETTINGS

As discussed in section III-B we use the Iterated Local Search method (Algorithm 2) as a high level search. Also, we select a pool of operators to use with our domains. The family of operators selected was the standard *n-flip*. These operators choose uniformly at random n bits in the current solution and flip their values (0 is changed to 1 and vice-versa). Our implementation used n values of 1, 3 and 5 [11].

As for algorithms variants, three Adaptive Operator Selector (AOS) variants were considered by combining three alternative credit assignment mechanisms: fitness improvement (*Fit*), estimated evolvability metric thought sampling taken from a previous study [28] (E_a-*S*) and estimated evolvability metric through parallel perceptron (E_a-*Pp*). For the selection rule we use DMAB (section III-B.2). Notice that *Fit* variant corresponds to the algorithm proposed in [11]. Finally, One last algorithm is selected as a control method; this algorithm identified as *Random* simply selects operators uniformly at random at each iteration.

Parameter settings used in the experiments are detailed in Table 1. Common values for all experiments (the credit assignment window size W, the Metropolis-Hastings sampling size for calculating evolvability metrics, and the control parameter δ associated to the *dmab* rule) can be seen in the first three entries of Table 1. Entries in Table 1 show the parameter values specific to each test problem. Specific parameters are: the maximum number of generations for the evolution strategy *MaxGener*, the chromosome length L, and the selection rule control parameters (C, γ_{ph} for *dmab*). OneMax values follow the suggestions in [11]. For the remaining problems, parameter values were obtained empirically.

V. RESULTS AND ANALYSIS

For each algorithm variant and test instance 35 independent runs were conducted. The stopping criteria consist in the number of iterations of our ILS. All domains are maximization problems. We record the fitness function value after the execution each run. For each instance, we train a parallel perceptron to behave like the E_a metric, as shown in section III-A

TABLE 1. Parameter settings.

Parameter	*OneMax*	*Royal Staircase*	*M. Knapsack*
W	30	30	30
S	7	7	7
δ	0.15	0.15	0.15
$MaxGener$	1,000	10,000	5,000
L	10,000	100 to 700	50 to 105
γ_{ph}	100	75	80
C	10	7	7.5

TABLE 2. Parallel perceptron training time in seconds for selected instances. (5000 epochs).

Domain	Time
$OneMax_{10,000}$	75.1
$Royal\ Staircase_{N40K5}$	112.3
$Royal\ Staircase_{N100K7}$	133.1
$Multiple\ Knapsack_{Weing7}$	97.2
$Multiple\ Knapsack_{Sento1}$	114.7

TABLE 3. Descriptive statistics of main AOS variants across all test problems.

OneMax	Mean	Median	Std.dev	Best
Ea-Pp	**8152.31**	**8101**	**27.50**	**8251**
Ea-S	8074.12	8050	27.92	8200
Fit	7832.52	7750	27.42	7957
Random	6834.53	6812	30.01	6892
R.StairCase	Mean	Median	Std.dev	Best
Ea-Pp	**17.82**	**17**	**12.5**	**21**
Ea-S	16.45	16	16.03	19
Fit	10.97	11	12.01	16
Random	10.78	10	13.63	15
M.Knapsack	Mean	Median	Std.dev	Best
Ea-Pp	**773562.12**	**753261**	**72378.4**	**894423**
Ea-S	706600.00	718500	73105.9	863198
Fit	495423.24	512432	77231.6	697673
Random	276634.77	269983	78332.4	35728

we execute 5000 epochs per instance. Table 2 shows the time in seconds used to train the parallel perceptron.

Table 3 summarizes basic descriptive statistics calculated from the final fitness values achieved by our algorithm variants, the compared algorithms are: Our proposal *Ea-Pp*, E_a-*S* from [28], *Fit* from [11] and the control algorithm named as Random across all domains. These domains are maximization problems so, bigger values means best results. Results indicate that our proposal is superior to other algorithms. Importantly, the calculation of evolvability through sampling (Ea-s) incurs additional computational costs, this situation is not present in our proposed Ea-Pp. Table 4 presents the approximate execution time observed in our experiments. However, this evidence is not enough to ensure that our proposal has the better performance with statistical significance.

Analyzing our results per instance and algorithm we found that data follows approximately a normal distribution and that they have stable variances, this was checked by Shapiro-Wilk tests. Therefore, we use parametric statistical tests to analyze adequately the performance of all algorithms.

In the case of Onemax domain we use one-way ANOVA F Test since there is a single factor (Algorithm Variant)

TABLE 4. Empirical running times in seconds on selected test instances.

Test Instance	Fit-dmab	E_a-S	E_a-Pp
$OneMax_{10,000}$	7.56	8.32	7.74
$Royal\ Staircase_{N40K5}$	7.48	12.75	7.76
$Royal\ Staircase_{N100K7}$	16.27	37.26	17.01
$Multiple\ Knapsack_{Weing7}$	0.84	1.12	0.91
$Multiple\ Knapsack_{Sento1}$	3.45	5.14	3.75

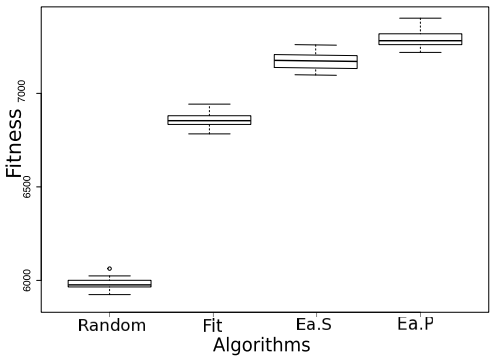


FIGURE 3. Fitness distributions for Onemax 10000.

that can explain performance differences in the results. For the royal staircase and multiple knapsack domains two-way Anova F tests are used because algorithm variant is not the only factor to analyze but also the specific instance can explain observed differences in the results.

A pairwise *t* test is applied post-hoc to identify specific differences in the performance (if exists) of a given pair of Algorithms. Furthermore, the p-value for each test is computed including a Bonferroni correction to ensure that the effect of the so-called family-wise error is controlled. Also Tukey HSD tests are applied to uphold *t* tests. The significance level used across all tests is 0.05. Finally, the null hypothesis (H_0) of all these tests are that there are not significant differences between the means of the results provided for the tested algorithms.

A detailed analysis of our results across all domains is presented in following sections.

A. OneMax

The magnitude and distribution of 35 independent executions for each algorithm with the Onemax domain of size 10000 is illustrated in Figure 3, the E_a-*Pp* algorithm outperforms other approaches. The one-way Anova f test reported in Table 4 supports the existence of differences between the means of the algorithms with statistical significance. Pairwise *t* tests were also conducted using Bonferroni correction to identify if there exist a difference between the results of a given pair of algorithms with statistical significance. We are specifically interested in differences between our approach (E_a-*Pp*) and other methods.

The Anova F Test in Table 5 shows evidence about the existence of differences between the means of the results obtained

TABLE 5. Onemax. One-way ANOVA F test, and pairwise *t* test.

ANOVA	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithms	3	36173892	12057964	6655	2.13e-16
Residuals	124	224686	1812		
t **Tests**	E_a-Pp		Ea-S		Fit
Ea-S	**0.011**		-		-
Fit	2.1e-6		2.1e-6		-
$Random$	2.1e-6		2.1e-6		2.1e-6

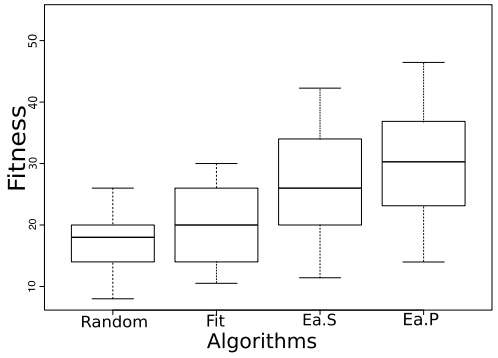


FIGURE 4. Fitness distributions for N80K5 instance.

by the execution of the algorithms. The pairwise t tests with Bonferroni correction enforces this evidence because each time our proposal is compared against other methods it reports a $p-value$ less than our significance level 0.05; this rejects the null hypothesis i.e. the E_a-Pp algorithm produces results that are significantly different when compared against the other algorithms. Therefore, we can ensure that the performance of our proposal shown by Figure 3 is consistently since we now have evidence with statistical significance that supports this.

B. ROYAL STAIRCASE

Royal Staircase functions are characterized by a high ruggedness level given by the numbers of blocks (N_r) and their size (K_r). Four combinations of $< N_r, K_r > : < 20, 5 >$, $< 40, 5 >, < 80, 5 >, < 10, 7 >$ are used in this section as instances of Royal Staircase functions. The two-way Anova f test is applied to the data obtained by the execution of the 4 algorithms variants over the 4 royal staircase instances, results of this test are reported in Table 6. The subsequent application of both: pairwise test with Bonferroni correction and the Tukey HSD test with confidence level of 95% supports the evidence that the E_a-Pp algorithm, proposed in this paper, produces results that differs with statistical significance against the other methods. Similar to the Onemax domain, Figure 4 reports boxplots obtained by our experimentation. Again, each boxplot represents 35 independent executions of each algorithm over the $< 80, 5 >$ instance (named N80K5) similar behavior was observed across all the royal staircase instances. Results from table 6 supports the evidence represented by Figure 4, which shows the algorithm E_a-Pp as the best algorithm for Royal Staircase instances.

TABLE 6. Royal staircase. Two-way ANOVA F test, pairwise *t* test and Tukey HSD test.

ANOVA	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithm	3	1744.9	581.62	49.332	**1.17e-16**
Instance	3	7590.6	2530.9	214.606	2.2e-16
Residuals	521	6142.6	11.79		
t **Tests**	E_a-Pp		Ea-S		Fit
Ea-S	**0.014**		-		-
Fit	**5.1e-8**		1.2e-5		-
$Random$	**6.5e-10**		2.8e-7		1.3e-3
TukeyHSD	diff		lwr	upr	**p adj**
E_a-S vs E_a-Pp	-1.65		-2.74	0.43	**0.00**
Fit vs E_a-Pp	-3.68		-4.77	-2.60	**0.00**
Rand vs E_a-Pp	-4.15		-5.24	-3.06	**0.00**
Fit vs E_a-S	-3.03		-4.11	-1.94	0.00
Rand vs E_a-S	-3.49		-4.58	-2.40	0.00
Rand vs Fit	-0.46		-1.55	0.62	0.019

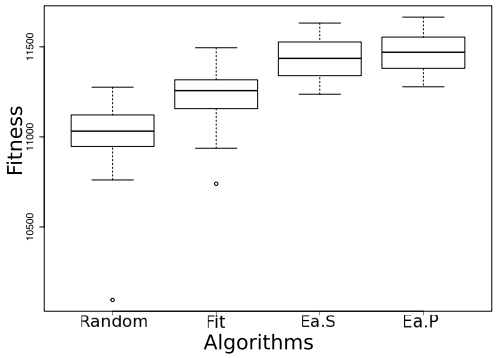


FIGURE 5. Fitness distributions for Weish30 instance.

TABLE 7. Multiple knapsack. Two-way ANOVA F test, pairwise *t* test and Tukey HSD test.

ANOVA	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithm	3	2.66e9	8.86e8	3.39e1	< 2.2e-16
Instance	5	1.73e14	3.47e13	1.32e6	< 2.2e-16
Residuals	783	2.04e10	2.61e7		
t Tests	E_a-Pp		Ea-S		Fit
Ea-S	**3.20e-2**		-		-
Fit	**2.32e-3**		3.14e-3		-
$Random$	**9.35e-5**		8.14e-4		7.34e-3
TukeyHSD	diff		lwr	upr	**p adj**
Ea-S **vs** Ea-Pp	-742		-2065	581	**2.99e-3**
Fit **vs** Ea-Pp	-1941		-3264	-618	9.75e-4
Rand vs Ea-Pp	-4812		-6135	-3489	0.00
Fit vs Ea-S	-1199		-2522	123	**9.11e-2**
Ran vs Ea-S	-4070		-5393	-2747	0.00
Rand vs Fit	-2871		-4194	-1547	0.00

C. MULTIPLE KNAPSACK

For the Multiple Knapsack problem, we select six instances, these instances are available online in the OR-library by Beasley.[1] Selected Instances varies from 50 to 105 objects and from 2 to 50 knapsacks. All instances are considered as multi-modal constrained problems.

[1]The OR Library is available at http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Following our previous experimentation, we apply a two-way Anova test to the data obtained by 35 independent executions of each algorithm over our selected instances. Also, we apply several pairwise t tests with Bonferroni correction and a Tukey HSD test with confidence level of 95% as post-hoc procedures. Table 7 reports our results for this domain. Figure 5 presents box-plots with the performance of all algorithm variants.

Again, our approach achieves better results when compared against other similar approaches, this evidence suggests that the use of a Parallel perceptron as estimator of an evolvability metric improves the performance of an on-line selection hyper-heuristic that utilize it as credit assignment mechanism, at least in the case of Onemax, Royal Staircase and Multiple Knapsack instances.

VI. CONCLUSIONS

We investigated the benefits of using a parallel perceptron as a estimator for an evolvability metric when used as a credit assignment mechanism in adaptive operator selection, which is a component of online selection hyper-heuristics. Traditionally, evolvability metrics are estimated through a sampling process, which increases the computational effort required by its calculation. Our approach, in contrast, offers a less computational-stressing method to estimate the E_a evolvability metric using a single layer of perceptrons. This layer was trained by a $p-delta$ rule which only has to tune a single layer of weights. Results on the selected binary-based problems reveal firstly, that the use of evolvability metrics as credit assigment mechanisms produces statistically significant positive results when compared against the use of fitness based metrics, second, the use of evolvability metrics could be improved to estimate these metrics with less exhaustive calculations, such as the application of a parallel perceptron. Our research contributes to the goal of using features of the local fitness landscape to inform dynamic self-configuring algorithms.

As for future work, real-world optimization problems such as timetabling and Vehicle routing problem will be tested using this approach. An idea to apply our approach to non-binary domains could be to normalize the values âĂŃâĂŃof each non-binary variable, by doing this it is possible to process each value without extra changes by our methodology. Our study uses an Iterated Local search strategy with a pool of mutation operators as the high-level search strategy. Other meta-heuristic algorithms can be used as high level strategy. In this paper only mutation operators are used, it is worth to test recombination and crossover operators or even construction-destruction operators. This research is therefore relevant to selective hyper-heuristics with adaptive large neighbourhood search.

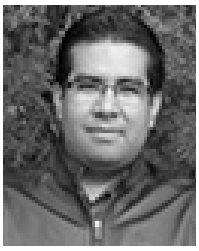
ACKNOWLEDGMENT

The authors wish to thank the Directorate for Research Support and Postgraduate Programs at the University of Guanajuato for their support in the translation and editing of the English-language version of this article.

REFERENCES

[1] L. Altenberg, "The evolution of evolvability in genetic programming," in *Advances in Genetic Programming*. Cambridge, MA, USA: MIT Press, 1994, pp. 47–74.
[2] P. Auer, H. Burgsteiner, and W. Maass, "A learning rule for very simple universal approximators consisting of a single layer of perceptrons," *Neural Netw.*, vol. 21, no. 5, pp. 786–795, 2008.
[3] R. C. Barros, M. P. Basgalupp, and A. C. P. L. F. de Carvalho, "Investigating fitness functions for a hyper-heuristic evolutionary algorithm in the context of balanced and imbalanced data classification," *Genetic Programm. Evol. Mach.*, vol. 16, no. 3, pp. 241–281, 2015.
[4] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2002, pp. 11–18.
[5] E. K. Burke *et al.*, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.
[6] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, "Handbook of Metaheuristics," *International Series in Operations Research & Management Science* (A Classification of Hyper-heuristic Approaches), vol. 146. New York, NY, USA: Springer, 2010, ch. 15, pp. 449–468.
[7] E. K. Burke *et al.*, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013.
[8] L. DeCosta, A. Fialho, M. Schoenauer, and M. Sebag, "Adaptive operator selection with dynamic multi-armed bandits," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2008, pp. 913–920.
[9] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Practice and Theory of Automated Timetabling III* (Lecture Notes in Computer Science), E. Burke and W. Erben, Eds., vol. 2079. Berlin, Germany: Springer, 2001, pp. 176–190.
[10] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 19–31, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210650211000022
[11] Á. Fialho, L. D. Costa, M. Schoenauer, and M. Sebag, "Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms," in *Learning and Intelligent Optimization* (Lecture Notes in Computer Science), vol. 5851. Heidelberg, Germany: Springer, 2009, pp. 176–190.
[12] Á. Fialho, L. D. Costa, M. Schoenauer, and M. Sebag, "Analyzing bandit-based adaptive operator selection mechanisms," *Ann. Math. Artif. Intell.*, vol. 60, no. 1, pp. 25–64, 2010.
[13] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, no. 1, pp. 267–306, 2009.
[14] S. Khuri, T. Bäck, and J. Heitkötter, "The zero/one multiple knapsack problem and genetic algorithms," in *Proc. ACM Symp. Appl. Comput.*, 1994, pp. 188–193.
[15] H. R. Lourenço, O. C. Martin, T. Stützle, "Iterated local search," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), F. Glover, G. Kochenberger, F. S. Hillier, Eds., vol. 57. New York, NY, USA: Springer, 2003, pp. 320–353.
[16] N. Lourenço, F. B. Pereira, and E. Costa, *The Optimization Ability of Evolved Strategies*. Cham, Germany: Springer, 2015, pp. 226–237. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23485-4_23
[17] S. Luke and A. K. M. Talukder, "Is the meta-EA a viable optimization method?" in *Proc. 15th Annu. Conf. Genetic Evol. Comput. Conf. (GECCO)*, New York, NY, USA, 2013, pp. 1533–1540.
[18] K. M. Malan and A. P. Engelbrecht, "A survey of techniques for characterising fitness landscapes and some possible ways forward," *Inf. Sci.*, vol. 241, pp. 148–163, Aug. 2013.
[19] Y. Martínez, L. Trujillo, P. Legrand, and E. A. Galván-López, "Prediction of expected performance for a genetic programming classifier," *Genetic Programm. Evol. Mach.*, vol. 17, no. 4, pp. 409–449, 2016.
[20] J. Maturana and F. Saubion, "A compass to guide genetic algorithms," in *Parallel Problem Solving From Nature—PPSN X*. Berlin, Germany: Springer, 2008, pp. 256–265.

[21] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Proc. 1st Eur. Conf. Artif. Life. Toward Pract. Auto. Syst.*, Cambridge, MA, USA, 1992, pp. 245–254.
[22] E. van Nimwegen and J. P. Crutchfield, "Optimizing epochal evolutionary search: Population-size dependent theory," Santa Fe Inst., Santa Fe, NM, USA, Tech. Rep. 98-06-046, 1998.
[23] G. Ochoa, J. Walker, M. Hyde, and T. Curtois, "Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework," in *Parallel Problem Solving from Nature—PPSN XII* (Lecture Notes in Computer Science), vol. 7492. Berlin, Germany: Springer, 2012, pp. 418–427.
[24] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal.*, vol. 12, no. 1, pp. 3–23, 2008.
[25] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan, "Contrasting meta-learning and hyper-heuristic research: The role of evolutionary algorithms," *Genetic Programm. Evolvable Mach.*, vol. 15, no. 1, pp. 3–35, 2014.
[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[27] T. Smith, P. Husbands, P. Layzell, and M. O'Shea, "Fitness landscapes and evolvability," *Evol. Comput.*, vol. 10, no. 1, pp. 1–34, 2002.
[28] J. A. S. Alcaraz, G. Ochoa, M. Carpio, and H. Puga, "Evolvability metrics in adaptive operator selection," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, Vancouver, BC, Canada, Jul. 2014, pp. 1327–1334.
[29] J. A. Soria-Alcaraz, G. Ochoa, J. Swan, M. Carpio, H. Puga, and E. K. Burke, "Effective learning hyper-heuristics for the course timetabling problem," *Eur. J. Oper. Res.*, vol. 238, no. 1, pp. 77–86, 2014.
[30] A. Sosa-Ascencio, G. Ochoa, H. Terashima-Marin, and S. E. Conant-Pablos, "Grammar-based generation of variable-selection heuristics for constraint satisfaction problems," *Genetic Programm. Evolvable Mach.*, vol. 17, vol. 2, pp. 119–144, 2016.
[31] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Verel, "Fitness clouds and problem hardness in genetic programming," in *Genetic and Evolutionary Computation—GECCO* (Lecture Notes in Computer Science), vol. 3103. Berlin, Germany: Springer, 2004, pp. 690–701.
[32] N. Veerapen, J. Maturana, and F. Saubion, "An exploration-exploitation compromise-based adaptive operator selection for local search," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2012, pp. 1277–1284.



JORGE A. SORIA-ALCARAZ received the B.S. degree in computational systems engineering and the master's degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de León in 2008 and 2010, respectively, and the Ph.D. degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de Tijuana in 2015. He has done several research stays at the Monterrey Institute of Technology and Higher Education in 2009 and Stirling Univesity, U.K., in 2014. He is currently a full-time Professor with the Organizational Studies Department, Universidad de Guanajuato. He has authored several journals and proceedings papers. His research lines cover evolutionary algorithms, autonomous search, hyper-heuristics, adaptive operator selection, heuristic optimization, and bio inspired algorithms.

ANDRÉS ESPINAL received the B.S. degree in computational systems engineering and the master's degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de León in 2009 and 2011, respectively, and the Ph.D. degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de Tijuana in 2017. He is currently a full-time Professor with the Organizational Studies Department, Universidad de Guanajuato. He has authored several journals and proceedings papers. His research lines cover evolutionary algorithms, artificial neural networks, computer vision, digital image processing, and bio inspired algorithms.

MARCO A. SOTELO-FIGUEROA received the B.S. degree in computational systems engineering and the master's degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de León in 2006 and 2010, respectively, and the Ph.D. degree in computer science from the Tecnológico Nacional de México–Instituto Tecnológico de Tijuana in 2015. He is currently a full-time Professor with the Organizational Studies Department, Universidad de Guanajuato. He has authored several journals and proceedings papers. His research lines cover evolutionary algorithms, heuristic optimization, numeric optimization, and bio inspired algorithms.