Hindawi
Complexity
Volume 2018, Article ID 8395193, 15 pages
https://doi.org/10.1155/2018/8395193

Research Article

A Binary Cuckoo Search Big Data Algorithm Applied to Large-Scale Crew Scheduling Problems

José García[1], **Francisco Altimiras**[2,3], **Alvaro Peña**[1], **Gino Astorga**[4] and **Oscar Peredo**[2]

[1]*Escuela de Ingeniería en Construcción, Pontificia Universidad Católica de Valparaíso, 2362807 Valparaíso, Chile*
[2]*Telefónica Investigación y Desarrollo, Santiago, Chile*
[3]*Faculty of Engineering and Sciences, Universidad Adolfo Ibañez, Santiago, Chile*
[4]*Universidad de Valparaíso, 2361864 Valparaíso, Chile*

Correspondence should be addressed to José García; joseantonio.garcia@telefonica.com

Received 27 September 2017; Accepted 9 May 2018; Published 30 July 2018

Academic Editor: Laszlo T. Koczy

The progress of metaheuristic techniques, big data, and the Internet of things generates opportunities to performance improvements in complex industrial systems. This article explores the application of Big Data techniques in the implementation of metaheuristic algorithms with the purpose of applying it to decision-making in industrial processes. This exploration intends to evaluate the quality of the results and convergence times of the algorithm under different conditions in the number of solutions and the processing capacity. Under what conditions can we obtain acceptable results in an adequate number of iterations? In this article, we propose a cuckoo search binary algorithm using the MapReduce programming paradigm implemented in the Apache Spark tool. The algorithm is applied to different instances of the crew scheduling problem. The experiments show that the conditions for obtaining suitable results and iterations are specific to each problem and are not always satisfactory.

1. Introduction

With the increase of different kinds of electronic devices, social networks, and the Internet of Things, the datasets are growing fast in volume, variety, and complexity. Currently, big data is emerging as a trend and working with large datasets typically aimed at extracting useful knowledge from them. To address this problem, different programming models have been developed, in which MapReduce is one of the most powerful [1].

In complex industrial systems, the engineers face challenges daily where their job is to make decisions on how to improve the production and reduce costs. They are continuously selecting where, how, when, and what it must do to achieve efficiency in the processes. Normally, these decisions are based on an optimization problem. On the other hand, nowadays, a greater data quantity is available and therefore we can build robust optimization models that support these decisions. However, this increase in data volume and variety implies an increase in the complexity of the calculations and therefore in the convergence time of the algorithms.

Moreover, computational intelligence and particularly metaheuristics have been successful in solving complex industrial problems. In the literature, we find metaheuristics that have satisfactorily solved problems of resource allocation [2, 3], vehicle routing [4], scheduling problems [5], reshuffling operations at maritime container terminals problems [6], antenna positioning problems [7], covering problems [8, 9], and also in bioinformatics problems such as protein structure prediction, molecular docking, and gene expression analysis [10]. However, in the big data era, the integration of metaheuristics into the decision-making process presents two fundamental difficulties: the first one is to get from computational intelligence algorithms, suitable results, and convergence times when dealing with large datasets, because much of the decisions must be close to real time. The second one relates to the programming model differences usually used in computational intelligence and big data algorithms.

These difficulties motivate the design and study of computational intelligence algorithms in programming models used in big data.

A recent framework in the big data area is the Apache Spark which has been widely used to solve industry problems [11]. This framework has advantages over the traditional MapReduce model, since it uses an abstraction called resilient distributed dataset (RDD). This abstraction allows to carry out operations in memory with high fault tolerance, being indicated for the use of iterative algorithms [12]. This work is mainly focused in the behavioural performance analysis of metaheuristic algorithms implemented with the big data Apache Spark tool. The specific objective is the reduction of their convergence times, to support the decision-making in complex industrial systems at the right times. For the design of the experiments, we will use the population size of the metaheuristic and the number of executors within the Apache Spark. To perform the evaluation, the average value, number of iterations, and speed up will be used. The following scenarios will be studied:

(1) The evaluation of the average value through the variation of the solutions number.

(2) The evaluation of iteration number through the solution number used to solve problems.

(3) The evaluation of algorithm scalability through executor number.

These analyses aim to understand which metaheuristic algorithm conditions, related to the solutions and executors number, can obtain suitable results and times to support the decision-making process in complex industrial problems. For this study, it was decided to use the metaheuristic Cuckoo Search; however, the method presented in this article could be applied to different problems of the complex industrial systems.

Cuckoo search is a relatively new metaheuristic that currently has been widely used in solving different types of optimization problems [13]. Some examples of solved problems by the cuckoo search algorithm are the problems in satellite image segmentation [14], the resource allocation problems [3, 15], the optimal power system stabilizers design problems [16], and the optimal allocation of wind based distributed generator problems [17] among others.

In order to carry out the experiments, two types of datasets were chosen. The first one is a benchmark dataset associated to the known set covering problem and a second dataset is associated with the large-scale railway crew scheduling problems, where the number of columns fluctuates between fifty thousand and one million. The results show that adequate scalability and convergence times are not always obtained, what depends on the dataset type and the number of solutions that are being used.

The remainder of this paper is organized as follows. Section 2 briefly introduces the crew scheduling problem. Section 3 details the cuckoo search algorithm. The state of the art of binarization techniques is described in Section 4. In Section 5, we explain the Apache Spark framework. In Sections 6 and 7, we detail the binary and distributed versions of our algorithm. The results of numerical experiments are presented in Section 8. Finally, we provide the conclusions of our work in Section 9.

2. Crew Scheduling Problems

In the crew scheduling problem (CrSP), a group of crew members is assigned to a set of scheduled trips. This allocation must be such that all trips necessarily are covered, while the safety rules and collective agreements must be respected. These allocation and restrictions make the CrSP one of the most difficult problems to solve in the transportation industry [18].

When a bibliographic search is performed, it was found that CrSP is a problem of great importance at present, appearing variations of the original problem associated mainly to the restrictions. As an example, we found CrSP applied to railway. In [19], CrSP with attendance rates was solved; a version of CrSP with fairness preferences was solved in [20]. Crew scheduling problem applications were also found for airlines and bus transportation. In a public transport of buses, in [21] a variation of CrSP was resolved. A new heuristic was proposed in [22] to solve a crew pairing problem with base constraints. In [23], a large-scale integrated fleet assignment and crew pairing problem were solved.

In this work, due to the addition of big data concepts, we will approach the CrSP in its original form. The problem is defined as follows: given a timetable of transport services which are executed every day in a certain period of hours. Each service is divided into a sequence of trips. A trip is performed by a crew, and it is characterized by a departure station, a departure time, an arrival time, and an arrival station. Given a period of time, a crew performs a roster. This is defined as a cyclical travel sequence and each roster assigns a cost.

The CrSP then consists in finding a roster subset that covers all trips, satisfying the constraints imposed and at a minimal cost. The problem is broken down into two phases:

(1) Pairing generation: a very large number of feasible pairings is generated. A pairing is defined as a sequence of trips which can be assigned to a crew in a short working period. A pairing starts and ends in the same depot and is associated with a cost.

(2) Pairing optimization: a selection is made of the best subset of all the generated pairings to guarantee that all the trips are covered at minimum cost. This phase follows quite a general approach, based on the solution of set-covering or set-partitioning problems.

In this research, we will assume that the pair generation phase has already been performed because we will use a benchmark dataset. Therefore, we will focus efforts in resolving the pairing optimization phase. The pairing optimization phase requires the determination of a min-cost subset of the generated pairings covering all the trips and satisfying

Objective function: $f(x)$, $x = (x_1, x_2, \dots, x_n)$
Generate an initial population of m host nests
while (t<MaxGeneration) or (stop criterion)
  Get a cuckoo randomly (say, i) and replace its solution by performing Lévy flights
  Evaluate its fitness F_i
  Choose a nest among n (say, j) randomly
  if $F_i < F_j$ **then**
    Replace j by the new solution
  end if
  a fraction p_a of the worse nests are abandoned and new ones are built
  Keep the best nests
  Rank the nests and find the current best
  Pass the current best solutions to the next generation
end while

Algorithm 1: Cuckoo search algorithm.

additional constraints. Usually it is solved through the set covering problem, and depending on the specific modeled problem, it is added as some type of constraint.

The set covering problem (SCP) is well known to be NP-hard [24]. Nevertheless, different algorithms for solving it have been developed. There exist exact algorithms that generally rely on the branch-and-bound and branch-and-cut methods to obtain optimal solutions [25, 26]. These methods, however, need an effort for solving an SCP instance that grows exponential with the problem size. Then, even medium-sized problem instances often become intractable and cannot be solved anymore using exact algorithms. To overcome this issue, the use of different heuristics has been proposed [27, 28].

For example, [28] presented a number of greedy algorithms based on a Lagrangian relaxation (called the Lagrangian heuristics); Caprara et al. [29] introduced relaxation-based Lagrangian heuristics applied to the SCP. Metaheuristics have also been applied to solve SCP, some examples are genetic algorithms [30], simulated annealing [31], and ant colony optimization [32]. More recently, swarm-based metaheuristics as cat swarm [33], artificial bee colony [34], and black hole [9] were also proposed.

The SCP can be formally defined as follows. Let $A = (a_{ij})$, be a $n \times m$ zero-one matrix, where a column j cover a row i if $a_{ij} = 1$, besides a column j is associated with a nonnegative real cost c_j. Let $I = \{1, \dots, n\}$ and $J = \{1, \dots, m\}$ be the row and column set of A, respectively. The SCP consists in searching a minimum cost subset $S \subset J$ for which every row $i \in I$ is covered by at least one column $j \in J$, that is,

$$\begin{aligned} &\min && f(x) = \sum_{j=1}^{m} c_j x_j \\ &\text{subject to} && \sum_{j=1}^{m} a_{ij} x_j \geq 1, \forall_i \in I, \text{ and } x_j \in \{0, 1\}, \forall_j \in J, \end{aligned} \tag{1}$$

where $x_j = 1$ if $j \in S$, $x_j = 0$ otherwise.

3. Cuckoo Search Algorithm

The cuckoo search is a bioinspired algorithm derived from some cuckoo bird species with an obligate brood parasitism, who lay their eggs in the nests of other bird species [13]. For simplicity, the cuckoo search algorithm is described using the following idealized rules:

(1) Each cuckoo lays one egg at a time and dumps it in a randomly chosen nest.

(2) The best nests with high-quality eggs will be carried over to the next generations.

(3) The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either get rid of the egg or simply abandon the nest and build a completely new nest.

The basic steps of the CS can be recapitulated as the pseudocode shown in Algorithm 1.

The updated cuckoo search solutions are shown in (2), in which γ corresponds to the step size, and $\oplus$ corresponds to the entry-wise multiplications. A random number denominated as Levy (κ) is given by the distribution shown in (3).

$$X_{t+1} = X_t + \gamma \oplus \text{Levy}(\kappa), \tag{2}$$

$$\text{Levy}(\kappa) \sim \mu = t^{-1-\kappa}. \tag{3}$$

The search engine of the cuckoo search algorithm performs naturally in continuous spaces. Nevertheless, the crew scheduling problems are solved in discrete or binary spaces, forcing the adaptation of the original algorithm. A state of the art of main techniques used in the binarization of swarm intelligence continuous metaheuristics is presented in Section 4.

4. Binarization Methods

There exists two main categories for binarization techniques [35]. General binarization frameworks are part of one of

these groups in which exists a mechanism that allows the binary transformation of any continuous metaheuristic without altering the operators. The most used of these frameworks are the transfer functions and the angle modulation. The binarizations designed specifically for a metaheuristic are the second group of binarization methods that include techniques such as the set-based approach and the quantum binarization.

The most used binarization method is the transfer function introduced by [36]. This function is an inexpensive operator that provides the probability values and models the solution positions of the transition. The transfer function is the beginning of the binarization method that allows to map the $\mathbb{R}^n$ solutions in $[0,1]^n$ solutions. The S shaped and the V shaped are the most used transfer functions, well described in [37, 38]. The next step is applying a rule to binarize the transfer function results, which could include the binarization rules elitist, the static probability, the complement, or the roulette [37].

The sizing optimization of the capacitor banks in radial distribution feeders was performed previously using a binary particle swarm optimization [39]. For the reliability analysis of the bulk power system, a transfer function based on swarm intelligence was used [40]. A binary coded firefly algorithm that solves the set covering problem was performed using the same transfer function [37]. A binary cuckoo search algorithm for solving the set covering problem was applied previously [41]. An improved firefly and particle swarm optimization hybrid algorithm was applied to the unit commitment problem [38]. A cryptanalytic attack on the knapsack cryptosystem was approached using the binary firefly algorithm [42]. The network and reliability constrained unit commitment problem was solved using a binary real coded firefly algorithm [43]. Similarly, using the firefly algorithm, the knapsack problem was solved [44].

The angle modulation method uses four parameters which control the frequency and shift of a trigonometric function as is shown in (4).

$$g_i(x_j) = \sin\left(2\pi(x_j - a_i)b_i \cos\left(2\pi(x_j - a_i)c_i\right)\right) + d_i. \quad (4)$$

Using a set of benchmark functions, the angle modulation method was first applied in the particle swarm optimization. Assuming a n-dimensional binary problem and $X = (x_1, x_2, \ldots, x_n)$ as a solution. The first step uses a four-dimensional space, in which each dimension corresponds to a coefficient of (4). The solutions (a_i, b_i, c_i, d_i) are linked to a g_i trigonometric function. The rule 6 is used for each element x_j:

$$b_{ij} = \begin{cases} 1 & \text{if } g_i(x_j) \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Now for each four-dimensional initial solution (a_i, b_i, c_i, d_i), we obtain a feasible n-dimensional solution binarized for our n-binary problem $(b_{i1}, b_{i2}, \ldots, b_{\text{in}})$. Several applications of the angle modulated method have been developed. This include the implementation of angle modulate using a binary PSO to solve network reconfiguration problems [45]. Another implementation is a binary adaptive evolution algorithm applied to multiuser detection in multicarrier cdma wireless broadband system [46]. An angle modulate binary bat algorithm was also previously applied for the mapping of functions when handling binary problems using continuous-variable-based metaheuristics [47].

Evolutionary computing (EC) and quantum computing are two research areas involving the use of three algorithms categories [48]. First, the quantum evolutionary algorithms are focused on the application of EC algorithms in a quantum-computing environment. The evolutionary-designed quantum algorithms are focused in the automatic manufacturing of new quantum algorithms. The quantum-inspired evolutionary algorithms use some concepts and bases of quantum computing to generate new EC algorithms.

(1) Quantum evolutionary algorithms: these algorithms focus on the application of EC algorithms in a quantum-computing environment.

(2) Evolutionary-designed quantum algorithms: these algorithms try to automate the generation of new quantum algorithms using evolutionary algorithms.

(3) Quantum-inspired evolutionary algorithms: these algorithms concentrate on the generation of new EC algorithms using some concepts and principles of quantum computing.

The quantum binary approach is part of this last category, in which the algorithms are adapted to be used on normal computers, integrating the concepts of q-bits and superposition. In this method, each achievable solution has a position $X = (x_1, x_2, .., x_n)$ and the quantum q-bits vector $Q = [Q_1, Q_2, \ldots, Q_n]$. Q stands for the probability of x_j take the value 1. For each dimension j, a random number between [0,1] is obtained and compared with Q_j, if rand $< Q_j$, then $x_j = 1$, else $x_j = 0$. The mechanism of Q vector updating is distinct to each metaheuristic.

The application of quantum swarm optimization has been used in different problems including combinatorial optimization [49], cooperative approach [50], knapsack problem [51], and power quality monitoring in [52]. The application of quantum differential evolution is also observed in the knapsack problem [53], combinatorial problems [54], and methods of image thresholding [55]. A quantum algorithm using cuckoo search metaheuristic was applied to the knapsack problem [56] and bin packing problem [57]. An application to image thresholding using quantum ant colony optimization is reported in [55]. Two quantum binarization applications to the knapsack problem are reported previously using harmony search in [58] and monkey algorithm in [59]. The quantum differential evolution algorithm was applied to the knapsack problem in [53], combinatorial problems [54], and image threshold methods in [55]. Using the cuckoo search metaheuristic, a quantum algorithm was applied to the knapsack problem [56] and bin packing problem [57]. A

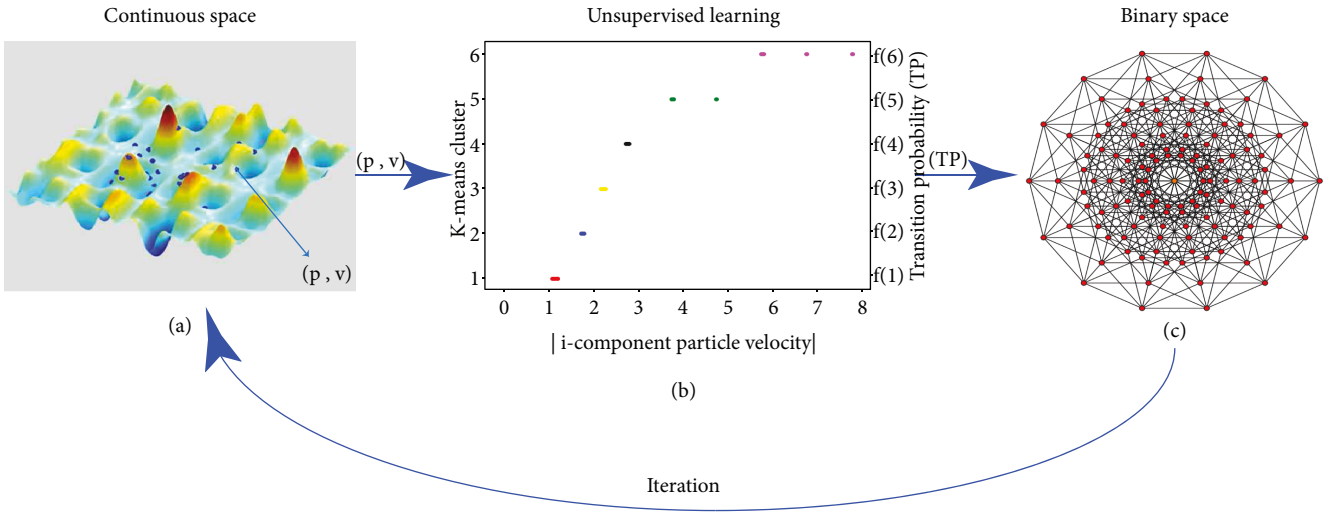

Figure 1: K-means binarization method.

quantum ant colony optimization was applied to image threshold in [55]. Using Harmony Search in [58] and Monkey algorithm in [59], quantum binarizations were applied to the knapsack problem.

The unsupervised learning K-means clustering method is used to perform binarization in different problems as is shown in Figure 1. This method starts with the cuckoo search algorithm generating the pair (p, v) in a continuous space, in which p is the position and v the velocity of the solution (Figure 1(a)). All the velocity module elements are considered, and the K-means is applied (Figure 1(b)). For each cluster k, we link a value $f(k) \in [0, 1]$ of the transition probability (Figure 1(b)). Finally, the transition is performed using the (6). In this equation, $\widehat{x}^i(t)$ corresponds to the complement of $x^i(t) \in 0, 1$. These transitions occur in the binary space (right pannel). If $x^i \in k$ cluster, then $TP(x^i) = f(k)$.

$$x^i(t+1) \coloneqq \begin{cases} \widehat{x}^i(t), & \text{if } \text{rand} < TP\left(x^i\right), \\ x^i(t), & \text{otherwise.} \end{cases} \tag{6}$$

In a previous work, we solved the knapsack problem by applying the transition probability function shown in (7) [3]. In this equation, $\alpha = 0.1$, $\beta = 1$, and $N\left(x^i\right)$ corresponds to the cluster that belongs x^i. P_{tr} corresponds to the transition probability; N can take values between $\{0, \ldots, 4\}$. The initial probability is run by α, and then β carries on the probability jump between the different groups.

$$P_{\text{tr}}\left(x^i\right) = \alpha + \beta N\left(x^i\right)\alpha. \tag{7}$$

5. Spark Distributed Framework

The purpose of this section is to present the Spark distributed framework that it has a target to work with big volumes of data. This framework will be used later in Section 7.

The Spark framework provides an interface of friendly work that allows using of the good way the storage, the memory, and CPU and a set of servers that have as their purpose processing large amounts of data in memory [11]. The requirement of processing large amounts of data is a need that is expressed in the last time, given principally by the low that has shown the cost of data storage which leads to a new need that is to obtain knowledge of this information gathered across the time. This new need arising out of the available storage capacity allowed to find a new line of action for researchers, since the amount, diversity, and complexity of the data [60–64], they are not capable of being tackled by the traditional methods of automatic learning.

Spark has a high performance in parallel computing, being used in machine learning algorithms [65], imaging processing [66], bioinformatics [67], computational intelligence [9], astronomy [68], medical information [69], and so on.

A pioneer in address the treatment of bulk data based on the principle of the locality of the data [70] was MapReduce framework [1] which has the disadvantage of being insufficient for applications that need to share information across several steps or for iterative algorithms [71]. The Spark framework has been very successful becoming a platform for generic use, such as batch processing, iterative process, interactive analysis, flow processing, automatic learning, and computer graphics.

The units of central data of Spark are the resilient distributed datasets (RDD). These units are distributed and are immutable, that is, the transformation of the RDD are RDD and abstraction of memory fault tolerant. Principally, there are two types of operations: transformations that take RDD and produce RDD and actions that take RDD and produce values. To execute Spark, there are several options of administration of cluster that can be used, from the simple independent solutions of Spark, Apache Mesos, and Hadoop YARN [72].

In our case and based on the engineering applications, we decide to use the management Hadoop YARN, being the latest implementation that uses cloud computing [73], that has the characteristic of putting at disposal large number of

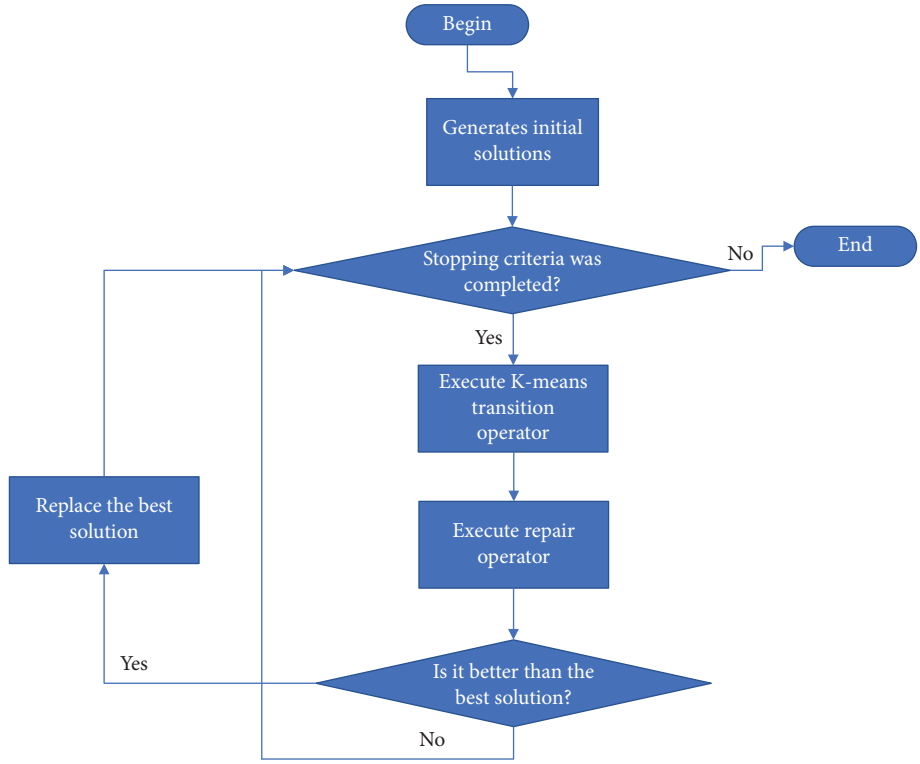

Figure 2: Binary cuckoo search algorithm flow chart.

devices to provide such services as computation and storage on demand that represent a lower cost of hardware, software, and maintenance [73].

6. Binary Cuckoo Search Algorithm

The general performance of the binary cuckoo search algorithm is summarized in this section. First, the algorithm creates the initial solutions with the operator. Once this happens, the algorithm evaluates compliance with the stop criterion. Maximum iteration number and obtaining the optimal value are the two stop criteria. When one of these criteria is not obtained, the K-means transition operator is executed to perform the binarization (detailed in Section 6.2). When the transitions are already obtained, a repair operator must be applied whether the solutions do not accomplish with the problem restrictions (detailed in Section 6.3). This iterative process is evaluated until the stop criterion is accomplished. A general diagram of the process described is detailed in Figure 2.

6.1. Initial Solution Operator. To obtain a new solution, the process begins with the random choice of a column. It is then queried whether the current solution covers all rows. The heuristic operator (Section 6.4) is run to add a new column, until all rows are covered, if the previous part does not happen. The final step is to delete a column, if there are columns that all their rows are covered by more than one column. The initialization process to obtain the solution is detailed in Algorithm 2.

```
1:  Function Initialization()
2:  Input
3:  Output Initialized solution S_out
4:  S ← SelecRandomColumn()
5:  while All row are not covered do
6:      S.append (Heuristic(S))
7:  end while
8:  S ← deleteRepeatedItem(S)
9:  S_out ← S
10:  return S_out
```

Algorithm 2: Initial solution operator.

6.2. K-Means Transition Operator. Cuckoo search is a continuous swarm intelligence metaheuristic. The solutions position at each iteration needs to be updated due its iterative nature. This update is performed in $\mathbb{R}^n$ space when the metaheuristic is continuous. The solution position update can be expressed in a general form for any continuous metaheuristics as is shown in (8). In this equation, $x(t+1)$ corresponds to the x position of the solution at time $t+1$. This position is obtained from the position x at time t plus a Δ function calculated at time $t+1$. The function Δ is due to each metaheuristic and generates values in $\mathbb{R}^n$. In cuckoo

```
Function K-meansTransition(ListX (t))
Input List solutions t (ListX (t))
Output List solution t + 1 (ListX (t + 1))
Δ^i List ← getΔ^i (ListX (t), MH)
X^i Groups ← K-means (Δ^i List, K)
for X(t) in ListX (t)
   for X^i (t) in X(t)
      X^i Groups ← get X^i Groups (i, X(t), X^i Groups)
      P_tr(X^i(t)) ← getTransitionProbability(X^i Group)
      X^i(t + 1) ← applyTransitionRule(P_tr(X^i(t)))
   end for
end for
for X(t + 1) in ListX(t + 1)
   X(t + 1) ← Repair(X(t + 1))
end for
return ListX(t + 1)
```

Algorithm 3: K-means transition algorithm.

search, for example, $\Delta(x(t)) = \gamma \oplus \text{Levy}(\kappa)(x)$, in black hole $\Delta(x(t)) = \text{rand} \times (x_{bh}(t) - x(t))$ and in the firefly, bat, and PSO algorithms, Δ can be expressed in simplified form as $\Delta(x(t)) = v(x(t))$.

$$x(t+1) = x(t) + \Delta(x(t)) \tag{8}$$

The movements generated by the cuckoo search algorithm in each dimension for all solutions are considered in the K-means transition operator. $\Delta^i(x(t))$ is the displacement magnitude of the $\Delta(x(t))$ in the ith position for the solution x at time t. Using $abs(\Delta^i(\text{x}(t)))$, the magnitude of the displacement, the displacements are subsequently grouped. The K-means method is used to do this, where K represents the number of clusters used. In the final step, a generic P_{tr} function given in (9) is proposed to assign a transition probability. In this case, $\mathbb{Z}/k\mathbb{Z}$ is the group obtained when quotient $\mathbb{Z}$ by $k\mathbb{Z}$, that is to say, $\mathbb{Z}/k\mathbb{Z} = \{0, 1, 2, \ldots, k-1\}$, where each element of the group identifies each of the clusters. Since $P_{tr}(i)$ is a probability, it take values in [0,1].

$$P_{\text{tr}} : \frac{\mathbb{Z}}{k}\mathbb{Z} \to [0, 1]. \tag{9}$$

Through the function P_{tr}, a transition probability is assigned to each group. We use the linear function given in (10) as a first approximation. In this equation, $N(x^i)$ corresponds to the location of the group to which $\Delta^i(x)$ belongs. The coefficient α allows defining the transition probability value for all the clusters. This increases proportional to α. For our particular case, $N(x^i) = 0$ corresponds to elements belonging to the group that has the lowest Δ^i values and therefore smaller transition probabilities will be assigned to them.

$$P_{\text{tr}}\left(x^i\right) = P_{\text{tr}}\left(N\left(x^i\right)\right) = \alpha + N\left(x^i\right)\alpha. \tag{10}$$

```
Function Repair(S_in)
Input Input solution S_in
Output The Repair solution S_out
S ← S_in
while needRepair(S) == True do
   S.append (Heuristic(S))
end while
S ← repeatedItem(S)
S_out ← S
return S_out
```

Algorithm 4: Repair algorithm.

```
Function Heuristic(S_in)
Input Input solution S_in
Output The new column C_out
listRows ← getBestRows(S_in, N =10)
listcolumsnOut ← getBestColumns(ListRows, M =5)
columnOut ← getColumn(listcolumnsOut)
return columnOut
```

Algorithm 5: Heuristic operator.

The K-means transition operator begins with the calculation for each solution of the Δ^i (Algorithm 3). The solutions are then grouped using K-means clusterization and the Δ^i as magnitude of distance. We obtain the transition probability with the group assigned to each solution using (10). Subsequently, the transition of each solution is performed. The rule 12 for the cuckoo search is used to perform the transition, where $\widehat{x}^i$ is the complement of x^i. In the final step, each solution is composed using the repair operator detailed in Algorithm 4.

$$x^i(t+1) := \begin{cases} \widehat{x}^i(t), & \text{if } \text{rand} < P_{\text{tr}}\left(x^i\right), \\ x^i(t), & \text{otherwise.} \end{cases} \tag{11}$$

6.3. Repair Operator. Using the K-means transition and the perturbation operators, the repair operator objective is to repair the solutions generated. The operator to perform the repairing process has as input parameter the solution S_{in} to repair and as output parameter the repaired solution S_{out}. We iteratively use the heuristic operator for the execution of the process, which specify the column that must be added. Once all the rows are covered, the deletion is applied to the columns that have all their rows covered by other columns.

6.4. Heuristic Operator. To repair the solutions that do not comply with the constraints is used the heuristic operator. The heuristic operator aims to select a new column for the cases that a solution needs to be built or repaired. The operator considers as input parameter the solution S_{in}

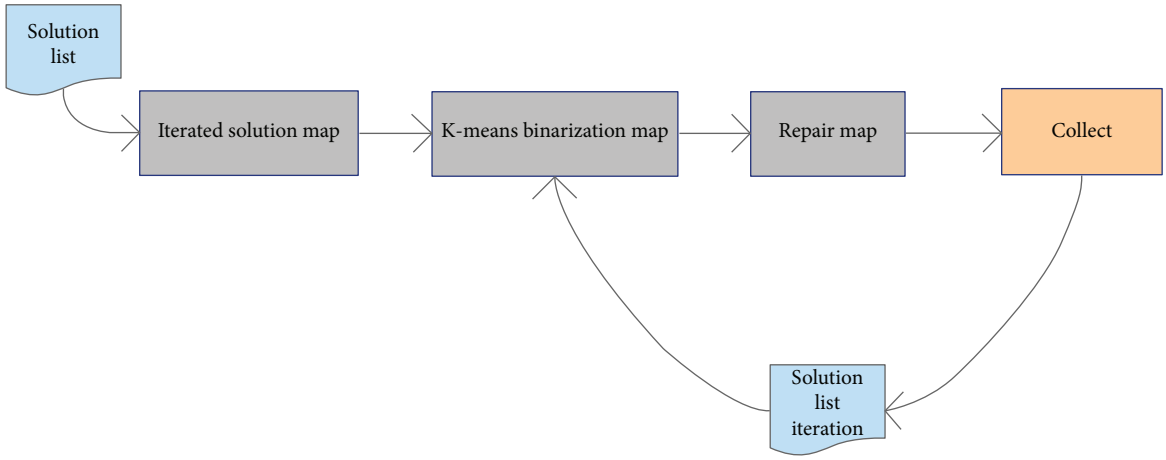

Figure 3: Flow chart of Spark binary cuckoo search algorithm.

```
1: Function: distribuitedBinaryCuckoo(lSol)
2: Input: List of solution (lSol)
3: Output: Iterated list of solution (lSol)
4: lSol ← lSol.map (lambda Solution: (idS, iteratedSolution(Sol))
5: lSol ← lSol.map (lambda Solution: (idS, K-meansTransition(Sol))
6: lSol ← lSol.map (lambda Solution: (idS, repair(Sol))
7: lSol ← lSol.collect()
8: return lSol
```

Algorithm 6: Distributed operator.

Table 1: SCP instances from Beasley's OR-Library.

Instance set	n	m	Cost range	Density	Optimal solution
4	200	1000	[1100]	2	Known
5	200	2000	[1100]	2	Known
6	200	1000	[1100]	5	Known
A	300	3000	[1100]	2	Known
B	200	1000	[1100]	5	Known
C	400	4000	[1100]	2	Known
D	400	4000	[1100]	5	Known
E	500	5000	[1100]	10	Known
F	500	5000	[1100]	20	Unknown
G	1000	10000	[1100]	2	Unknown
H	1000	10000	[1100]	5	Unknown

which needs to be completed, and in the case of being a new solution $S_{\text{in}} = \varnothing$. With the list of columns belonging to S_{in}, you get the set of rows R not covered by the solution. With the set of rows not covered and using (12), we obtain in line 4 the best 10 rows to be covered. With this list of rows (*listRows*) on line 5, we obtain the list of the best columns according to the heuristic indicated in (13). Finally, as a random process, we obtain in line 6 the column to incorporate.

$$\text{WeightRow}(i) = \frac{1}{L_i}, \tag{12}$$

Where L_i is the sum of all ones in row i,

$$\text{WeightColumn}(j) = \frac{c_j}{\left|R \cap M_j\right|}, \tag{13}$$

Where M_j is the set of rows covered by Col j.

7. Binary Cuckoo Search Big Data Algorithm

In this section, we describe the distributed version of the algorithm developed with Apache Spark. The key in each of the map transformations and collect actions used corresponds to the solution identifier that will be denoted by *idS*. When the identifier is used as a key during the execution, it allows the calculations associated to a solution to be executed always in the same partition for the different stages and therefore to be more efficient regarding the data transfer between different workers. In Figure 3, the flow diagram for the distributed algorithm is shown, and in Algorithm 6, the pseudo-code of an iteration is detailed.

Table 2: Parameter setting for cuckoo search big data algorithm.

Parameters	Description	Value	Range
α	Transition probability coefficient	0.1	[0.08, 0.1, 0.12]
K	Number of transition groups or clusters	5	[4–6]
γ	Step length	0.01	[0.009, 0.01, 0.011]
κ	Levy distribution parameter	1.5	[1.4, 1.5, 1.6]
Iteration number	Maximum iterations	700	[600, 700, 800]

Table 3: Average result by problem type of dataset OR-Library.

Instance	Best known	BCSBA (5)	BCSBA (10)	BCSBA (20)	BCSBA (50)	BCSBA (100)	BCSBA (500)
4	510	514.1	513.3	511.6	510.9	510.9	510.9
5	257.3	259.5	258.8	258.1	257.6	257.6	257.6
6	144.2	145.1	144.9	144.7	144.5	144.5	144.5
A	241.4	243.4	243.2	243.0	242.9	242.8	242.8
B	75.2	75.4	75.2	75.2	75.2	75.2	75.2
C	224.6	227.7	226.3	226.1	225.6	225.6	225.6
D	64.2	65.3	65.1	64.9	64.7	64.7	64.6
E	28.4	28.8	28.6	28.5	28.5	28.5	28.5
F	14	14.5	14.4	14.2	14.2	14.2	14.2
G	166.4	173.2	172.4	171.6	168.1	168.1	168.0
H	59.4	64.9	63.4	63.1	60.7	60.8	60.7
Average	162.28	164.72	163.87	163.73	162.99	162.99	162.96
p value			2.1e-4	1.5e-5	3.5e-7	2.7e-7	3.1e-08

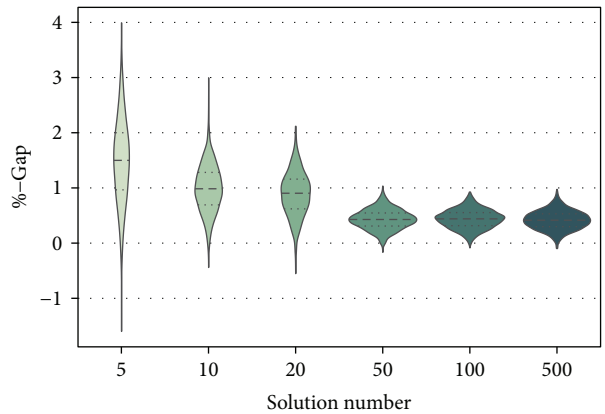

Figure 4: Violin chart of the results quality by solution number.

LSol contains the solution list to be iterated with the cuckoo search algorithm. Each of these solutions has the position and velocity information. The first step is to iterate the solutions using the cuckoo search algorithm; this is done at line 4. The key corresponds to the *idS* particle identifier, and the returned value corresponds to the iterated solution *Sol*, in which the velocity values have been updated. The next step is to perform an iteration of the positions. For this, the K-means transition operator described in Section 6.2 and executed at line 5 of Algorithm 6 is used. With the K-means transition operator, the velocities obtained in the previous step are used to get the new binary values of the solution position. Subsequently since there is a possibility that the iterated solutions do not meet with the constraints, a repair operator is applied. This operator acts on the positions and updates them to fulfill with the constraints. The detail of the repair algorithm is described in Section 6.3. Finally, the list of solutions is collected and stored for further analysis.

8. Results

In this section, we present computational experiments with the proposed Spark binary cuckoo search algorithm. We test the algorithm on two classes of well-known problems.

(1) OR-Library benchmarks: this class includes 65 small and medium size randomly generated problems that were frequently used in the literature. They are available in the OR-Library and are described in Table 1.

(2) Railway scheduling problems: this class includes seven large-scale railway crew scheduling problems from Italian railways and are available in OR-Library.

Binary cuckoo search big data algorithm was implemented in python using Spark libraries. It was executed in Azure platform, Spark 1.6.1 and Hadoop 2.4.1 versions. To perform the statistical analysis in this study, the Wilcoxon signed-rank nonparametric test was used. For the results, each problem was executed 30 times.

The first stage corresponds to perform the parameter configuration used by the algorithm. To develop this activity,

Table 4: Average result by problem type for railway scheduling dataset.

Instance	Best known	BCSBA (5)	BCSBA (10)	BCSBA (20)	BCSBA (50)	BCSBA (100)	BCSBA (500)
Rail507	174	192.1	190.3	187.2	184.5	182.4	182.3
Rail516	182	189.4	187.2	183.2	183.2	182.4	182.4
Rail582	211	227.4	226.1	224.4	223.3	221.2	221.4
Rail2586	948	1152.3	1142.2	1140.1	1132.2	1130.8	1130.6
Rail2536	691	836.1	832.4	830.9	826.2	822.1	822.3
Average	441.2	519.46	515.64	513.16	509.88	507.58	507.6

Table 5: Average iteration by problem type.

Instance	BCSBA (5)	BCSBA (10)	BCSBA (20)	BCSBA (50)	BCSBA (100)	BCSBA (500)
4	82.2	80.1	80.6	76.3	74.9	76.9
5	81.7	76.4	76.4	77.1	70.6	71.4
6	88.6	87.8	86.4	84.7	82.5	82.3
A	112.5	109.1	108.2	106.3	97.8	98.8
B	105.6	107.5	100.2	101.1	99.2	96.6
C	130.5	128.3	120.1	110.8	108.4	105.4
D	134.7	132.1	130.9	125.4	109.9	107.6
E	138.1	131.6	130.5	124.5	109.5	112.4
F	145.6	136.4	135.2	115.6	108.4	107.6
G	297.3	276.4	271.6	254.1	196.5	194.6
H	267.1	256.6	245.1	221.2	160.7	161.8
Average	143.99	138.39	135.02	127.01	110.76	110.49

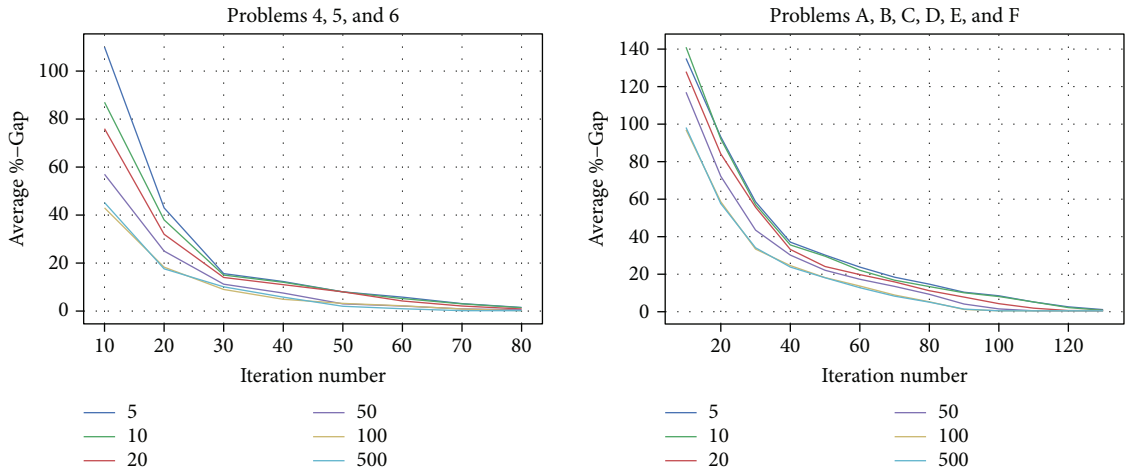

Figure 5: Convergence charts for instances of small and medium size problems.

the methodology described in [3] was used. In this methodology, four standard measures are used: the worst case, the best case, the average case, and the average execution time. With these four measurements, the area under the radar chart curve is obtained to define the best configuration. The dataset used to determine the best configuration corresponds to the first problem of each group $\{4.1, 5.1, \ldots, G.1, H.1\}$. The results are shown in Table 2. In this table, the range column corresponds to the evaluated ranges and the value column to the value that will be used. The value of the parameters γ, κ, and iteration number corresponds to those frequently used by the cuckoo search algorithm in the literature. The parameters α and K are specific to the K-means binarization method and are referenced to (10).

8.1. Evaluation of Result Quality through the Variation of the Solution Number. The goal of this section is to evaluate the number of solutions to be used by the binary cuckoo search big data algorithm (BCSBA) with respect to the quality of the results. For the execution of this experiment, the other

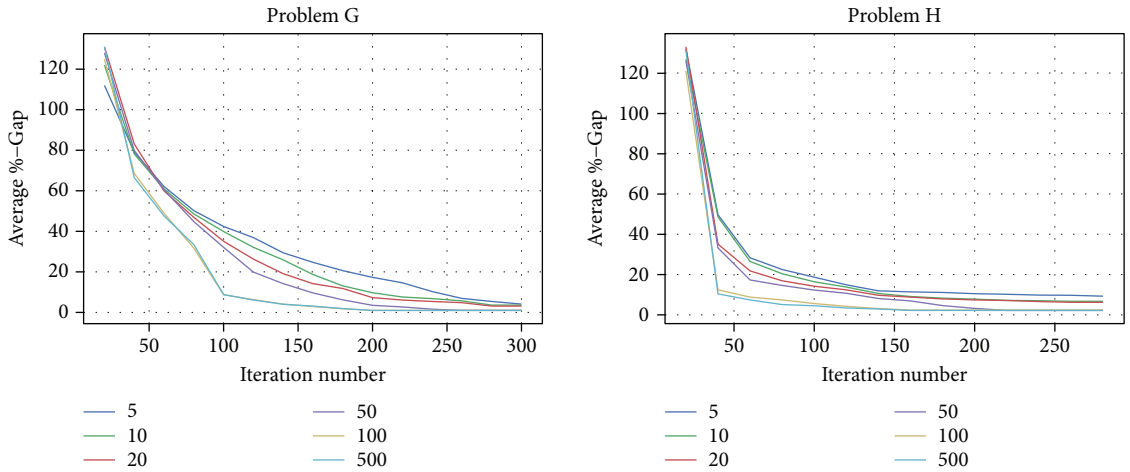

Figure 6: Convergence charts for instances of big size problems.

Table 6: Spark configuration.

Num-executors	Executor-cores	Executor-memory (Gb)
1	3	4
2	3	4
4	3	4
8	3	4
16	3	4

parameters used by the algorithm were the values described in the value column of the Table 2. In Table 3, the results are shown for cases that consider 5, 10, 20, 50, 100, and 500 solutions using the OR-Library dataset. From the table we observe that the results for cases 50, 100, and 500 are superior to the rest, nevertheless between them, they are very similar. Additionally, to see the significance, the Wilcoxon test was performed, comparing CSBA(5) with respect to other cases, obtaining that in all cases there is a significant difference. To complement the above analysis, violin charts were used to compare the distributions of the results through their shapes and interquartile ranges. The results are shown in Figure 4. The x-axis corresponds to the number of solutions used to solve the problem and the y-axis to %−Gap defined in (14). In the distributions, the superiority of the cases 50, 100, and 500 over the rest is appreciated. When we compare the cases 50, 100, and 500, between them, we see there is a similarity in the shape of their distributions as well as in the interquartile ranges.

$$\%-\text{Gap} = 100\frac{\text{SolutionValue} - \text{BestKnown}}{\text{BestKnown}}. \quad (14)$$

In Table 4, the results for the railway scheduling problems are displayed. In this table, a behavior similar to the previous analysis is observed. The cases in which it uses 100 and 500 solutions obtained better results than the other cases. When comparing BCSBA-100 with BCSBA-500, similar results are observed.

8.2. Evaluation of Algorithm Convergence Time through the Solution Number. In this section, the convergence of the BCSBA algorithm with respect to the number of solutions is evaluated. For this analysis, the problems were grouped into 4 groups: the small group, which considers problems 4, 5, and 6; the median group, which considers problems A, B, C, D, E, and F; problem group G; and problem group H. Table 5 and Figures 5 and 6 show the results for different groups. In the table, it is observed that BCSBA has better convergence in cases 100 and 500 than in the rest of the cases, the result being very similar between 100 and 500. In Figures 5 and 6 the x-axis corresponds to the number of average iterations and the y-axis is the average of the %−Gap defined in (14). The data was collected every 10 iterations in the small and medium groups and every 20 iterations in the G and H groups. For the case of the small and medium groups, although the convergence curves are better in cases 100 and 500, the difference is quite small, which does not justify the increase in the number of solutions. For the case of the groups G and H, this difference becomes much more notorious.

8.3. Evaluation of Algorithm Scalability through Core Number. This last experiment aims to evaluate the scalability of our algorithm when considering more than one core for calculation. In Section 8.1 and Section 8.2, we see that increasing the number of solutions improves the results and decreases the number of iterations. However, the increase in the number of solutions has a computation cost. In this section, we evaluate whether the cost of computing can be diminished by the use of more processing cores.

In this section, we evaluate whether the cost of computing can be diminished by the use of more processing cores.

For the Spark configuration, three parameters were considered: num-executors which controls the number of executor requested, executor-cores property which controls the number of concurrent tasks an executor can run, and executor-memory which corresponds to the memory per executor. For the proper use of an executor, it is recommended to use between 3 and 5 cores. The considered Spark settings are shown in Table 6.

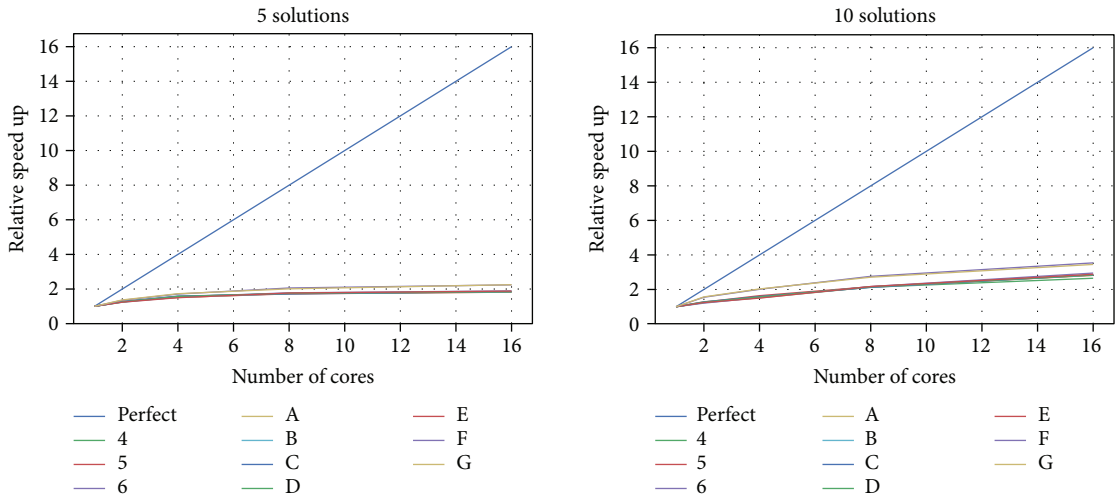

Figure 7: Speed up charts for 5 and 10 solutions.

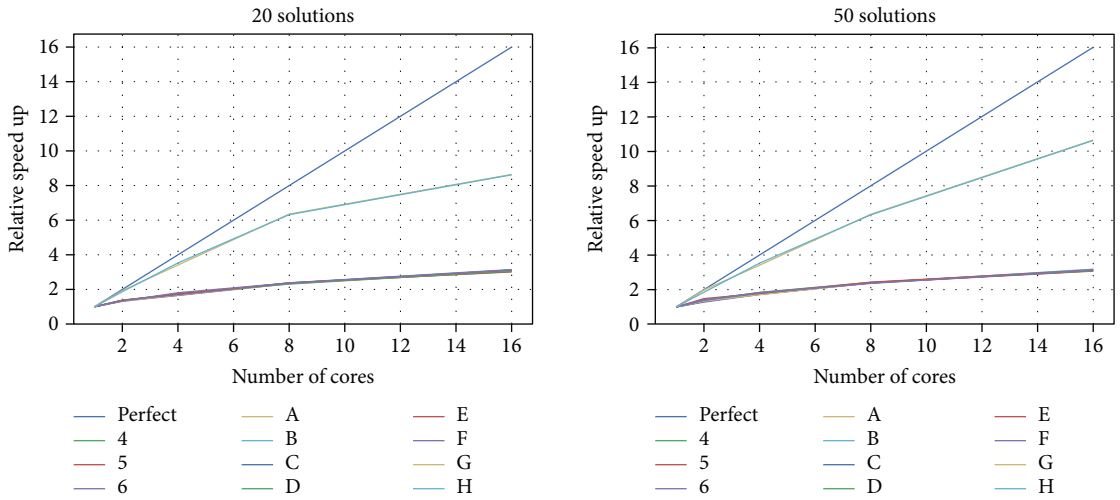

Figure 8: Speed up charts for 20 and 50 solutions.

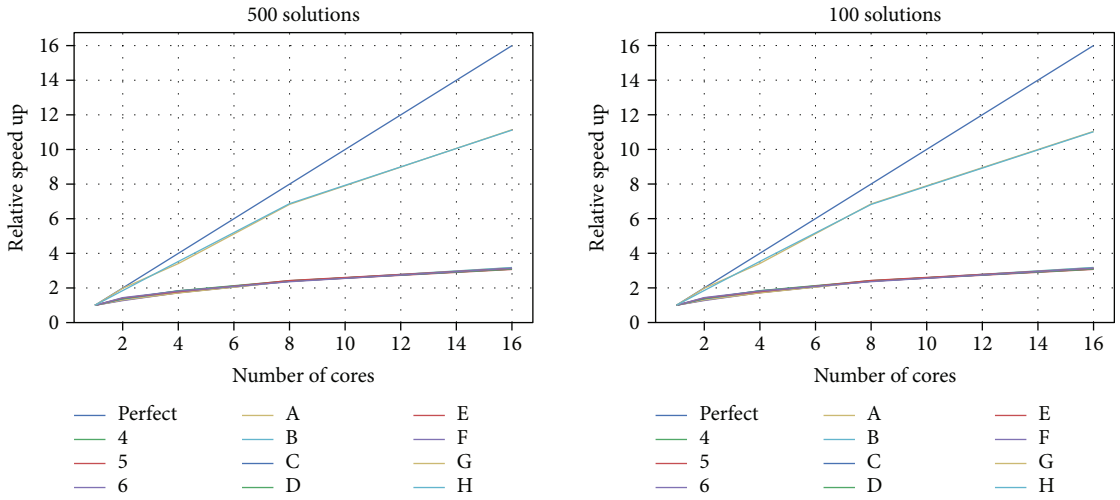

Figure 9: Speed up charts for 100 and 500 solutions.

In Figures 7, 8, and 9 we show the results of speed up charts for BCSBA using different numbers of solutions and considering between 1 and 16 executors. From the charts, it is observed that the best scalability is obtained for the case of 100 particles and in the problems G and H. For smaller problems, scalability is significantly reduced. The worst scalability was obtained for the algorithm using 5 particles. Another interesting fact is observed in the 500-particle chart where scalability was superior in G and H problems than in the rest; however, the performance is lower than in the case of 100 particles.

9. Conclusions

In this work, we have presented a binary cuckoo search big data algorithm applied to different instances of crew scheduling problem. We used an unsupervised learning method based on the K-means technique to perform binarization. Later, to develop the distributed version of the algorithm, Apache Spark was used as framework. The quality, convergence, and scalability of the results were evaluated in terms of the number of solutions used by the algorithm. It was found that quality, convergence, and scalability are affected by the number of solutions; however, these depend additionally on the problem that is being solved. In particular, it is observed that for medium size problems, the effects are not very relevant as opposed the large problems such as G and H, where the effect of the number of solutions is much more significant. On the other hand, when evaluating the scalability, we observe that it is also dependent on the number of solutions used by the algorithm and the size of the problems. The best performances were for problems G and H considering solutions between 20 and 500.

As a future work, it is interesting to investigate the proposed algorithm with other NP-hard problems with the intention of observing similar behaviours to observe in the case of CrSP. Also, we want to investigate how is the performance of autonomous search tuning algorithms [74] in big data environment. Finally, we also want to explore the performance of other metaheuristics in Big data Frameworks.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] J. Garca, B. Crawford, R. Soto, and G. Astorga, "A percentile transition ranking algorithm applied to binarization of continuous swarm intelligence metaheuristics," in *Recent Advances on Soft Computing and Data Mining. SCDM 2018. Advances in Intelligent Systems and Computing, Vol 700*, R. Ghazali, M. Deris, N. Nawi, and J. Abawajy, Eds., pp. 3–13, Springer, Cham, 2018.

[3] J. García, B. Crawford, R. Soto, C. Castro, and F. Paredes, "A k-means binarization framework applied to multidimensional knapsack problem," *Applied Intelligence*, vol. 48, no. 2, pp. 357–380, 2018.

[4] A. Franceschetti, E. Demir, D. Honhon, T. Van Woensel, G. Laporte, and M. Stobbe, "A metaheuristic for the time-dependent pollution-routing problem," *European Journal of Operational Research*, vol. 259, no. 3, pp. 972–991, 2017.

[5] H. E. Nouri, O. B. Driss, and K. Ghédira, "Hybrid metaheuristics for scheduling of machines and transport robots in job shop environment," *Applied Intelligence*, vol. 45, no. 3, pp. 808–828, 2016.

[6] R. Jovanovic, M. Tuba, and S. Voß, "A multi-heuristic approach for solving the pre-marshalling problem," *Central European Journal of Operations Research*, vol. 25, no. 1, pp. 1–28, 2017.

[7] S. Basbug, "A general model for pattern synthesis of linear antenna arrays by metaheuristic algorithms," in *2017 International Applied Computational Electromagnetics Society Symposium - Italy (ACES)*, pp. 1-2, Florence, Italy, March 2017.

[8] B. Crawford, R. Soto, E. Monfroy, G. Astorga, J. Garca, and E. Cortes, "A meta-optimization approach for covering problems in facility location," in *Applied Computer Sciences in Engineering. WEA 2017. Communications in Computer and Information Science*, J. Figueroa-García, E. López-Santana, J. Villa-Ramírez, and R. Ferro-Escobar, Eds., pp. 565–578, Springer, Cham, 2017.

[9] J. Garca, B. Crawford, R. Soto, and P. Garca, "A multi dynamic binary black hole algorithm applied to set covering problem," in *Harmony Search Algorithm. ICHSA 2017. Advances in Intelligent Systems and Computing, Vol 514*, J. Ser, Ed., pp. 42–51, Springer, Singapore, 2017.

[10] S. Santander-Jimenez and M. A. Vega-Rodriguez, "Asynchronous non-generational model to parallelize metaheuristics: a bioinformatics case study," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1825–1838, 2017.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceeding HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, p. 10, Boston, MA, USA, June 2010.

[12] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceeding NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, p. 2, San Jose, CA, USA, April 2012.

[13] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 210–214, Coimbatore, India, December 2009.

[14] A. K. Bhandari, V. K. Singh, A. Kumar, and G. K. Singh, "Cuckoo search algorithm and wind driven optimization based study of satellite image segmentation for multilevel thresholding using kapur's entropy," *Expert Systems with Applications*, vol. 41, no. 7, pp. 3538–3560, 2014.

[15] J. Garca, B. Crawford, R. Soto, and G. Astorga, "A percentile transition ranking algorithm applied to knapsack problem," in *Applied Computational Intelligence and Mathematical Methods. CoMeSySo 2017. Advances in Intelligent Systems and Computing, Vol 662*, R. Silhavy, P. Silhavy, and Z. Prokopova, Eds., pp. 126–138, Springer, Cham, 2017.

[16] S. M. Abd Elazim and E. S. Ali, "Optimal power system stabilizers design via cuckoo search algorithm," *International Journal of Electrical Power & Energy Systems*, vol. 75, pp. 99–107, 2016.

[17] S. Sudabattula and M. Kowsalya, "Optimal allocation of wind based distributed generators in distribution system using cuckoo search algorithm," *Procedia Computer Science*, vol. 92, pp. 298–304, 2016.

[18] A. Kasirzadeh, M. Saddoune, and F. Soumis, "Airline crew scheduling: models, algorithms, and data sets," *EURO Journal on Transportation and Logistics*, vol. 6, no. 2, pp. 111–137, 2017.

[19] K. Hoffmann, U. Buscher, J. S. Neufeld, and F. Tamke, "Solving practical railway crew scheduling problems with attendance rates," *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 147–159, 2017.

[20] S. Jütte, D. Müller, and U. W. Thonemann, "Optimizing railway crew schedules with fairness preferences," *Journal of Scheduling*, vol. 20, no. 1, pp. 43–55, 2017.

[21] H. Öztop, U. Eliiyi, D. T. Eliiyi, and L. Kandiller, "A bus crew scheduling problem with eligibility constraints and time limitations," *Transportation Research Procedia*, vol. 22, pp. 222–231, 2017.

[22] F. Quesnel, G. Desaulniers, and F. Soumis, "A new heuristic branching scheme for the crew pairing problem with base constraints," *Computers & Operations Research*, vol. 80, pp. 159–172, 2017.

[23] O. Ö. Özener, M. Örmeci Matoğlu, G. Erdoğan, M. Haouari, and H. Sözer, "Solving a large-scale integrated fleet assignment and crew pairing problem," *Annals of Operations Research*, vol. 253, no. 1, pp. 477–500, 2017.

[24] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[25] E. Balas and M. C. Carrera, "A dynamic subgradient-based branch-and-bound procedure for set covering," *Operations Research*, vol. 44, no. 6, pp. 875–890, 1996.

[26] J. E. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, no. 1, pp. 85–93, 1987.

[27] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *European Journal of Operational Research*, vol. 94, no. 2, pp. 392–404, 1996.

[28] J. E. Beasley, "A lagrangian heuristic for set-covering problems," *Naval Research Logistics*, vol. 37, no. 1, pp. 151–164, 1990.

[29] A. Caprara, M. Fischetti, and P. Toth, "A heuristic method for the set covering problem," *Operations Research*, vol. 47, no. 5, pp. 730–743, 1999.

[30] B. Yelbay, Ş. İlker Birbil, and K. Bülbül, "The set covering problem revisited: an empirical study of the value of dual information," *European Journal of Operational Research*, vol. 11, no. 2, pp. 575–594, 2015.

[31] M. J. Brusco, L. W. Jacobs, and G. M. Thompson, "A morphing procedure to supplement a simulated annealing heuristic for cost-andcoverage-correlated set-covering problems," *Annals of Operations Research*, vol. 86, pp. 611–627, 1999.

[32] C. Valenzuela, B. Crawford, R. Soto, E. Monfroy, and F. Paredes, "A 2-level metaheuristic for the set covering problem," *International Journal of Computers Communications & Control*, vol. 7, no. 2, pp. 377–387, 2014.

[33] B. Crawford, R. Soto, N. Berríos et al., "A binary cat swarm optimization algorithm for the non-unicost set covering problem," *Mathematical Problems in Engineering*, vol. 2015, Article ID 578541, 8 pages, 2015.

[34] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Tech. Rep. tr 06, Erciyes university, engineering faculty, computer engineering department, 2005.

[35] B. Crawford, R. Soto, G. Astorga, J. García, C. Castro, and F. Paredes, "Putting continuous metaheuristics to work in binary search spaces," *Complexity*, vol. 2017, Article ID 8404231, 19 pages, 2017.

[36] R. C. Eberhart and J. Kennedy, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, pp. 4104–4108, Orlando, FL, USA, October 1997.

[37] B. Crawford, M. Olivares-Suarez, W. Palma, F. Paredes, E. Olguin, and E. Norero, "A binary coded firefly algorithm that solves the set covering problem," *Romanian Journal of Information Science and Technology*, vol. 17, no. 3, pp. 252–264, 2014.

[38] Y. Yang, Y. Mao, P. Yang, and Y. Jiang, "The unit commitment problem based on an improved firefly and particle swarm optimization hybrid algorithm," in *2013 Chinese Automation Congress*, pp. 718–722, Changsha, China, November 2013.

[39] M. M. A. Aziz, T. M. Khalil, and H. K. M. Youseef, "A binary particle swarm optimization for optimal placement and sizing of capacitor banks in radial distribution feeders with distorted substation voltages," in *Proceedings of AIML International Conference*, pp. 137–143, Sharm El-Sheikh, Egypt, 2006.

[40] D. G. Robinson, "Reliability analysis of bulk power systems using swarm intelligence," in *Annual Reliability and Maintainability Symposium, 2005. Proceedings*, pp. 96–102, Alexandria, VA, USA, January 2005.

[41] J. Barraza, R. Soto, B. Crawford, R. Olivares, F. Johnson, and F. Paredes, "A binary cuckoo search algorithm for solving the set covering problem," in *Bioinspired Computation in Artificial Systems. IWINAC 2015. Lecture Notes in Computer Science, Vol 9108*, J. Ferrández Vicente, J. Álvarez-Sánchez, F. Paz López, F. Toledo-Moreo, and H. Adeli, Eds., pp. 88–97, Springer, Cham, 2015.

[42] S. Palit, S. N. Sinha, M. A. Molla, A. Khanra, and M. Kule, "A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm," in *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, pp. 428–432, Allahabad, India, September 2011.

[43] K. Chandrasekaran and S. P. Simon, "Network and reliability constrained unit commitment problem using binary real coded firefly algorithm," *International Journal of Electrical Power & Energy Systems*, vol. 43, no. 1, pp. 921–932, 2012.

[44] X. Zhang, C. Wu, J. Li et al., "Binary artificial algae algorithm for multidimensional knapsack problems," *Applied Soft Computing*, vol. 43, pp. 583–595, 2016.

[45] W. Liu, L. Li, and D. A. Cartes, "Angle modulated particle swarm optimization based defensive islanding of large scale power systems," in *2007 IEEE Power Engineering Society Conference and Exposition in Africa – PowerAfrica*, pp. 1–8, Johannesburg, South Africa, July 2007.

[46] S. Das, R. Mukherjee, R. Kundu, and T. Vasilakos, "Multi-user detection in multi-carrier cdma wireless broadband system using a binary adaptive differential evolution algorithm," in *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, pp. 1245–1252, Amsterdam, The Netherlands, July2013.

[47] Z. Dahi, C. Mezioud, A. Draa et al., *Computer Science and Its Applications. CIIA 2015. IFIP International Conference on Computer Science and Its Applications*, A. Amine, L. Bellatreche, Z. Elberrichi, E. Neuhold, and R. Wrembel, Eds., Springer, Cham, 2015.

[48] G. Zhang, "Quantum-inspired evolutionary algorithms: a survey and empirical study," *Journal of Heuristics*, vol. 17, no. 3, pp. 303–351, 2011.

[49] Y. Zhou, I. Wang, and Y. Zhang, "Discrete quantum-behaved particle swarm optimization based on estimation of distribution for combinatorial optimization," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 897–904, Hong Kong, China, June 2008.

[50] J. Zhao, J. Sun, and W. Xu, "A binary quantum-behaved particle swarm optimization algorithm with cooperative approach," *International Journal of Computer Science*, vol. 10, no. 2, pp. 112–118, 2005.

[51] J. Licheng, Y. Shuyuan, and W. Min, "A quantum particle swarm optimization," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, pp. 19–23, Portland, OR, USA, June 2004.

[52] A. A. Ibrahim, A. Mohamed, H. Shareef, and S. P. Ghoshal, "An effective power quality monitor placement method utilizing quantum-inspired particle swarm optimization," in *2011 International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 1–6, Bandung, Indonesia, July 2011.

[53] A. R. Hota and A. Pat, "An adaptive quantum-inspired differential evolution algorithm for 0-1 knapsack problem," in *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 703–708, Fukuoka, Japan, December 2010.

[54] J. Alegra and Y. Túpac, "A generalized quantum-inspired evolutionary algorithm for combinatorial optimization problems," in *XXXII International Conference of the Chilean Computer Science Society SCCC*, pp. 11–15, Temuco, Chile, November 2014.

[55] S. Dey, S. Bhattacharyya, and U. Maulik, "New quantum inspired meta-heuristic techniques for multi-level colour image thresholding," *Applied Soft Computing*, vol. 46, pp. 677–702, 2016.

[56] A. Layeb, "A novel quantum inspired cuckoo search for knapsack problems," *International Journal of Bio-Inspired Computation*, vol. 3, no. 5, pp. 297–305, 2011.

[57] A. Layeb and S. R. Boussalia, "A novel quantum inspired cuckoo search algorithm for bin packing problem," *International Journal of Information Technology and Computer Science*, vol. 4, no. 5, pp. 58–67, 2012.

[58] A. Layeb, "A hybrid quantum inspired harmony search algorithm for 0-1 optimization problems," *Journal of Computational and Applied Mathematics*, vol. 253, pp. 14–25, 2013.

[59] Y. Zhou, X. Chen, and G. Zhou, "An improved monkey algorithm for a 0-1 knapsack problem," *Applied Soft Computing*, vol. 38, pp. 817–830, 2016.

[60] J. García, C. Pope, and F. Altimiras, "A distributed *K*-means segmentation algorithm applied to *Lobesia botrana* recognition," *Complexity*, vol. 2017, Article ID 5137317, 14 pages, 2017.

[61] E. Graells-Garrido and J. Garca, "Visual exploration of urban dynamics using mobile data," in *Ubiquitous Computing and Ambient Intelligence. Sensing, Processing, and Using Environmental Information. UCAmI 2015. Lecture Notes in Computer Science, Vol 9454*, J. García-Chamizo, G. Fortino, and S. Ochoa, Eds., pp. 480–491, Springer, Cham, 2015.

[62] E. Graells-Garrido, O. Peredo, and J. García, "Sensing urban patterns with antenna mappings: the case of Santiago, Chile," *Sensors*, vol. 16, no. 7, p. 1098, 2016.

[63] M. Minelli, M. Chambers, and A. Dhiraj, *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for today's Businesses*, John Wiley & Sons, 2012.

[64] O. F. Peredo, J. A. Garca, R. Stuven, and J. M. Ortiz, "Urban dynamic estimation using mobile phone logs and locally varying anisotropy," in *Geostatistics Valencia 2016. Quantitative Geology and Geostatistics, Vol 19*, J. Gómez-Hernández, J. Rodrigo-Ilarri, M. Rodrigo-Clavero, E. Cassiraga, and J. Vargas-Guzmán, Eds., pp. 949–964, Springer, Cham, 2017.

[65] R. Shyam, H. B. Bharathi Ganesh, S. Sachin Kumar, P. Poornachandran, and K. P. Soman, "Apache Spark a big data analytics platform for smart grid," *Procedia Technology*, vol. 21, pp. 171–178, 2015.

[66] S. Sarraf and M. Ostadhashem, "Big data application in functional magnetic resonance imaging using apache spark," in *2016 Future Technologies Conference (FTC)*, pp. 281–284, San Francisco, CA, USA, December 2016.

[67] W. Zhou, R. Li, S. Yuan et al., "Metaspark: a spark-based distributed processing tool to recruit metagenomic reads to reference genomes," *Bioinformatics*, vol. 33, no. 7, pp. 1090–1092, 2017.

[68] Z. Zhang, K. Barbary, F. A. Nothaft et al., "Kira: processing astronomy imagery using big data technology," *IEEE Transactions on Big Data*, no. 1, p. 1, 2016.

[69] L. Hao, S. Jiang, B. Si, and B. Bai, "Design of the research platform for medical information analysis and data mining," in *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1985–1989, Datong, China, October 2016.

[70] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in MapReduce," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pp. 419–426, Ottawa, ON, Canada, May 2012.

[71] K. Grolinger, M. Hayes, W. A. Higashino, A. L'Heureux, D. S. Allison, and M. A. M. Capretz, "Challenges for MapReduce in Big Data," in *2014 IEEE World Congress on Services*, pp. 182–189, Anchorage, AK, USA, June–July 2014.

[72] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*, O'Reilly Media, Inc., 2015.

[73] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology - EDBT/ICDT '11*, pp. 530–533, Uppsala, Sweden, March 2011.

[74] Y. Hamadi, E. Monfroy, and F. Saubion, "What is autonomous search?," in *Hybrid Optimization. Springer Optimization and Its Applications, Vol 45*, P. Hentenryck and M. Milano, Eds., pp. 357–391, Springer, New York, NY, USA, 2011.