A Binary Cuckoo Search and Its Application for Feature Selection

Chapter · January 2014

DOI: 10.1007/978-3-319-02141-6_7

7 authors, including:

Caio Ramos
São Paulo State University
30 PUBLICATIONS **231** CITATIONS
SEE PROFILE

André N. DE Souza
São Paulo State University
95 PUBLICATIONS **400** CITATIONS
SEE PROFILE

Xin-She Yang
Middlesex University, UK
412 PUBLICATIONS **16,993** CITATIONS
SEE PROFILE

João Paulo Papa
São Paulo State University
240 PUBLICATIONS **1,871** CITATIONS
SEE PROFILE

A Binary Cuckoo Search and its Application for Feature Selection

L. A. M. Pereira, D. Rodrigues, T. N. S. Almeida, C. C. O. Ramos, A. N. Souza, X.-S. Yang, J. P. Papa

Abstract In classification problems, it is common to find datasets with a large amount of features, some of theses features may be considered as noisy. In this context, one of the most used strategies to deal with this problem is to perform a feature selection process in order to build a subset of features that can better represents the dataset. As feature selection can be modeled as an optimization problem, several studies have to attempted to use nature-inspired optimization techniques due to their large generalization capabilities. In this chapter, we use the Cuckoo Search (CS) algorithm in the context of feature selection tasks. For this purpose, we present a binary version of the Cuckoo Search, namely BCS, as well as we evaluate it with

Luís A. M. Pereira
Department of Computing, UNESP - Univ Estadual Paulista, Bauru, São Paulo, Brazil, e-mail: luis.pereira@fc.unesp.br

Douglas Rodrigues
Department of Computing, UNESP - Univ Estadual Paulista, Bauru, São Paulo, Brazil, e-mail: douglasrodrigues.dr@gmail.com

Tiago. N. S. Almeida
Department of Computing, UNESP - Univ Estadual Paulista, Bauru, São Paulo, Brazil, e-mail: almeida.tns@gmail.com

Caio C. O. Ramos
Department of Electrical Engineering, University of São Paulo, São Paulo, São Paulo, Brazil, e-mail: caioramos@gmail.com

André N. Souza
Department of Electrical Engineering, UNESP - Univ Estadual Paulista, Bauru, São Paulo, Brazil, e-mail: andrejau@feb.unesp.br

Xin-She Yang
School of Science and Technology, Middlesex University, Hendon, London, United Kingdom, e-mail: xy227@cam.ac.uk

João Paulo Papa
Department of Computing, UNESP - Univ Estadual Paulista, Bauru, São Paulo, Brazil, e-mail: papa@fc.unesp.br

different transfer functions that map continuous solutions to binary ones. Additionally, the Optimum-Path Forest classifier accuracy is used as the fitness function. We conducted simulations comparing BCS with binary versions of the Bat Algorithm, Firefly Algorithm and Particle Swarm Optimization. BCS has obtained reasonable results when we consider the compared techniques for feature selection purposes.

Keywords

Feature Selection, Pattern Classification, Meta-heuristic Algorithms, Optimum-Path Forest, Cuckoo Search Algorithm;

1 Introduction

In pattern recognition tasks, many problems are characterized for instances composed of hundreds, thousands or millions of features, as face recognition for instance. As such, working with high dimensional space may demand much computational power and requires long processing time. Therefore, it is often desirable to find a subset that can represent the whole set without losing of information. However, finding this subset is known to be NP-hard and the computational load may become intractable [8].

Support by this previous scenario, several works attempt to model the feature selection as a combinatorial optimization problem, in which the set of features that leads to the best feature space separability is then employed to map the original dataset to a new one. The objective function can be the accuracy of a given classifier or some other criterion that may consider the best trade-off between feature extraction computational burden and effectiveness.

In this fashion, natural inspired optimization techniques have been used to model the feature selection as an optimization problem. The idea is to lead with the search space as a n-cube, where n stands for the number of features. In such case, the optimal (near-optimal) solution is chosen among the 2^n possibilities, and it corresponds to one hypercube's corner. For this purpose, we can employ binary versions of optimization heuristic techniques.

Kennedy and Eberhart [11] proposed a binary version of the well-known Particle Swarm Optimization (PSO) [10] called BPSO, in which the standard PSO algorithm was modified in order to handle binary optimization problems. Further, Firpi and Goodman [5] extended BPSO to the context of feature selection. Rashedi et al. [19] proposed a binary version of the Gravitational Search Algorithm (GSA) [18] called BGSA, which was applied for feature selection by Papa et al. [14]. Ramos et al. [17] presented their version of the Harmony Search (HS) [7] for purpose in the context of theft detection in power distribution system. In addition, Nakamura et al. [13]

proposed a binary version of Bat algorithm (BBA) and Banati and Monika [1] introduced Firefly algorithm for feature selection.

Recently, Yang and Deb [24] proposed a new meta-heuristic method for continuous optimization namely Cuckoo Search (CS), which is based on the fascinating reproduction strategy of cuckoo birds. Several species engage the brood parasitism laying their eggs in the nests of others host birds. Such approach has demonstrated to outperform some well-known nature-inspired optimization techniques, such as PSO and Genetic Algorithms. Furthermore, CS has been applied with success in several distinct engineer applications [6,9,25], and several studies have proposed variants of standard CS to handle particular problems, such as discrete optimizations [12, 22].

In artificial intelligence, CS has also aroused interests specially for machine learning applications. Vazquez [23], for instance, employed CS to train spiking neural models. In addition, Valian [3] proposed a improved Cuckoo Search for feedforward neural network training. For this purpose, the authors investigated the CS behavior by setting its parameters dynamically. Bansal et al. [21] use CS to optimize the local minimal convergence of k-means method for clustering tasks.

In this chapter, we present a binary version of the Cuckoo Search (BCS) for feature selection purposes. The main idea is to associate a set of binary coordinates for each solution that denote whether a feature will belong to the final set of features or not. The function to be maximized is the one given by a supervised classifier's accuracy. As the quality of the solution is related with the number of nests, we need to evaluate each one of them by training a classifier with the selected features encoded by the eggs' quality and also to classify an evaluating set. Thus, we need a fast and robust classifier, since we have one instance of it for each nest. As such, we opted to use the Optimum-Path Forest (OPF) classifier [15,16], which has been demonstrated to be so effective as Support Vector Machines, but faster for training. The experiments have been performed in five public datasets against Bat Algorithm, Firefly Algorithm and Particle Swarm Optimization in order to evaluate the robustness of CS.

The remainder of the chapter is organized as follows. In Section 2 we revisit the Optimum-Path Forest theory. Section 3 presents the Cuckoo Search algorithm and its binary version. Section 4 discuss a framework to evaluate feature selection algorithms based on nature-inspired. Some simulations and its results are shown in Section 5. Finally, conclusions are stated in Section 6.

2 Supervised Classification Through Optimum-Path Forest

The OPF classifier works by modeling the problem of pattern recognition as a graph partition in a given feature space. The nodes are represented by the feature vectors and the edges connect all pairs of them, defining a full connectedness graph. This kind of representation is straightforward, given that the graph does not need to be explicitly represented, allowing us to save memory. The partition of the graph is carried out by a competition process between some key samples (prototypes),

which offer optimum paths to the remaining nodes of the graph. Each prototype sample defines its optimum-path tree (OPT), and the collection of all OPTs defines an optimum-path forest, which gives the name to the classifier [15, 16].

The OPF can be seen as a generalization of the well known Dijkstra's algorithm to compute optimum paths from a source node to the remaining ones [2]. The main difference relies on the fact that OPF uses a set of source nodes (prototypes) with any smooth path-cost function [4]. In case of Dijkstra's algorithm, a function that summed the arc-weights along a path was applied. In regard to the supervised OPF version addressed here, we have used a function that gives the maximum arc-weight along a path, as explained below.

Let $Z = Z_1 \cup Z_2 \cup Z_3$ be a dataset labeled, in which Z_1, Z_2 and Z_3 are, respectively, a training, evaluating and test sets. Let $S \subseteq Z_1$ a set of prototype samples. Essentially, the OPF classifier creates a discrete optimal partition of the feature space such that any sample $s \in Z_2 \cup Z_3$ can be classified according to this partition. This partition is an optimum path forest (OPF) computed in $\Re^n$ by the Image Foresting Transform (IFT) algorithm [4].

The OPF algorithm may be used with any *smooth* path-cost function which can group samples with similar properties [4]. Particularly, we used the path-cost function $f_{\max}$, which is computed as follows:

$$f_{\max}(\langle s \rangle) = \begin{cases} 0 & \text{if } s \in S, \\ +\infty & \text{otherwise} \end{cases}$$
$$f_{\max}(\pi \cdot \langle s,t \rangle) = \max\{f_{\max}(\pi), d(s,t)\}, \quad (1)$$

in which $d(s,t)$ means the distance between samples s and t, and a path π is defined as a sequence of adjacent samples. In such a way, we have that $f_{\max}(\pi)$ computes the maximum distance between adjacent samples in π, when π is not a trivial path.

The OPF algorithm works with a training and a testing phase. In the former step, the competition process begins with the prototypes computation. We are interested into finding the elements that fall on the boundary of the classes with different labels. For that purpose, we can compute a Minimum Spanning Tree (MST) over the original graph and then mark as prototypes the connected elements with different labels. Figure 1b displays the MST with the prototypes at the boundary. After that, we can begin the competition process between prototypes in order to build the optimum-path forest, as displayed in Figure 1c. The classification phase is conducted by taking a sample from the test set (black triangle in Figure 1d) and connecting it to all training samples. The distance to all training nodes are computed and used to weight the edges. Finally, each training node offers to the test sample a cost given by a path-cost function [maximum arc-weight along a path - Equation (1)], and the training node that has offered the minimum path-cost will conquer the test sample. This procedure is shown in Figure 1e.

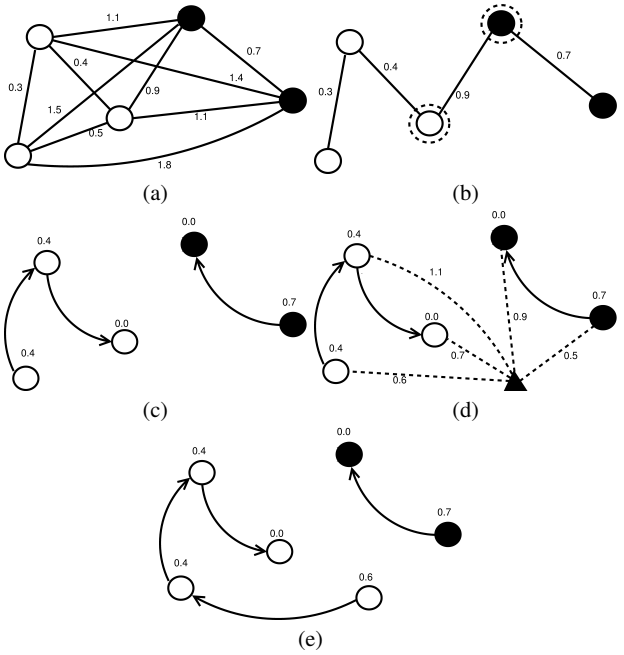


Fig. 1 OPF pipeline: (a) complete graph, (b) MST and prototypes bounded, (c) optimum-path forest generated at the final of training step, (d) classification process and (e) the triangle sample is associated to the white circle class. The values above the nodes are their costs after training, and the values above the edges stand for the distance between their corresponding nodes.

3 Cuckoo Search

3.1 Standard Cuckoo Search

The parasite behavior of some cuckoo species is extremely intriguing. These birds can lay down their eggs in a host nests, and mimic external characteristics of host eggs such as color and spots. In case of this strategy is unsuccessful, the host can throw the cuckoo's egg away, or simply abandon its nest, making a new one in another place. Based on this context, Yang and Deb [24] have developed a novel evolutionary optimization algorithm named as Cuckoo Search (CS), and they have summarized CS using three rules, as follows:

1. Each cuckoo choose a nest randomly to lays eggs.
2. The number of available host nests is fixed, and nests with high quality of eggs will carry over to the next generations.
3. In case of a host bird discovered the cuckoo egg, it can throw the egg away or abandon the nest, and build a completely new nest. There is a fixed number of

host nests, and the probability that an egg laid by a cuckoo is discovered by the host bird is $p_a \in [0,1]$.

CS performs a balanced combination of a local random walk and the global explorative random walk, controlled by a switching parameter $p_a \in [0,1]$. The local random walk can be written as

$$x_i^j(t) = x_i^j(t-1) + \alpha \cdot s \oplus H(p_a - \varepsilon) \oplus (x_{k'}^j(t-1) - x_{k''}^j(t-1)), \tag{2}$$

where $x_{k'}^j$ and $x_{k''}^j$ are two different solutions selected by random permutation, and and x_i^j stands for the j^{th} egg at nest i , $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,d$. $H(\cdot)$ is a Heaviside function, ε is a random number drawn from a uniform distribution, and s is the step size.

The global random walk is carried out using Lévy flights as follows:

$$x_i^j(t) = x_i^j(t-1) + \alpha \cdot L(s,\lambda), \tag{3}$$

where

$$L(s,\lambda) = \frac{\lambda \cdot \Gamma(\lambda) \cdot \sin(\lambda)}{\pi} \cdot \frac{1}{s^{1+\lambda}}, \quad s \gg s_0 > 0 \tag{4}$$

The Lévy flights employ a random step length which is drawn from a Lévy distribution. Therefore, the CS algorithm is more efficient in exploring the search space as its step length is much longer in the long run. The parameter $\alpha > 0$ is the step size scaling factor, which should be related to the scales of the problem of interest. Yang and Deb [26] claim that $\alpha = O(S/10)$ can be used in most cases, where S denotes the scale of the problem of interest, while $\alpha = O(S/100)$ can be more effective and avoid flying too far.

3.2 Binary Cuckoo Search for Feature Selection

In standard CS, the solutions are updated in the search space towards continuous-valued positions. Unlike, in the BCS for feature selection [20], the search space is modelled as a n-dimensional boolean lattice, in which the solutions are updated across the corners of a hypercube. In addition, as the problem is to select or not a given feature, a solution binary vector is employed, where 1 corresponds whether a feature will be selected to compose the new dataset and 0 otherwise. In order to build this binary vector, we have employ the Equation 6, which can provide only binary values in the boolean lattice restricting the new solutions to only binary values:

$$S(x_i^j(t)) = \frac{1}{1 + e^{-x_i^j(t)}} \tag{5}$$

$$x_i^j(t+1) = \begin{cases} 1 & \text{if } S(x_i^j(t)) > \sigma, \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 1: BCS-Feature Selection Algorithm

input : Labeled training set Z_1 and evaluating set Z_2, loss parameter p, α value, number of nests n, dimension d, number of iterations T, c_1 and c_2 values.
output : Global best position $\widehat{g}$.
auxiliaries: Fitness vector f with size m and variables acc, $maxfit$, $globalfit$ and $maxindex$.

for *each nest* n_i $(\forall i = 1,\ldots,m)$ **do**
  for *each dimension* j $(\forall j = 1,\ldots,d)$ **do**
    $x_i^j(0) \leftarrow$ Random$\{0,1\}$;
  end
  $f_i \leftarrow -\infty$;
end
$globalfit \leftarrow -\infty$;
for *each iteration* t $(t = 1,\ldots,T)$ **do**
  for *each nest* n_i $(\forall i = 1,\ldots,m)$ **do**
    Create Z_1' and Z_2' from Z_1 and Z_2, respectively, such that both contains only features in n_i in which $x_i^j(t) \neq 0, \forall j = 1,\ldots,d$;
    Train OPF over Z_1', evaluate its over Z_2' and stores the accuracy in acc;
    if $(acc > f_i)$ **then**
      $f_i \leftarrow acc$;
      for *each dimension* j $(\forall j = 1,\ldots,d)$ **do**
        $\widehat{x}_i^j \leftarrow x_i^j(t)$;
      end
    end
  end
  $[maxfit, maxindex] \leftarrow max(f)$;
  if $(maxfit > globalfit)$ **then**
    $globalfit \leftarrow maxfit$;
    for *each dimension* j $(\forall j = 1,\ldots,d)$ **do**
      $\widehat{g}^j \leftarrow x_{maxindex}^j(t)$;
    end
  end
  for *each nest* n_i $(\forall i = 1,\ldots,m)$ **do**
    for *each dimension* j $(\forall j = 1,\ldots,d)$ **do**
      Select the worst nests with $p_a \in [0,1]$ and replace them for new solutions;
    end
  end
  for *each nest* n_i $(\forall i = 1,\ldots,m)$ **do**
    for *each dimension* j $(\forall j = 1,\ldots,d)$ **do**
      $x_i^j(t) \leftarrow x_i^j(t-1) + \alpha \oplus$ Lévy(λ);
      if $(\sigma < \frac{1}{1+e^{x_i^j(t)}})$ **then**
        $x_i^j(t) \leftarrow 1$;
        else
          $x_i^j(t) \leftarrow 0$;
        end
      end
    end
  end
end

in which $\sigma \sim U(0,1)$ and $x_i^j(t)$ denotes the new egg's value at time step t. Algorithm 1 presents the proposed BCS algorithm for feature selection using the OPF classifier as the objective function.

The algorithm starts with the first loop in Lines $1-4$, which initialize each nest with a vector of binary values (Line 3), and Lines $8-42$ stand to the main algorithm loop. To evaluate each solution, is necessary to build new training Z_1' and and evaluating Z_2' sets. To fulfill this purpose, each sample $s_i \in Z_1$ and $t_i \in Z_2$ is multiplied by a binary solution vector, i.e., $s_i \times n_i \rightarrow Z_1'$ and $t_i \times n_i \rightarrow Z_2'$. Then, Z_1' can be used to generate an OPF training model, which is evaluated using Z_2'. The the classification rate f_i is associated with the nest n_i, and then each nest is evaluated in order to update its fitness value (Lines $12-13$).

Lines $14-15$ find out and store in $\widehat{g}^j$ the best nest with the best so far vector solutions. The loop in the Lines $26-30$ is responsible to replace the nests with the worst solutions using the probability p, generating new nests randomly as described in [25]. Finally, Lines $31-41$ update the binary vector for each nest restricting the generated solutions via Lévy flights [See Eqs. (2) to (4)] and the sigmoid function (Equation 6).

4 Methodology

We now describe the proposed methodology to evaluate the performance of feature selection techniques discussed in previous sections (Figure 2 depicts a pipeline to clarify this procedure). Firstly, we randomly partitioned the dataset into N folds, i.e., $Z = F_1 \cup F_2 \cup \ldots \cup F_N$. Note that each fold should be large enough to contain representative samples of the problem. Further, for each fold, we train a given instance of the OPF classifier over a subset of this fold, $Z_i^1 \in F_i$, and an evaluation set $Z_i^2 \leftarrow F_i \backslash Z_i^1$ is then classified in order to compute a fitness function which will guide a stochastic optimization algorithm to select the most representative set of features. Each member of the population in the meta-heuristic algorithm is associated with a string of bits denoting the presence or absence of a feature. Thus, for each member, we construct a classifier from the training set with only the selected features and compute a fitness function by means of classifying Z_i^2. As long as the procedure converges, i.e, all generations of a population were computed, the agent (bat, firefly, mass, harmony, particle) with the highest fitness value encodes a solution with the best compacted set of features.

Furthermore, we build a classification model using the training set and the selected features, and we also evaluate the quality of the solution computing an effectiveness over the remaining folds, $F_j \in Z \backslash F_i$. Algorithm 2 details the methodology for comparing feature selection techniques.

Figure 2 displays the above procedure. As aforementioned, the feature selection is carried on over the fold i, which is partitioned in a training Z_i^1 and an evaluating set Z_i^2. The idea is to represent a possible subset of features as a string of bits, which

Algorithm 2: Feature Selection Evaluation

input : A dataset Z, number of folds N, number of agents A, number of iterations I, and a percentage for the training set Z_i^1.
output : A predictive performance for each methods defined by a λ function.
auxiliaries: A bitmap vector V of selected features, and a final training and test sets, $\widehat{Z}^1, \widehat{Z}^2$.

for *each fold* $F \in Z$ **do**
  $Z^1 \leftarrow$ random set of $|Z_1| \times |F|$ samples from F;
  $Z^2 \leftarrow F \backslash Z^1$;
  for *each technique* T **do**
    $V \leftarrow$ find a minimal subset of features using T, Z^1, Z^2, and the parameters A, I;
    $\widehat{Z}^1 \leftarrow Z^1 \backslash V$;
    Create a classifier instance from $\widehat{Z}^1$;
    for *each fold* $F' \in Z \backslash F$ **do**
      $\widehat{Z}^2 \leftarrow F' \backslash V$;
      Classify $\widehat{Z}^2$;
      Compute the predictive performance on $\widehat{Z}^2$;
    end
  end
end

encodes each agent's position in the search space. Thus, for each agent, we model the dataset using its string of bits, and an OPF classifier is trained over the new Z_i^1 and its effectiveness using this subset of features is assessed over Z_i^2. This recognition rate is then used as the fitness function to guide each agent to new positions until we reach the convergence criterion. The agent with the best fitness function is then employed to build $\widehat{Z}_i^1$, which is used for OPF training. The final accuracy using the selected subset of features is computed over the remaining folds (red rectangle in Figure 2). This procedure is repeated over all folds for mean accuracy computation.

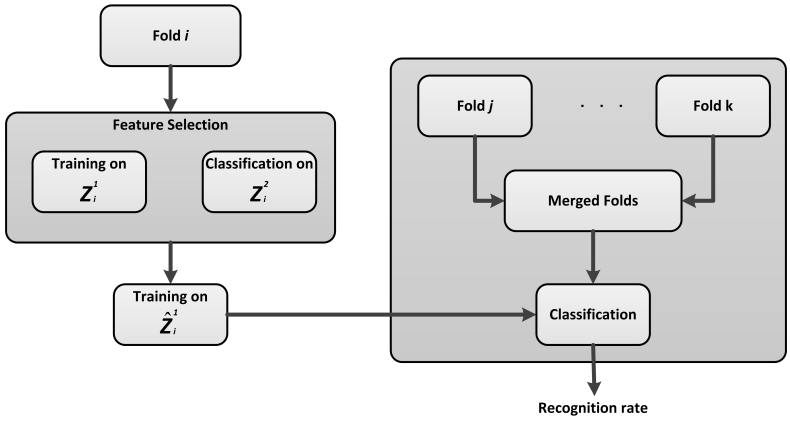


Fig. 2 Flowchart of the proposed methodology.

5 Simulation, Results and Discussion

In this section, we evaluate the robustness of BCS to accomplish the feature selection task, comparing it with the binary versions of Bat Algorithm (BA), Firefly Algorithm (FA) and Particle Swarm Optimization (PSO). We applied the methodology presented in Section 4 to obtain a better quality estimation of each solution. More precisely, we defined $k = 10$ for a cross-validation scheme which implied in ten rounds of feature selection for each method, being the quality of solution evaluated from the remaining nine folds. The performance of those techniques were evaluate over the four public datasets[1], which the main characteristics are presented in Table 1. In addition, regarding the fitness function and the final classification rates, we used an accuracy measure proposed by Papa et al. [16], which considers the fact that classes may have different concentrations in the dataset. This information can avoid a strong estimation bias towards the majority class in high class imbalance datasets.

Table 1 Description of the datasets used for feature selection.

Dataset	# samples	# features	# classes
Diabetes	768	8	2
DNA	2,000	180	3
Heart	270	13	2
Mushrooms	8,124	112	2

We also evaluate how the techniques work with continuous optimization for feature selection purposes, using a sigmoid (Equation 7) and hyperbolic tangent (Equation 7) function to map the continuous values to binary ones, respectively:

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{7}$$

and

$$g(x) = |\tanh(x)|. \tag{8}$$

Table 2 presents the parameters used for each evolutionary-based techniques. It is important to clarify that, for all techniques, we assumed a model with a population size of 10 agents and 100 generations to reach a solution.

Figure 3 displays the accuracy performance of all techniques, as well as the number of selected feature over the four datasets. We can see the optimization techniques presented quite similar performances in both case: binary and continuous optimizations. If we observe only the graph of accuracy rates (Figure 3a, c, e and g), we can infer that feature selection did not improve the classification rates significantly, excepting for the Heart dataset. However, there were considerable feature reductions,

[1] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 2 Parameters setting of optimization techniques.

Technique	Parameters
BA	$\alpha = 0.9$, $\gamma = 0.9$
CS	$\alpha = 1$, $p_a = 0.35$, $\lambda = 1.5$
FA	$\alpha = 0.8$, $\beta = 1$, $\gamma = 0.1$
PSO	$c_1 = 2.0$, $c_2 = 2.0$, w$= 0.7$

specially employing the binary and the continuous optimizations with the sigmoid function.

For Diabetes and Heart datasets, BCS selected the best subset of feature that maximized the OPF accuracy rates (selecting in average six out twelve and five out eight features respectively). The binary PSO was the best on DNA in which it selects the lowest number of feature, around 47%, and maximized the accuracy rate. For the Mushrooms dataset, BA, FA and PSO performed similarly and two percent better than BCS, even if it has the lowest number of selected features.

In regard to continuous optimization, we can observe the hyperbolic tangent did not work well to transfer the continuous values to binary ones. With this function, the techniques had some difficult to reduced the number of features. This behave may due to the threshold value that those function provide. Unlike, the sigmoid function worked well, providing good feature reductions. However, we can infer that binary and continuous optimization with sigmoid function did not present difference indeed.

6 Conclusions

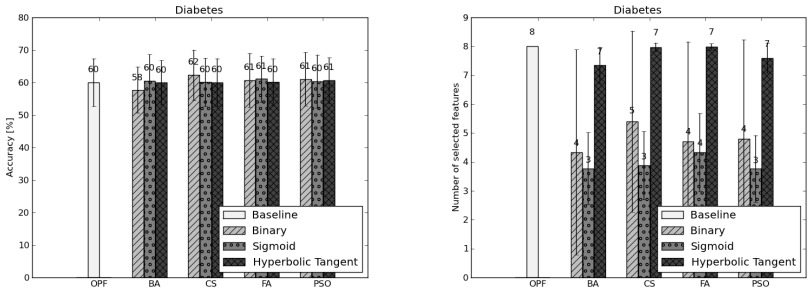
In this chapter, we discuss the feature selection task as an optimization problem. The main purpose is to evaluate the robustness of Cuckoo Search algorithm to accomplish this task. A binary version of Cuckoo Search was presented and compared against with three other nature-inspired optimization techniques. We provided simulations and analysis over four public datasets, employing a cross-validation strategy to verify how the techniques work for feature selection purposes. The results demonstrated that Cuckoo Search has good capabilities to lead with this kind of problem, being the best one on two out of four datasets, and also similarly to other techniques on the remaining datasets.

As the reader may observe, the binary cuckoo version keeps the parameters α and p_a fixed during all iterations. However, these parameters have an important whole regarding to fine-tunning and convergence rates of the standard Cuckoo Search, as stated by Valian [3]. For the future works, it might be interesting to investigate how much the binary Cuckoo Search is sensitive to the aforementioned parameters. Further, we should consider to set the parameters dynamically to improve the performance of Binary Cuckoo Search.

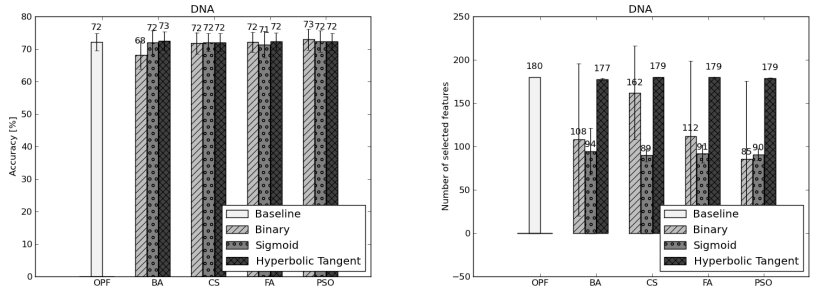
References

1. Banati, H., Bajaj, M.: Fire Fly Based Feature Selection Approach. International Journal of Computer Science Issues **8**(4), 473–480 (2011)
2. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
3. E. Valian, S.M., Tavakoli, S.: On the mean accuracy of statistical pattern recognizers. IInternational Journal of Artificial Intelligence & Applications **2**(3), 36–43 (11)
4. Falcão, A., Stolfi, J., Lotufo, R.: The image foresting transform theory, algorithms, and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(1), 19–29 (2004)
5. Firpi, H.A., Goodman, E.: Swarmed feature selection. In: Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop, pp. 112–118. IEEE Computer Society, Washington, DC, USA (2004)
6. Gandomi, A., Yang, X.S., Alavi, A.: Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Engineering with Computers **29**(1), 17–35 (2013)
7. Geem, Z.W.: Music-Inspired Harmony Search Algorithm: Theory and Applications, 1st edn. Springer Publishing Company, Incorporated (2009)
8. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
9. Kaveh, A., Bakhshpoori, T.: Optimum design of steel frames using cuckoo search algorithm with lvy flights. The Structural Design of Tall and Special Buildings pp. n/a–n/a (2011)
10. Kennedy, J., Eberhart, R.: Swarm Intelligence. M. Kaufman (2001)
11. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 5, pp. 4104–4108 (1997)
12. Layeb, A.: A novel quantum inspired cuckoo search for knapsack problems. Int. J. Bio-Inspired Comput. **3**(5), 297–305 (2011)
13. Nakamura, R.Y.M., Pereira, C.R., Papa, J.P., Falcão, A.: Optimum-path forest pruning parameter estimation through harmony search. In: Proceedings of the 24th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 181–188. IEEE Computer Society, Washington, DC, USA (2011)
14. Papa, J., Pagnin, A., Schellini, S., Spadotto, A., Guido, R., Ponti, M., Chiachia, G., Falcão, A.: Feature selection through gravitational search algorithm. In: Proceedings of the 36th IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 2052–2055 (2011)
15. Papa, J.P., Falcão, A.X., Albuquerque, V.H.C., Tavares, J.M.R.S.: Efficient supervised optimum-path forest classification for large datasets. Pattern Recognition **45**(1), 512–520 (2012)
16. Papa, J.P., Falcão, A.X., Suzuki, C.T.N.: Supervised pattern classification based on optimum-path forest. International Journal of Imaging Systems and Technology **19**(2), 120–131 (2009)
17. Ramos, C., Souza, A., Chiachia, G., Falcão, A., Papa, J.: A novel algorithm for feature selection using harmony search and its application for non-technical losses detection. Computers & Electrical Engineering **37**(6), 886–894 (2011)
18. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: GSA: A gravitational search algorithm. Information Sciences **179**(13), 2232–2248 (2009)
19. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: BGSA: binary gravitational search algorithm. Natural Computing **9**, 727–745 (2010)
20. Rodrigues, D., Pereira, L.A.M., Almeida, T.N.S., Ramos, C.C.O., Souza, A.N., Yang, X.S., Papa, J.P.: BCS: A binary cuckoo search algorithm for feature selection. In: Proceedings of the IEEE International Symposium on Circuits and Systems. Beijing, China (2013)
21. Senthilnath, J., Das, V., Omkar, S., Mani, V.: Clustering using levy flight cuckoo search. In: J.C. Bansal, P. Singh, K. Deep, M. Pant, A. Nagar (eds.) Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012), *Advances in Intelligent Systems and Computing*, vol. 202, pp. 65–75. Springer India (2013)

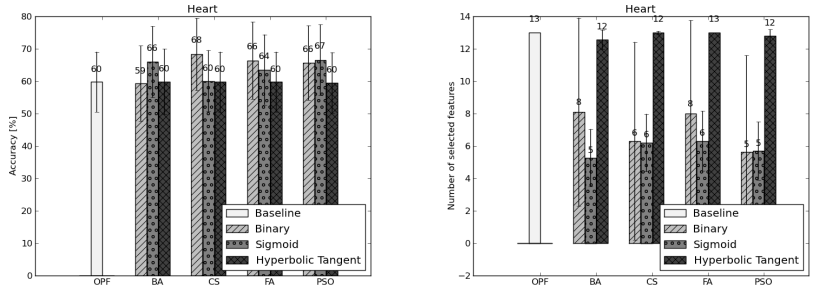
22. Tein, L.H., Ramli, R.: Recent advancements of nurse scheduling models and a potential path. In: Proceedings of 6th IMT-GT conference on mathematics, statistics and its applications (ICMSA 2010) (2010)
23. Vazquez, R.: Training spiking neural models using cuckoo search algorithm. In: Evolutionary Computation (CEC), 2011 IEEE Congress on, pp. 679–686 (2011)
24. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, pp. 210–214 (2009)
25. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation **1**, 330–343 (2010)
26. Yang, X.S., Deb, S.: Cuckoo search: recent advances and applications. Neural Computing and Applications pp. 1–6 (2013)



(a) Diabetes dataset.



(b) DNA-4 dataset.



(c) Heart dataset.

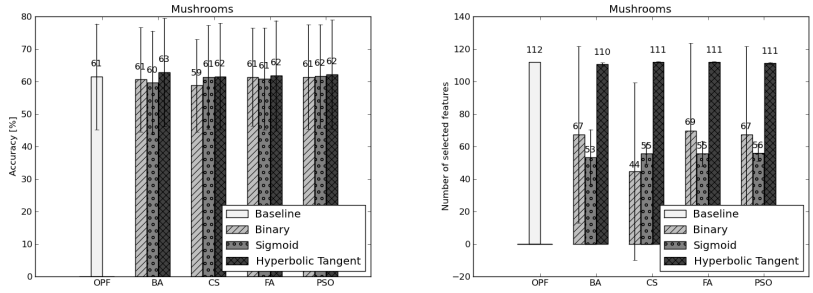


Fig. 3 Experimental results using different transfer functions for each swarm-based optimization technique.