Applied Soft Computing 53 (2017) 88–96

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Not all PBILs are the same: Unveiling the different learning mechanisms of PBIL variants

M. Zangari [a,*], R. Santana [b], A. Mendiburu [b], A.T.R. Pozo [a]

[a] DInf – Federal University of Parana, CP: 19081, CEP 19031-970, Curitiba, Brazil
[b] Intelligent System Group, University of the Basque Country, San Sebastian, Spain

ARTICLE INFO

Article history:
Received 2 September 2015
Received in revised form
10 November 2016
Accepted 21 December 2016
Available online 3 January 2017

Keywords:
Probabilistic modeling
PBIL
Estimation of distribution algorithm

ABSTRACT

Model-based optimization using probabilistic modeling of the search space is one of the areas where research on evolutionary algorithms (EAs) has considerably advanced in recent years. The population-based incremental algorithm (PBIL) is one of the first algorithms of its kind and it has been extensively applied to many optimization problems. In this paper we show that the different applications of PBIL reported in the literature correspond, in fact, to two essentially different algorithms, which are defined by the way the learning step is implemented. We analytically and empirically study the impact of the learning method on the search behavior of the algorithm. As a result of our research, we show examples in which the choice of a PBIL variant can produce qualitatively different outputs of the search process.

1. Introduction

Estimation of Distribution Algorithms (EDAs) are model-based evolutionary algorithms that have been successfully applied for solving combinatorial optimization problems [27,42,30,35]. These algorithms, instead of using the classical genetic operators, use machine learning techniques to estimate a probability distribution associated with the set of most promising solutions, and use this probabilistic model to sample new candidate solutions.

The population-based incremental learning algorithm (PBIL) [2] is one of the simplest model-based evolutionary algorithms and arguably one of the first EDAs [35]. The genesis of PBIL was largely influenced by previous work on population-based recombination in genetic algorithms (GAs) [43] and the concept of competitive learning, as applied in neural networks [22]. In a string of papers, Baluja et al. [2,5,3], described how the exchange of information between solutions, implicit in the behavior of crossover operators, could be efficiently transformed in the explicit manipulation of probabilistic vectors describing the statistics of the population.

In simple terms, at each generation, PBIL works by updating a vector describing the univariate statistics of the best solutions. The update of this simple model is governed by a parameter that works as a learning rate. The model is used to generate new candidate solutions and the loop comprising selection, model learning, and model sampling continues until a stop criterion is met.

The simplicity of PBIL and its ability to retain much of the performance of GAs with uniform and one-point crossover operators has contributed to its popularity. In general, some of the most praised characteristics of PBIL are: its easy implementation [7,15] in comparison to more complex algorithms, and its robustness [12]. Moreover, PBIL has been applied to a variety of real-world problems including energy applications [12,18,19], automatic control [15], biomedical problems [7,23,31], robotics [25], and other problems [1,8,9,40,10,11]. Also, extensions for continuous problems have been introduced [41,46,11]. Usually, in these works, the authors have proposed PBIL algorithms with some enhancements for solving the target problem, achieving competitive results (e.g., self-adaptive approach [12], multiple-population PBIL [19], hybrid approaches [15], different learning and sampling procedures to guarantee a higher diversity [44], parallel schemes [18]).

Some of the findings from the PBIL applications have been: (i) PBIL improves the behavior of simple genetic algorithms [45]; (ii) PBIL algorithms outperform conventional GA approaches [25]; (iii) PBIL can be outperformed by EDAs that use more complex models or tuned search strategies [1,8,23].

Moreover, theoretical analyses of PBIL have been conducted. These papers have focused on the convergence proof of the

* Corresponding author.
E-mail addresses: murilo.zangari@gmail.com (M. Zangari), roberto.santana@ehu.eus (R. Santana), alexander.mendiburu@ehu.eus (A. Mendiburu).

http://dx.doi.org/10.1016/j.asoc.2016.12.045
1568-4946/

algorithm and its expected behavior regarding its ability to find local optima of the function [21,28,38,45].

Notably, and this is the main claim presented in this paper, the numerous reported applications of the discrete PBIL can be split into two main groups according to the way the learning step of the algorithm is interpreted. The critical aspect of this split, which is unveiled in this paper for the first time, is that two essentially different algorithms (in terms of their learning mechanisms) are considered in the current literature as a unique algorithm. This fact has profound implications for the applications of PBIL, the comparison with other methods, and the regarding reproducibility of the results reported for the algorithm. Depending on the choice of PBIL learning inadvently made by the authors, the results of the experiments and the conclusions may be different. Therefore, it is important to throw light on this question and report the existence of these two interpretations. Furthermore, for one of the two PBIL variants that are described in the literature, the algorithm is not well specified, in the sense that the commonly used description of the algorithm allows different implementations and as we show in the paper, from theoretical and empirical perspectives, the changes in the implementations can lead to completely different results. This question is further discussed in Sections 2 and 3.

We show that the differences between the identified PBIL variants are critical in terms of the effect that they can produce in the search process. We conduct an analysis from a theoretical and empirical point of view. On the theoretical side, we provide equations that describe what is the expected form of probabilistic model (mean univariate probabilities) for the two PBIL variants. On the empirical side, we apply the variants for solving a set of well-known benchmark optimization functions. Our analysis has shown that not all PBILs are the same and the differences in the learning mechanism have a major impact on their search ability. The authors of this paper did not find any previous published analysis of the fact the PBIL algorithm has different interpretations regarding its learning mechanism.

The paper is organized as follows: Section 2 introduces PBIL and discusses the variants reported in the literature. In Section 3, we derive formulas for computing the mean univariate probabilities from the two PBIL variants when all possible orders are considered. Section 4 presents the experimental studies using a number of well-known optimization functions. In Section 5, we conclude the paper and discuss topics for future research.

2. PBIL

Let $\mathbf{X} = (X_1, \ldots, X_n)$ denote a vector of discrete n random variables. $\mathbf{x} = (x_1, \ldots, x_n)$ is used to denote an assignment to the variables and $\mathbf{x} \in \{0, 1\}^n$. x_j represents the assignment to the jth variable of the corresponding solution. We work with positive distributions denoted by p. $p(\mathbf{x}_I)$ denotes the marginal probability for $\mathbf{X}_I = \mathbf{x}_I$.

A basic EDA follows these steps: (i) The solutions in population *Pop* are initialized randomly. (ii) A selection scheme is used to select a set of the most promising solutions $S \subset Pop$; an example of selection operator is the truncation selection with truncation parameter $T = 0.5$, where the size of S equals half of the population size of *Pop*. (iii) The univariate probabilistic vector $p(\mathbf{x}) = p(x_1), \ldots, p(x_n)$ is updated using the solutions in S. (iv) Sampling the probabilistic model learned, a new set of candidate solutions S_{new} is created; and then, (v) the current population *Pop* is set with the best solutions from the previous population and the new sampled solutions. The stop condition is usually the maximum number of generations.

Algorithm 1 shows the pseudocode of PBIL, adapted from [2], in which the algorithm was first introduced. The parameter α represents the learning rate, and μ is the probability of mutation occurring in each variable.

Algorithm 1. PBIL (adapted from the original formulation [2])

1 $p(\mathbf{x}) \leftarrow$ initialize each position of the probability vector $p(x_j) = 0.5\ \forall j \in 1, \ldots, n$
2 $Pop \leftarrow$ Generate N solutions randomly
3 For each solution $\mathbf{x}$, compute its fitness function $F(x)$
4 While a termination condition is not met
5 $best \leftarrow$ the solution corresponding to best fitness
6 For each variable, the corresponding entry probability is updated
7 $p(x_j) = (1.0 - \alpha) * p(x_j) + (\alpha * best_j)$
8 Generate a new solution $\mathbf{y}$ sampling from $p(\mathbf{x})$
9 For each variable, if (random(0, 1) < μ) $y_j = 1 - y_j$
10 Add the new sampled solution y to *Pop*
11 End while

One distinguished feature of this PBIL presented in Algorithm 1 is that the update of the univariate vector is done using the best solution found in each generation.

It is also acknowledged in [2] that, when large populations are used, adjusting the prototype vector based upon the single best solution vector in each generation has the potential of ignoring a large amount of the work and exploration performed by the algorithm. The straightforward solution proposed is to move the prototype vector in the direction of the best ($S \ll Pop$) solutions. An enumeration regarding possible ways to implement this variant is presented in [2]. One of the suggested ideas is to move the probability vector equally in the direction of each of the selected solutions. An implementation of this idea is presented in [5].

The main differences in the learning mechanism between the PBIL presented in [5] and the original algorithm are described in the pseudocode of Algorithm 2. Two characteristics of this variant are that the solutions in *Pop* are first sorted according to the evaluation of the solutions (Step 1), and the probabilistic model learned is sensitive to this order (Steps 2 and 3). This dependence on the order is due to the fact that the values of the probabilistic vector are iteratively modified within the loop that passes over all the selected solutions. The last solution vector in the order will have a stronger impact on the final configuration of the probability vector. We call this variant the *order-sensitive* PBIL, or in short PBIL-OS.

Algorithm 2. PBIL (Order-sensitive variant) learning procedure [5]

1 Pop = sort(Pop)
Update Probability Vector towards best solutions
2 For $i = 1$ to NUMBER_OF_SOLUTIONS_TO_UPDATE_FROM (S) do
3 For $j = 1$ to n do
$p(x_j) = (1.0 - \alpha) * p(x_j) + (\alpha * x_j^i)$

Finally, another interpretation of PBIL assumes that, before updating the probability vector, a vector $r(\mathbf{x})$ with the univariate probabilities of the S selected solutions is computed. The auxiliary vector $r(\mathbf{x})$ is then used to update the vector of the univariate probabilities. The pseudocode of this variant is shown in Algorithm 3. We call this variant PBIL-iUMDA, because the strategy used to update the probabilities is the same as that proposed for the iUMDA in [34].

Algorithm 3. PBIL (iUMDA variant) learning procedure [34]

1 # (Compute current probabilities)
2 For $j = 1$ to n do $r(x_j) = \frac{\sum_{i=1}^{S} x_j^i}{S}$
(Update Probability Vector)
3 For $j = 1$ to n do $p(x_j) = (1.0 - \alpha) * p(x_j) + \alpha * r(x_j)$

A considerable amount of work has been published on these two main PBIL variants without recognizing the difference