applied sciences

MDPI

Article

Resource Scheduling in Cloud Computing Based on a Hybridized Whale Optimization Algorithm

Ivana Strumberger, Nebojsa Bacanin *, Milan Tuba * and Eva Tuba

Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11010 Belgrade, Serbia; istrumberger@singidunum.ac.rs (I.S.); etuba@ieee.org (E.T.)

* Correspondence: nbacanin@singidunum.ac.rs (N.B.); tuba@ieee.org (M.T.); Tel.: +381-653093-223 (M.T.)

Received: 19 October 2019; Accepted: 6 November 2019; Published: 14 November 2019

Abstract: The cloud computing paradigm, as a novel computing resources delivery platform, has significantly impacted society with the concept of on-demand resource utilization through virtualization technology. Virtualization enables the usage of available physical resources in a way that multiple end-users can share the same underlying hardware infrastructure. In cloud computing, due to the expectations of clients, as well as on the providers side, many challenges exist. One of the most important nondeterministic polynomial time (NP) hard challenges in cloud computing is resource scheduling, due to its critical impact on the cloud system performance. Previously conducted research from this domain has shown that metaheuristics can substantially improve cloud system performance if they are used as scheduling algorithms. This paper introduces a hybridized whale optimization algorithm, that falls into the category of swarm intelligence metaheuristics, adapted for tackling the resource scheduling problem in cloud environments. To more precisely evaluate performance of the proposed approach, original whale optimization was also adapted for resource scheduling. Considering the two most important mechanisms of any swarm intelligence algorithm (exploitation and exploration), where the efficiency of a swarm algorithm depends heavily on their adjusted balance, the original whale optimization algorithm was enhanced by addressing its weaknesses of inappropriate exploitation–exploration trade-off adjustments and the premature convergence. The proposed hybrid algorithm was first tested on a standard set of bound-constrained benchmarks with the goal to more accurately evaluate its performance. After, simulations were performed using two different resource scheduling models in cloud computing with real, as well as with artificial data sets. Simulations were performed on the robust CloudSim platform. A hybrid whale optimization algorithm was compared with other state-of-the-art metaheurisitcs and heuristics, as well as with the original whale optimization for all conducted experiments. Achieved results in all simulations indicate that the proposed hybrid whale optimization algorithm, on average, outperforms the original version, as well as other heuristics and metaheuristics. By using the proposed algorithm, improvements in tackling the resource scheduling issue in cloud computing have been established, as well enhancements to the original whale optimization implementation.

Keywords: cloud computing; resource scheduling; metaheuristics; swarm intelligence; whale optimization algorithm; hybridization

1. Introduction

One of the most important benefits of cloud computing is on-demand provisioning of requested services and resources over high speed computer networks. With permanent growth and advancements of network technologies and infrastructure, cloud computing has demonstrated superior performance in serving different kinds of large-scale and complex end-users' (clients') tasks. Heterogeneous clients' requirements are supported by sophisticated development platforms and

applications, along with state-of-the-art physical and virtualized hardware. At the current stage of technology advancements, cloud computing and the internet of things (IoT) have become essential concepts that set new goals for industry innovations [1].

One of the most important challenges in the cloud computing domain is resource scheduling. When performing resource scheduling, at least a satisfying level of quality of service (QoS) should be maintained by utilizing proper hardware infrastructure and algorithms. A component of the cloud infrastructure, usually referred as the broker in modern literature, is responsible for mapping requested end-users' tasks to the available virtualized hardware that is in most cases implemented in a form of virtual machines (VMs). The broker performs mapping by executing the scheduling algorithm. With the growth of the number of submitted tasks and the number of available resources, it becomes extremely difficult to map tasks to the appropriate VMs for execution. If an inappropriate scheduling algorithm is used, some VMs may be over-utilized or under-utilized, and the implication of such scenarios is performance degradation of the cloud system as a whole. The resource scheduling problem belongs to the group of NP (nondeterministic polynomial time) hard optimization problems.

It should be noted that in the recent computer science literature, the terms task and cloudlet scheduling are also adopted for the process of mapping submitted end-users' task to the available VMs.

Many algorithms and techniques for resource scheduling in cloud computing environments are available. For example, in some systems, classical (deterministic) algorithms are used. However, classical optimization approaches are not efficient due to the fact that deterministic algorithms are not capable of generating satisfying, nor optimal or near optimal solutions within a reasonable computational time for NP hard challenges. Due to the search space complexity and exponential number of possible solutions, classical approaches are not able to evaluate every potential solution from the search domain in polynomial time.

When tackling NP hard tasks, such as cloud computing resource scheduling, instead of using classical optimization techniques, methods that evaluate only promising parts of the search space (not the whole search domain) by using a smart mechanism when choosing which solutions to evaluate next, should be utilized. One of the most efficient ways to tackle the resource scheduling problem is the implementation of heuristics and metaheuristics based approaches that do not guarantee finding an optimal solution, but in practice, they proved able to generate satisfying solutions within the polynomial time.

According to the literature survey, metaheuristics, especially those that are inspired by nature (nature-inspired, bio-inspired), have proven to be robust approaches that can efficiently tackle the cloud computing resource scheduling issue [2–4].

One class of the most well-known metaheuristics that are inspired by natural phenomena is swarm intelligence. Swarm intelligence approaches are population-based, stochastic and iterative. Many real-life problems, such as the localization problem in wireless sensor networks (WSNs) [5,6], drone placement [7], robot path planning [8,9], the network planning problem in radio frequency identification (RFID) networks [10], machine learning optimization [11], image processing [12], computer aided diagnostic systems [13–15], portfolio optimization [16], and more have been successfully tackled by utilizing swarm intelligence. In addition to numerous applications, swarm intelligence algorithms have been constantly modified, hybridized [17–19] and parallelized [20,21] with the aim of achieving the best possible results.

1.1. Objectives, Contributions, Question and Methodology of the Proposed Research

In this paper we propose an implementation of the enhanced whale optimization algorithm (WOA) adapted for tackling the resource scheduling challenge in cloud computing environments. We improved the basic WOA approach by performing hybridization with the artificial bee colony (ABC) and firefly algorithms (FA), which are also categorized as swarm intelligence metaheuristics.

The WOA was created by Mirjalili and Lewis in 2016 [22] and shortly after many modified and improved WOA versions emerged [23–25]. Some WOA adaptations for resource scheduling in cloud

computing can be found in the literature survey [26]. However, a hybrid between the WOA and ABC and FA metaheuristics, as is proposed in this paper, has never been implemented and tested before on any problem.

The main motivation behind the proposed research is the fact that swarm intelligence metaheuristics proved to be robust and effective optimization methods for solving many different types of real-environment NP hard optimization problems [27,28], as well as the fact that swarm intelligence algorithms have already been successfully applied to various models (single and multi-objective) of resource scheduling in cloud computing [2,29,30].

The main objective of the research proposed in this paper is to try to establish further enhancements in tackling the resource scheduling challenge in cloud computing environments by using swarm intelligence approaches. Guided by the main objective, we have implemented a hybridized WOA metaheuristic and have validated its performance for two resource scheduling models. The first model is single-objective, and aims to minimize the makespan indicator, while the second model belongs to the group of multi-objective optimization, where makespan and total cost objectives were taken into consideration.

The secondary objective of the proposed research is an attempt to address observed deficiencies of the original WOA implementation by performing hybridization with other state-of-the-art swarm intelligence algorithms. For this purpose, as well as for the goal of establishing more accurate evaluation of the proposed hybrid WOA's performance, we first tested our devised approach on a larger set of bound-constrained benchmarks and compared the obtained solutions' quality and convergence speed with the results of the original WOA implementation against the same test instances.

It has to be noted that the authors have conducted research with improved and hybridized swarm intelligence algorithms before [31,32], and that they have also implemented some swarm intelligence approaches for resource scheduling tasks in cloud computing environment [4,33]. Hence, the research presented in this paper is the result of authors previous experience in this domain, as well as their recent work with WOA metaheuristic and resource scheduling problems in cloud computing.

According to the research objectives, the basic research question addressed in this paper is: "Is it achievable to establish further improvements in solving the cloud computing resource scheduling problem by using swarm intelligence algorithms?" The second research question, that falls into domain of bio-inspired metaheuristics, can be formulated as follows: "Is it possible to improve performance of original WOA approach by performing hybridization with other swarm algorithms that proved to be efficient optimization methods?"

In the presented research, a simulation in the standard environment with classic benchmark instances was utilized as the research methodology. Three types of simulations were performed. The first simulation was conducted for standard unconstrained benchmark problems, while the second and the third were performed by using two different resource scheduling models in cloud computing with real (second simulation), as well as with artificial (third simulation) data sets.

The main contributions of the proposed research can be differentiated into two groups: enhancements of resource scheduling in cloud computing and improvements by hybridization of the WOA metaheuristic.

Resource scheduling simulations were conducted in a robust and adaptable CloudSim framework environment. All the necessary details and inner workings of the proposed research, including the settings of the algorithms' control parameters, simulation framework settings and utilized data sets, are fully provided in this paper, hence the researchers who want to implement proposed approaches and to run simulations have more than enough information to do this on their own.

1.2. Paper's Composition

The proposed paper is organized as follows. After the Introduction, Section 2 presents the cloud computing environment and describes the concept of resource scheduling with all the details that are necessary for understanding the conducted research. The whole section is devoted to cloud

computing as its principles and architecture are enabling efficient utilization of available resources over the communication network with the objective of satisfying clients' requests. The need for resource scheduling and provisioning increases with the rise of submitted requests. The notion of resource scheduling becomes an important aspect of the cloud computing paradigm, where the goal is to execute all the tasks within the least possible time interval. Section 2 also addresses the applications of swarm intelligence algorithms in cloud scheduling where many examples can be found in the literature.

The mathematical formulation of the cloud computing resource scheduling models, that were employed in our simulations, is given in Section 3. The single-objective scheduling model that was used in simulations with the real data set is described, followed by the multi-objective model that was utilized with the artificial data set. In this section, basic terminology and performance metrics that are necessary for understanding the conducted simulations are also introduced.

The original, as well as the proposed hybrid WOA metaheuristics, are described in Section 4. In this section the original WOA approach is presented and its deficiencies that were noticed during the empirical simulations, are addressed from the theoretical, as well as from the practical standpoint. Finally, the proposed hybrid WOA, which incorporates certain carefully chosen components from the ABC and FA swarm intelligence metaheuristics, is shown.

Section 5 is the experimental section, where obtained simulation results of the proposed hybrid algorithm are presented. As stated above, to validate and to measure improvements of the hybrid WOA over the original WOA, first, tests were performed on a wide set of bound-constrained benchmarks. Side-by-side comparison was conducted between the basic and hybrid approaches. Moreover, in comparative analysis other state-of-the-art swarm algorithms have been included that were validated against the same benchmarks. A major part of this section shows results of conducted simulations for two cloud computing resource scheduling models. As in the case of unconstrained benchmarks, comparative analysis was also performed with the original WOA, as well as with other well-known heuristics and metaheuristics. Most of the simulation results are visualized.

In the final Section 6, conclusions, regarding the performance of our proposed hybrid WOA, that have been derived from the results of conducted simulations, are summarized. Also, future potential research directions from the domain of bio-inspired metaheuristics, as well as from the sphere of cloud computing challenges, are provided.

2. Cloud Computing Paradigm, Scheduling and Literature Review

In this section of the paper, the architecture and basic principles of the cloud computing paradigm are described. Later, the process of cloud computing resource scheduling is presented, along with the most commonly utilized scheduling algorithms. Finally, a brief overview of swarm intelligence applications in the area of cloud computing is provided.

2.1. Cloud Computing Principles and Architecture

The permanent and continual improvements in computing and networking technologies have led to the rapid growth of requirements for accessing the data and software services from an efficient networking environment that provides elasticity, scalability, security, and reliability. This environment that enables the usage of resources with enough processing power to meet all end-user's demands over high-speed broadband networks, is cloud computing. Due to its adaptability and versatility, the cloud computing paradigm has been extensively implemented in many industries, and it has also been adopted by the academic and scientific communities.

Many definitions of cloud computing are available, but one of the most important is provided by the National Institute of Standards and Technology (NIST), which states that "cloud computing can be depicted as a model for enabling ubiquitous and convenient on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort, or service provider interaction" [34]. One other definition defines cloud computing as an "elastic and distributed

system in which computing resources, such as processing power and storage space, information and software, are propagated through the network and delivered in the distributed location in the cloud where, it can be shared and obtained" [35]. The third definition, that we selected from the computer science literature, states that the "cloud represents a collection of heterogeneous hardware and software resources which provides services and fast processing resources with high availability to the cloud users in a way that it is not necessary that the end-users have their own hardware and software infrastructure" [30].

In order to successfully execute submitted tasks, the cloud computing environment is supported by virtualization technology and hyper-converged infrastructure (HCL). Moreover, these two technologies are enabling cloud computing services due to the fact that they make cloud computing feasible. Without virtualization and HCL, cloud computing as a service delivery model would not be economically justified for cloud clients (end-users), as well as for cloud service providers (CSPs).

The concept of virtualization, as an abstraction of computing resources, is used along with the scheduling, where simultaneously submitted diverse tasks are being executed by different VMs, that are disjointed from the physical hardware infrastructure. The basic component of virtualization, which is responsible for controlling and synchronizing VM execution is the hypervisor, also known as the virtual machine monitor (VMM). In cloud environments, type 1 hypervisors are used. The type 1 VMM (bare-metal hypervisor) executes directly above the physical hardware and it is much faster than the type 2 hypervisor (hosted VMM) that runs above the host operating system.

2.2. Resource Scheduling in Cloud Computing

The flexibility of cloud computing services is reflected in the effectiveness of the demanded resources being provided to the end-users over the internet. Cloud computing centralized infrastructure administers various types of tasks, which are sometimes complex and resource consuming. In order to establish the connection between the tasks and the resources that will execute those tasks, with the availability in mind, scheduling techniques and algorithms that enable this link between the resources (VMs) hosted in the cloud data centers and the tasks submitted by the end-users, should be employed.

The notion of resource scheduling becomes an important aspect of the cloud computing paradigm, where the goal is to execute the tasks in the least amount of time and with the least possible resource consumption. More precisely, the goal is to determine the best resource (VM) for executing each of requested tasks. This objective is carried by the scheduling algorithm. The scheduler should help in enhancements of QoS, such as reliability, productivity, resource utilization, energy consumption, cost of the execution, etc. [29].

By using scheduling algorithms, an optimal (or near optimal) allocation of available resources among requested tasks can be obtained in finite time to achieve desired QoS [2]. The goal of scheduling is to build a plan that determines on which resources and when each task will be executed. It should be noted that a scheduling has been for years a current research topic in various domains, for example job shop scheduling, operating system scheduling, etc.

As already stated in Section 1, the terms resource scheduling, task scheduling, and cloudlet scheduling are used interchangeably in the modern literature. The notion of the cloudlet is adopted from the CloudSim framework, where the submitted end-users' tasks are referred to as cloudlets. Also, the term resource refers to the VM.

With resource scheduling certain issues may arise. Arisen issues can be handled with effective provisioning algorithms, which should analyze and organize the upcoming workload in an adequate way. These issues are connected to the workload arrangement, which depends on the types of requests. If a classical pay-as-you-go model is utilized, on-demand scheduling, where the cloud provider enables access to the resources in a reasonable time followed by random workload, is appropriate.

However, online scheduling can cause problems. If distribution of the workload to VMs is unequal, a scenario where some machines are executing more tasks and are exceeding their capacity

and overflowing the schedule time, may emerge. This is known as the issue of load balancing. Efficient load balancing algorithms, that distribute the load evenly on all available VMs should be employed.

Another issue may arise in scenarios where there are more or less available VMs than the real resources' need of submitted tasks. In the first case, where there is less than the needed VMs available, a scenario of under provisioning may occur. In the latter case, there is a scenario of over provisioning. Neither of them are good. If there is under provisioning, clients' needs could not be processed efficiently and that can lead to end-users' dissatisfaction. On the contrary, if there is over provisioning, the costs of resource utilization are unnecessarily increased.

Scheduling algorithms can be divided into static and dynamic algorithms [29,36]. Static scheduling algorithms before execution (upfront) require detailed information regarding the tasks, such as length, number of tasks, and the deadlines for its execution, as well as information regarding the resources (VMs) that should be provided, such as available processing power, memory capacity, energy consumption, etc. Due to the dynamic nature and inconsistencies in the cloud computing environment and the number of requests and resources, the static algorithms are not adequate for this type of system, because they are not able to adjust the distribution of workload between the VMs well. In most cloud computing scenarios, static scheduling algorithms establish lower than average performance.

Moreover, practice has shown that the performance issues with static scheduling algorithms affect the QoS indicators: makespan, reliability, availability, etc. Some examples of static scheduling algorithm instances include heuristics, like round robin (RR), first in first out (FIFO), and shortest job first (SJF).

With the above mentioned in mind, dynamic scheduling algorithms are more appropriate to for implementation in cloud computing infrastructure, as, instead of focusing on detailed information regarding the tasks and resources, their activities are based towards the nodes (VMs) monitoring activities. Dynamic scheduling algorithms are constantly monitoring changes in the cloud environment and shifting the workload from one to different nodes depending on the overloaded condition.

Dynamic scheduling algorithms are able to establish good load balancing in cloud systems with a lower degree of imbalance between VMs (nodes). Some examples of dynamic resource scheduling algorithms are heterogeneous earliest finish time (HEFT), clustering based heterogeneous with duplication (CBHD), and weighted least connection (WLC). Also, as already stated in Section 1, metaheuristic approaches, particularly swarm intelligence, have proven to be efficient dynamic resource scheduling techniques. For more information regarding swarm algorithm applications to the cloud computing domain, please refer to Section 2.3.

The need for sophisticated resource scheduling algorithms rises with the constant growth in number—as well as in the complexity—of submitted tasks. Existing scheduling algorithms and techniques should be improved, and new ones should be devised, accordingly. The scheduling of resources must be efficient in order to satisfy end-users' requirements with no negative impact on the service level agreement (SLA).

2.3. Swarm Intelligence Overview and Cloud Computing Applications

Metaheuristics can be defined as high-level procedures or heuristics that simulate behavior of some kind of natural phenomenon or system. Metaheuristics are trying to gradually improve potential problem solutions in a set of iterations by using randomness.

By utilizing the criteria of type of phenomenon that is simulated, metaheuristics can be categorized into two groups: bio-inspired (nature-inspired) metaheuristics and those that are not inspired by nature. The nature-inspired approaches are further segmented into evolutionary algorithms (EA) and swarm intelligence. The most widely used EA approach is genetic algorithm (GA) [37].

Swarm intelligence is a rather new category of nature-inspired algorithms that mimics the collective behavior of a group (swarm) of organisms from nature. Examples of such groups include colonies of bees and ants, flocks of birds and fish, herds of elephants, groups of bats, etc. Nature can be portrayed as an inspiring source of concepts, mechanisms, and principles for designing artificial

computing systems that are able to address complex computational problems [38]. One of the most essential characteristics of a swarm is that it consists of relatively simple and unsophisticated agents, which collectively show intelligent behavior.

Swarm intelligence algorithms have many implementations for heterogeneous NP hard tasks from the real-world, including in the area of cloud computing, as can be seen in the literature. In the following few paragraphs, a brief review of some the most well-known swarm intelligence approaches is presented along with its applications in the cloud computing domain.

The artificial bee colony (ABC) algorithm simulates groups of honey bee swarms [39], and it is known as a successful solver of NP hard challenges [40–42]. The ABC algorithm has several successful applications in cloud computing. In [43], the authors implemented a technique that combines task migration with cloud computing load balancing using the ABC approach. The agile task handling approach using ABC has been shown in [44], where the authors have focused on improving makespan time and the degree of imbalance indicators. Multi-objective task scheduling approaches using ABC have also been implemented where the objectives of energy consumption, time span of the tasks, resource cost, and utilization were taken into consideration [45].

The firefly algorithm (FA) algorithm was inspired by the lighting properties of the fireflies. It was created by Yang in 2009, and it was first tested on standard benchmark problems [46]. Many modified and hybridized versions of the FA can be found in the literature [47–50]. Recently, the FA has been applied in the deep learning domain [51] for designing convolutional neural network (CNN) architecture [52]. The FA algorithm has also been implemented in many practical problems from the cloud computing domain, where it showed good performance. Some of the examples include the workflow scheduling problem [53] and load balancing [54].

Other swarm intelligence algorithms that have been devised by Yang are the bat algorithm (BA) [55] and cuckoo search (CS) [56]. Similar to the FA, the BA and CS have showed superior performance metrics when tackling many practical challenges [16,57–59]. The BA algorithm has also been adapted for cloud computing challenges such as scheduling workflow applications [60,61] and cloud service composition [62]. There are also some implementations of the CS algorithm for the cloud computing domain [63,64].

The monarch butterfly optimization (MBO) is a relatively novel algorithm that was created by Wang [65]. Shortly after it was devised, many MBO versions emerged [66–68]. The MBO has also been implemented for cloudlet scheduling problems in cloud computing environments [4]. Another relatively new swarm approach that is worth mentioning is the tree growth algorithm (TGA) [69]. With many implementations, the TGA is positioned as a robust optimization method [70,71]. Some TGA adaptions for cloud computing load scheduling can be found in the literature survey [33].

Some other swarm intelligence representatives that have been implemented for cloud computing problems include particle swarm optimization (PSO) [30], gravitational search algorithm (GSA) [35], ant colony optimization (ACO) [72–74], moth search (MS) [75], and many others [76].

In addition to the mentioned swarm intelligence algorithms, many others have also proven to be efficient optimization approaches, for example elephant herding optimization (EHO) [5,7,31,32,77], the fireworks algorithm (FWA), [78–80], and brain storm optimization (BSO) [81].

3. Proposed Models

As stated above, for the purpose of the research that was presented in this paper we conducted two types of cloud computing resource scheduling simulations: one with a real data set and one with an artificial data set. In each simulation, we used slightly different resource scheduling models.

When performing resource scheduling simulations with a real data set, we used the single-objective model, where we took the makespan (MS) as the objective, as in [75]. In simulations with an artificial data set we employed multi-objective resource scheduling with MS and budget cost objectives, as in [26,64].

After we show a basic background information, both models will be presented.

The cloud hardware infrastructure is organized in the cloud data centers. The cloud data centers have a limited number of physical servers, which are usually refereed as hosts. Each host has many attributes such as a host unique identifier (hostID), number of processing elements (PE), performance of each PE defined in MIPS (million instructions per second), etc. Each physical server can host multiple VMs that utilize a time-shared or space-shared VM scheduling policy.

When clients (end users) send tasks to the cloud system environment, the tasks first arrive at the task manager component, which organizes the tasks and provides the status of each requested task to the end user. It then forwards the requested tasks to the next component, the task scheduler. The task scheduler assigns all the tasks to the available and suitable VMs by employing the scheduling algorithm. Each VM is considered available if it has finished the processing of previous tasks, or if it does not have a task up ahead. The goal is to effectively use available VMs for tasks processing, but not to overload the cloud system. The result of overloading the cloud system would be malfunctions, end-user dissatisfaction, etc.

The basic goal of tackling the resource scheduling problem in cloud infrastructure is allocation of available cloud system resources to submitted tasks, by achieving one or more objectives, like minimization of the makespan (MS) or time completion of the tasks from the queue, minimization of the total cost and energy consumption by cloud resources, etc. [26,75].

In modern cloud computing literature, the most commonly used resources are VMs. For that reason, throughout the rest of this paper, the terms VMs and resources will be used interchangeably.

3.1. The Model Utilized in Simulations with a Real Data Set (First Model)

As already mentioned, the first model was utilized in simulations with a real data set and it is similar to the model presented in [75]. This model includes only the MS objective.

The mathematical formulation of the single-objective resource scheduling model includes the following. Let CI denote the cloud infrastructure that consists of N_{ph} physical hosts (PH), where in turn each host is composed by the N_{vm} virtual machines (VM):

$$CI = \{PH_1, PH_2, ..., PH_i, ..., PH_{N_{ph}}\}, \tag{1}$$

where each $PH_i (i = 1, 2, 3, ..., N_{ph})$ can be denoted as:

$$PH_i = \{VM_1, VM_2, ..., VM_j, ..., VM_{N_{vm}}\}, \tag{2}$$

where VM_j represents the j-th VM that is allocated within the particular physical host. Each VM_j is defined with the following set of properties (attributes):

$$VM_j = \{VMID_j, MIPS_j\}, \tag{3}$$

where $VMID_j$ and $MIPS_j$ denote the unique identifier (serial number) and processing performance in terms of MIPS units of VM_j, respectively.

The end-users submit set of tasks (TSK) that should be mapped and processed on a available and adequate VM:

$$TSK = \{T_1, T_2, T_3, ..., T_k, ..., T_{N_{tsk}}\}, \tag{4}$$

where the N_{tsk} presents the total number of tasks submitted by the end-users, and task T_k represents the k-th task in the sequence, which can be more accurately defined as:

$$T_k = \{TID_k, length_k, ETC_k, P_k\}, \tag{5}$$

where TID_k and $length_k$ denote the unique identifier and the length expressed in million instruction (MI) units of the task k. The P_k represents the priority of the task k, while the ETC_k denotes the expected time to complete for a task k.

The allocation (mapping) of T_{tsk} tasks to N_{vm} VMs has a immense impact on the general cloud system performance.

The expected time to complete (ETC) of the k-th task on the j-th VM can be calculated as follows:

$$ETC_{k,j} = \frac{length_k}{MIPS_j}, \tag{6}$$

where $j = 1, ..., N_{vm}$ and $k = 1, ..., N_{tsk}$.

The ETC matrix could be considered and used as a task model for a cloud environment with heterogeneous resources [29].

The ETC matrix of a size $N_{tsk}xN_{vm}$ denote the execution time required to complete each task on a each VM machine [75]:

$$ETC = \begin{bmatrix} ETC_{1,1} & ETC_{1,2} & ETC_{1,3} & \cdots & ETC_{1,j} & \cdots & ETC_{1,N_{vm}} \\ ETC_{2,1} & ETC_{2,2} & ETC_{2,3} & \cdots & ETC_{2,j} & \cdots & ETC_{2,N_{vm}} \\ ETC_{3,1} & ETC_{3,2} & ETC_{3,3} & \cdots & ETC_{3,j} & \cdots & ETC_{3,N_{vm}} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ ETC_{N_{tsk},1} & ETC_{N_{tsk},2} & ETC_{N_{tsk},3} & \cdots & ETC_{N_{tsk},j} & \cdots & ETC_{N_{tsk},N_{vm}} \end{bmatrix}.$$

If we ought to establish the MS objective as in [75], first the execution time (ET) of all VMs has to be calculated. The execution time of the j-th VM (ET_j) for the task k depends on the decision variable $x_{k,j}$ [29]:

$$x_{k,j} = \begin{cases} 1, \text{ if } C_k \text{ is allocated to } VM_j, \\ 0, \text{ if } C_k \text{ is not allocated to } VM_j. \end{cases} \tag{7}$$

Then, the ET_j, where j is in the range $[1, N_{vm}]$, can be calculated as:

$$ET_j = \sum_{k=1}^{N_{tsk}} x_{k,j} \cdot ETC_{k,j}. \tag{8}$$

The MS objective is the maximum of ET for all VMs:

$$MS = \max[ET_j]_{j=1}^{N_{vm}}, \tag{9}$$

$$\forall k \in [1, N_{tsk}] \text{mapped to } j\text{th VM}, j = 1, 2, 3, ..., N_{vm}. \tag{10}$$

In order to more adequately evaluate the performance of the proposed metaheuristics, we have considered the degree of imbalance (DI) which can be calculated as:

$$DI = \frac{T_{max} - T_{min}}{T_{avg}}, \tag{11}$$

where T_{max}, T_{min} and T_{avg} defines the maximum, minimum, and average execution time of all VMs, respectively. The degree of imbalance (DI) was employed as well in [75].

3.2. Model Used for Simulations with Artificial Data Set (Second Model)

The second model, which was utilized in practical simulations with artificial data sets, belongs to the group of multi-objective resource scheduling models with the basic objectives of minimizing the MS and the budget cost. A similar model was employed in [26,64].

This model is known in the literature as a resource scheduling model based on the performance and budget constraints [64]. The performance function is represented by the MS. One of the

requirements of this model is that all the tasks should finish before the deadline and with costs that fall into the scope of a predefined budget.

Our multi-objective model in the cloud computing environment, besides the MS objective, also takes into account the cost function of the CPU and memory of the VMs (resources). Mathematical formulations which are used in this research are similar to those in [26]. The overall budget cost for task scheduling is defined by combining the cost function of the CPU and memory. Afterwards, the fitness is calculated by adding the budget cost function and MS of the process of scheduling.

The fitness depends on the solution's quality. The solutions ought to have minimum MS and minimum cost functions. First, the cost of the CPU time of VM j is calculated by using the following equation:

$$C_{(x)} = \sum_{j=1}^{N_{VM}} C^{cost}(j), \tag{12}$$

where the $C^{cost}(j)$ denotes the cost of the CPU of the virtual machine VM_j, and the N_{VM} represents the total number of VMs in the cloud infrastructure. The notation x denotes a feasible solution. The $C^{cost}(j)$ can be further calculated as:

$$C^{cost}(j) = C_{base} \times C_j \times t_{ij} + C_{Trans}, \tag{13}$$

where the C_{base} denotes the cost basis, C_j denotes the CPU time cost of the virtual machine VM_j, the t_{ij} is the time in which the task T_i is refined in the resource R_j (virtual machine VM_j). C_{Trans} represents the transmission cost of the CPU. The C_{base} and C_{Trans} are constants:

$$C_{base} = 0.17/hr, \tag{14}$$

$$C_{Trans} = 0.005. \tag{15}$$

To calculate the cost function of a memory, the following equation is used:

$$M_{(x)} = \sum_{j=1}^{N_{VM}} M^{cost}(j), \tag{16}$$

in which the $M^{cost}(j)$ presents the cost of the memory of the virtual machine VM_j. The total number of virtual machines is denoted as N_{VM}. The $M^{cost}(j)$ can be calculated as follows:

$$M^{cost}(j) = M_{base} \times M_j \times t_{ij} + M_{Trans}, \tag{17}$$

where the base of the memory is represented as M_{base}, the memory of virtual machine VM_j is M_j and t_{ij} denotes a time when the task T_i is processed on the resource R_j (virtual machine VM_j). The M_{Trans} presents the transmission cost of the memory. The values of M_{base} and M_{Trans} are constants:

$$M_{base} = 0.05GB/hr, \tag{18}$$

$$M_{Trans} = 0.5. \tag{19}$$

Finally, the budget cost is calculated by adding the cost function of the CPU and memory of the VM:

$$B_{(x)} = C_{(x)} + M_{(x)}, \tag{20}$$

where the $B_{(x)}$ presents the budget cost function for the end-user, $C_{(x)}$ denotes the cost function of the CPU and the $M_{(x)}$ is the cost function of the memory. Hence, the fitness is calculated by the following:

$$H_{(x)} = MS_{(x)} + B_{(x)}, \tag{21}$$

where $MS_{(x)}$ denotes the MS value, that is, the performance function and it should be less than or equal to the deadline of the task. The MS is defined by the following equation:

$$MS_{(x)} \leq \sum_{i=1}^{|T|} D_i, \tag{22}$$

in which D_i presents the deadline for the task T_i, In the previous equation (Equation (21)), $B_{(x)}$ denotes the budget cost function of the tasks that comprises of the CPU cost and memory cost, and it should be less than or equal to the end users' budget cost. According to this, the constrained budget cost function $B_{(x)}$ can be modeled:

$$B_{(x)} \leq \sum_{i=1}^{|T|} B_i, \tag{23}$$

where B_i denotes the end users' budget cost of the task T_i.

For more information about this resource scheduling model in cloud enviroments, please refer to [26,64].

4. Original and Hybrid Whale Optimization Algorithm

The WOA was first proposed in 2016 by Mirjalili and Lewis [22] for tackling unconstrained and constrained continuous optimization problems. The main inspiration behind this approach is bubble-net hunting strategy of humpback whale. In only a couple of years, the WOA positioned as a robust optimizer capable of solving different kinds of problems, particularly in modified/improved implementations [23,24].

In this section, we first give brief insights into the details of the original WOA. Afterwards we present theoretical discussion of the basic WOA's version drawbacks that we discovered during practical simulations, along with the possible directions for its improvements. Finally, we show our devised hybrid WOA approach that overcomes some of the deficiencies of the original WOA implementation.

4.1. Original Whale Optimization Algorithm Overview

The search process of the WOA is performed by mathematically modeling the humpback whales bubble-net feeding strategy. In the nature, humpback whales express a form of cooperating behavior while hunting their prey by performing a distinctive hunting strategy, which is in the literature refereed as a bubble-net feeding strategy.

The humpback whales perform the bubble-net feeding strategy by diving below a shoal of prey, where they simultaneously blow bubbles and move in a circular trajectory in the direction towards the surface of the water. In this way, the shoal of prey is surrounded by a circular path of bubbles that stimulates it to swim towards the surface [82].

More information about the nature-inspired background of the WOA can be obtained from [22].

As in the case of every other swarm intelligence metaheuristic, the WOA's search process is conducted by simultaneously performing global (exploration) and local (exploitation) search phase. The process of exploitation models the humpback whales' prey encircling and spiral bubble-net attack strategy, while the exploration emulates a search for a prey in a pseudo-random manner.

Since the WOA belongs to the category of population-based optimization methods, a group of artificial agents conduct the search process independently, while at the same time they also establish a form of indirect communications, that enhance the search process. According to the basic WOA terminology, the current best (global best) candidate solution (solution with the greatest fitness value) represents the target prey, while every other solution represents a whale. In order to be consistent with the metaheuristics' terminology and to avoid confusion, terms like whale and prey will be omitted, instead the terms candidate solution and current best solution will be utilized in the rest of the paper.

4.1.1. Exploitation Process

During the phase of exploitation, each candidate solution performs a search in its neighborhood and it is directed towards the position of the current global best solution.

After every solution in the population, a fitness is calculated, and positions of all solutions in the population are updated with respect to the position of the global best solution (individual that has the greatest fitness value) [22]:

$$\vec{D} = |\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)| \tag{24}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D}, \tag{25}$$

where $\vec{X}(t)$ and $\vec{X}^*(t)$ denote candidate and current best solutions in iteration t, respectively, $\vec{A}$ and $\vec{C}$ represent coefficient vectors, while the symbol $\cdot$ is element-wise multiplication operator.

The following expressions are used for calculating coefficient vectors $\vec{A}$ and $\vec{C}$ [22]:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \tag{26}$$

$$\vec{C} = 2 \cdot \vec{r}. \tag{27}$$

In Equations (26) and (27), $\vec{r}$ represents a uniformly distributed pseudo-random vector in the interval $[0, 1]$, while $\vec{a}$ is a vector that is in the basic WOA implementation that is linearly decreased from 2 to 0 during the algorithm's execution.

Based on the Equations (24)–(27), the candidate solution $\vec{X}$ in each iteration can be directed towards any position within the neighborhood of the current best solution $\vec{X}^*$ by adjusting the values of coefficient vectors $\vec{A}$ and $\vec{C}$ by using the pseudo-random vector $\vec{r}$.

The shrinking encircling mechanism and spiral-shaped path mathematical models are used for the purpose of emulating the bubble-net attack strategy.

By linearly decreasing vector α from 2 to 0, in every iteration of algorithm's run, the shrinking encircling mechanism is modeled. In the original WOA version, the following expression is used [22]:

$$\vec{a} = 2 - t\frac{2}{maxIter}, \tag{28}$$

where t and $maxIter$ denote the current and maximum number of iterations in one run, respectively.

Also, in the very first WOA paper [22], the authors recommended that the value of $\vec{A}$ should be adjusted within the interval $[-1, 1]$. In this way, an updated position of the current solution can be set anywhere between its current position and the position of the global best solution from the population.

The second mechanism that guides the process of exploitation (a spiral-shaped path) is executed in two steps: First, the distance between the global best solution ($\vec{X}^*(t)$) and current solution ($\vec{X}(t)$) in iteration t is calculated, and then, a new (updated) position of candidate solution ($\vec{X}(t+1)$) is determined by using a spiral equation [22]:

$$\vec{X}(t+1) = \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*(t), \tag{29}$$

where $\vec{D}'$, as distance between the i-th candidate solution and the global best solution, can be expressed as $\vec{D}' = |\vec{X}^*(t) - \vec{X}(t)|$, and b represents a constant that defines a shape of logarithmic spiral, while l denotes pseudo-random number within the range of -1 and 1.

The fact that the humpback whales move around the prey along a spiral-shaped path and shrinking circle simultaneously, is simulated by choosing between shrinking encircling and spiral-shaped path mechanisms in each iteration with equal probability p:

$$\vec{X}(t+1) = \begin{cases} \vec{X}^*(t) - \vec{A} \cdot \vec{D} \text{ , if } p < 0.5 \\ \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*(t) \text{ , if } p \geq 0.5 \end{cases}. \tag{30}$$

In each iteration, probability p is drawn from a random distribution between 0 and 1.

4.1.2. Exploration Process

The exploration phase is conducted by updating each candidate solution in the population with respect to the position of a randomly chosen solution, instead of the global best solutions, as it is the case in the process of exploitation.

For this exploration phase, a value of coefficient vector $\vec{A}$ is used. If a vector $\vec{A}$ with random values that are greater than or equal to 1 is generated ($|A| \geq 1$), updated positions of candidate solutions are directed towards randomly chosen solution and the global search is performed.

The following expression models WOA's exploration phase [22]:

$$\vec{X}(t+1) = \vec{X}_{rnd}(t) - \vec{A} \cdot \vec{D}, \tag{31}$$

where $\vec{D}$, as the distance between the i-th candidate solution and the random solution rnd from the population at iteration t, can be expressed as $\vec{D} = |\vec{C} \cdot \vec{X}_{rnd}(t) - \vec{X}(t)|$.

4.2. Deficiences of the Original WOA Approach and Related Literature

We performed practical simulations with the original WOA implementation and concluded that some possibilities for improvement exist. We tested WOA on a standard unconstrained (bound-constrained) benchmark set and observed two deficiencies: inappropriate exploitation–exploration balance adjustment and the premature convergence.

The process of exploration in the original WOA is performed only in case when both conditions $p < 0.5$ and $|A \geq 1|$ are satisfied. Moreover, the exploration directs the search process towards existing random solutions from the population and then performs search in its neighborhood (refer to Equation (31). Novel random solutions from the search domain are not considered.

In all other cases, the search process of the original WOA conducts intensification by generating new solutions in the direction of the current best solution vector $\vec{X}^*$. If the conditions $p < 0.5$ and $|A < 1|$ are satisfied, the search process is performed by utilizing Equations (24)–(27), and when the condition $p >= 0.5$ holds, the Equation (29) is used.

The WOA's behavior described above exhibits the inappropriate exploitation–exploration balance adjustment that has influence on the algorithm's behavior, especially in the early iterations of its execution. In the early phases of algorithm's execution, the exploration power in some runs is not enough for the WOA to find the right part of the search space, where an optimum solution is located. This as a consequence generates worse mean values. Moreover, the exploration is oriented towards existing random solutions from the population. Completely new solutions are not generated at all.

According to our conducted simulations, the exploitation–exploration trade-off, that is set towards (in favor of) exploitation may lead to the scenario of premature convergence, when the search process gets trapped in some of the suboptimal regions of the search domain. Premature convergence takes place when two inner control parameters $\vec{A}$ and $\vec{C}$ are unable to generate better solutions in consecutive iterations, and as a consequence, the diversity of population is lost. Moreover, due to the premature convergence, convergence speed of the WOA is not satisfying in some runs.

For example, in some algorithm runs, due to randomization, the initial population is generated far from the optimal domain of the search space. By utilizing exploration, which examines the neighborhood for such solutions, the search process is unable to converge to the optimal region. In such cases, solutions can not be improved in consecutive generations, and the whole population loses diversity.

Some research that proposed improved/hybridized versions of the WOA, which overcome some of its deficiencies, already exists in the literature. For example, Mafarja and Mirjalili presented two hybridized WAO metaheurisitcs, where the simulated annealing (SA) was incorporated into the WOA to improve and to confirm the global best solution found by the WOA search process [23]. In the

LWOA (Levy fight trajectory-based whale optimization algorithm) metaheuristics, after the solutions are updated by the WOA, another update is performed by utilizing the Levy fight trajectory [24]. In [83], enhanced WOA (EWOA) for tackling skeletal structure problems was proposed. The EWOA in each iteration updates only selected variables of the chosen candidate solution.

The improved WOA (IWOA), that adopts a differential evolution (DE) mutation operator and utilizes adaptive strategy for balancing between exploitation and exploration was proposed in [25] and proved to be more efficient an approach than the original WOA. Another improved WOA version that is based on different searching paths and perceptual disturbance was proposed and tested on 23 standard unconstrained benchmarks and obtained better result quality than the original version [84].

4.3. Proposed Hybrid Whale Optimization Algorithm

Taking into account that the two most important mechanisms of any swarm algorithm are exploitation (intensification) and exploration (diversification), and that the efficiency of a swarm algorithm in both terms, convergence speed and solutions quality, heavily depends on the adjusted balance between these two processes, we tried to improve the original WOA implementation by addressing the exploitation–exploration trade-off adjustments.

In general, any swarm algorithm may be enhanced by using minor and/or major improvement strategies. Minor improvements include modifications of some component(s) of the search equation(s), as well as tweaking behavior of the algorithm's control parameters (in many cases, researches introduce dynamic parameter behavior). Major improvements usually refer to hybridization with some other metaheuristics or heuristics. Hybrid algorithms combine the best features of two or more approaches, by replacing the weakness of one approach with advantages (strengths) of some other approach. By examining available literature sources, it can be seen that hybrid algorithms can be very efficient in tackling different types of problems [85–87]. Moreover, hybrids between different categories of artificial intelligence (AI) algorithms also exist. For example, artificial neural networks (ANNs) can be efficiently combined with swarm intelligence [27].

Based on our previous research with hybrid swarm intelligence algorithms [4,31,40,48,59], we devised and implemented a hybridized WOA approach that overcomes deficiencies of the original version. Modifications of the basic WOA can be summarized as follows:

- first, we adapted the exploration mechanism from the ABC metaheuristics;
- second, we introduced an additional dynamic control parameter that controls a new exploration mechanism, and
- third, our proposed approach incorporates the search equation of the firefly algorithm (FA).

Taking into account all listed modifications, we named the proposed approach WOA ABC exploration firefly search (WOA-AEFS). In the following subsections, each of the listed modifications will be explained and the pseudo-code will be provided.

By the end of this section, the following notation will be used: each candidate solution i from the population is represented as a vector $\vec{X}_i$, where each solution is comprised of M decision variables $\vec{X}_i = x_i^1, x_i^2, ...x_i^M$. The value of each decision variable j of the i-th solution (x_i^j) is within the scope $[lb_j, ub_j]$, where lb_j and ub_j denote the lower and upper bound of the search space in j-th dimension, respectively. The notation $fit(\vec{X}_i)$ represents the fitness of the i-th solution, while the objective function value of the same solution is denoted as $f(\vec{X}_i)$.

4.3.1. ABC Exploration Meachanism and Additional Control Parameter

As it was already stated in Section 4.2, in the original WOA implementation, the exploration process is conducted only in scenarios where both conditions $p < 0.5$ and $|A \geq 1|$ are satisfied. Also, according to Equation (31), the exploration is oriented towards existing solutions from the population and new solutions are not generated.

As a consequence of this, in some algorithm runs, where solutions that are distant from the optimum domain of the search space are generated, exploration around the neighborhood of such existing solutions is not enough for the search process to converge to the optimum region. At the other side, the exploration mechanism of the ABC algorithm generates completely random solutions, in the same way as it is performed in the algorithm's initialization phase. This fact, as well as our previous research experience with hybrids between ABC and other metaheuristics [40,50,68,88] inspired us to try to improve WOA's exploration process by adapting the ABC exploration mechanism that is being triggered at the end of every iteration.

For each solution $\vec{X}_i$ from the population, we included additional variable $trial_i$. In each iteration, when a solution $\vec{X}_i$, can not be improved, its $trial_i$ value is increased by one. When a *trial* value of a particular solution reaches a predetermined threshold value called *limit*, this solution is dumped from the population and it is replaced with another, randomly generated solution within the boundaries of the search space by applying the following expression:

$$x_i^j = \theta \cdot (ub_j - lb_j) + lb_j, \tag{32}$$

where x_i^j is the j-th decision variable of the i-th solution, and θ is a uniformly distributed number in the range $[0, 1]$.

This exploration mechanism, especially in early iterations of algorithm's execution, significantly improves convergence speed and maintains population diversity. Moreover, in cases when a search is trapped in some of the suboptimal regions, such a mechanism is able to efficiently direct the search process towards other search domain regions.

However, in later iterations, when a search process has converged to the optimum region and when a fine-tuned search is required, the ABC's exploration mechanism could waste potentially good solutions, that have been stuck for a few iterations. In order to avoid such scenarios, we included additional control parameter, the exploration influence rate (*eir*), where the value is expressed in percents.

At the beginning of the run, the value of *eir* is set to 100, which means that the whole population (100% of population) is under the influence of ABC's exploration. However, during one run, the value of *eir* is gradually decreased, until it finally reaches the threshold value of 5. For example, if the value of *eir* is 50, that means that the 50% worst candidate solutions are exposed to the risk of being discarded from the population if their *trial* value has reached a predetermined *limit* threshold. For this purpose, before the ABC's exploration is applied, the whole population is sorted according to fitness value criteria in descending order.

According to performed empirical experiments, we found that the near optimal value of the dynamic *eir* parameter can be calculated as:

$$eir = 100 \cdot (1 - \frac{t}{maxIter}). \tag{33}$$

4.3.2. FA's Search Equation

The FA is a well-known swarm optimizer that was devised by Yang in 2009 for tackling unconstrained optimization challenges [46]. By examining the literature, it can be seen that the FA is able to tackle many real-life NP hard problems [28,50,52,89,90].

Guided by the goal of further enhancing the performance of the original WOA by improving the convergence speed, we implemented the firefly search strategy in our hybridized WOA-AEFS approach. According to our previously conducted research [40], FA's search equation can substantially improve the convergence speed.

In each iteration, if the condition $p_1 \geq 0.5$ is satisfied, the exploitation process is conducted using standard WOA's spiral-shaped search equation (Equation (29)) or FA's search equation with equal probability p_1.

The FA's search strategy is applied for each parameter j of solution i by using the following equation:

$$x_i^j(t+1) = x_i^j + \beta_0 \cdot e^{-\gamma \cdot r_{i,k}^2}(x_k^j - x_i^j) + \alpha \cdot (rand - 0.5), \tag{34}$$

where β_0 represents attractiveness at r=0, the variation of attractiveness is denoted as γ, α is FA's randomization parameter, *rand* is pseudo-random number between 0 and 1, and k represents a random solution from the population. The distance between solutions i and k, denoted as $r_{i,k}$ is calculated using Cartesian distance:

$$r_{i,k} = ||\vec{X}_i - \vec{X}_k|| = \sqrt{\sum_{j=1}^{M}(x_{i,j} - x_{k,j})^2}, \tag{35}$$

where M is the number of decision variables (solutions' components).

As in the original FA, in our implementation we used a dynamic value for parameter $alpha$ that depends on the current iteration t and the maximum number of iterations ($maxIter$) [46]:

$$\alpha(t) = (1 - (1 - ((10^{-4}/9)^{1/maxIter}))) \cdot \alpha(t-1). \tag{36}$$

For more information about the FA's control parameters, please refer to [46].

4.3.3. Population Initialization and WOA-AEFS Pseudo-Code

As in every other swarm approach, generation of the initial population is performed at the beginning of every run. In WOA-AEFS, in the initialization phase, a population of N solutions $\vec{X}_i(i = 1,2,3,...N)$, where each solution is comprised of M decision variables $\vec{X}_i = x_i^1, x_i^2, ...x_i^M$, is created. The population is represented as matrix of size NxM:

$$P = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & \cdots & x_1^j & \cdots & x_1^M \\ x_2^1 & x_2^2 & x_2^3 & \cdots & x_2^j & \cdots & x_2^M \\ x_3^1 & x_3^2 & x_3^3 & \cdots & x_3^j & \cdots & x_3^M \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_N^1 & x_N^2 & x_N^3 & \cdots & x_N^j & \cdots & x_N^M \end{bmatrix}.$$

Every parameter (decision variable) j of each solution from the population is generated by using the Equation (32) in the initialization phase.

Taking into account everything stated in Section 4.3, the basic steps of execution of our proposed WOA-AEFS approach can be summarized in Algorithm 1.

Algorithm 1 Pseudo-code of the WOA ABC exploration firefly search (WOA-AEFS)

Initialization. Generate random initial population P by applying Equation (32) and for all solutions set the value of *trial* to 0.
Control parameters initialization. Initialize values for ABC exploration control parameters *eir* and *limit* and FA's search control parameter α.
Fitness calculation. Calculate fitness of each candidate solution and determine the global best solution $\vec{X}^*$
while ($t < maxIter$) **do**
  for each candidate solution **do**
    Update $\vec{A}$, $\vec{C}$, $\vec{a}$, l, p, p_1
    if $p < 0.5$ **then**
      if $|A| < 1$ **then**
        Update current candidate solution $\vec{X}$ by using Equation (25) and store new value in $\vec{X}_{new}$
      else if $|A \geq 1|$ **then**
        Choose random solution $\vec{X}_{rnd}$ form the population
        Update current candidate solution $\vec{X}$ by using Equation (31) and store new solution in $\vec{X}_{new}$
      end if
    else if $p \geq 0.5$ **then**
      if $p_1 < 0.5$ **then**
        Update current candidate solution $\vec{X}$ by using FA's search equation (Equation (34)) and store new solution in $\vec{X}_{new}$
      else if $p_1 \geq 0.5$ **then**
        Update current candidate solution $\vec{X}$ by using WOA's spiral-shaped search (Equation (29)) and store new solution in $\vec{X}_{new}$
      end if
    end if
    Choose between old $\vec{X}$ and new $\vec{X}_{new}$ solutions by using greedy selection mechanism
    if new solution is chosen, replace $\vec{X}$ with $\vec{X}_{new}$ and set its *trial* value to 0.
    If old solution is chosen, increment *trial* value of $\vec{X}$.
  end for
  If any solution goes beyond feasible region of the search space, modify it
  Evaluate all solutions in the population by calculating fitness
  Sort all solutions by fitness criteria in descending order
  Replace all solutions that belong to eif % worst solutions in the population and for which $trial \geq limit$ holds with random solution by using Equation (32).
  Update position of the global best solution $\vec{X}^*$ if necessary
  $t = t + 1$
  Recalculate values for *eir* and α parameters by using Equations (33) and (36), respectively.
end while
return The best solution ($\vec{X}^*$) from the population

5. Practical Simulations, Analysis, and Discussion

This section is divided into two parts. In the first part, we present simulation results for standard bound-constrained (unconstrained) benchmarks of our proposed WOA-AEFS algorithm. Real performance and improvements over the basic WOA version can be evaluated only by conducting tests on a wider set of problems, specifically designed for benchmark purposes. If an improved/hybridized algorithm is tested only for one specifically problem, real enhancement over basic implementation could not be established.

The second part of the experimental section shows simulation results for two different instances of cloud resource scheduling problems with both artificial and real data sets. In this part, WOA-AEFS's performance metrics for challenging real world NP hard problems are evaluated.

In both parts of the practical section, WOA-AEFS control parameters' adjustments are given along with comparative analysis with the basic WOA and other state-of-the-art algorithms from the literature. Moreover, a detailed discussion about the convergence speed and behavior of our proposed approach in terms of a solution's quality is presented.

5.1. Tests with Standard Benchmark Functions

In this subsection we first show benchmark function characteristics and the mathematical formulation along with control parameter adjustments of the proposed WOA-AEFS. Later, we present a comparative analysis with the basic WOA, as well as with other outstanding algorithms that were tested on the same benchmark instances.

5.1.1. Benchmark Problems Definitions

To test the robustness, solutions' quality and convergence of the WOA-AEFS, we utilized a set of 23 classical unconstrained benchmarks. The original WOA has also been tested on the same benchmark instances, as it was presented in [22].

For the sake of easier readability, and according to the practice from modern computer science literature [91], we divided benchmark sets into two groups: unimodal and multimodal. From the group of multimodal benchmarks, a special group of fixed-dimension multimodal functions can be extrapolated. The basic difference between multimodal and fixed-dimension multimodal functions is the ability to define a desired number of decision variables. When using fixed-dimension multimodal benchmarks in experiments, the number of decision variables can not be tuned.

For each function (unimodal and multimodal), a unique identifier (ID) is given. In Table 1 characteristics of unimodal benchmarks that were utilized in simulations, are shown. Mathematical formulation of these functions can be found in [22].

Table 1. Characteristics of unimodal benchmark functions

ID	Name of the Benchmark	Separability	Convexity	Scalability
F1	Sphere	Separable	Convex	Scalable
F2	Schwefel's Problem 2.22	Non-separable	Convex	Scalable
F3	Schwefel's Problem 1.2	Separable	Convex	Scalable
F4	Schwefel's Problem 2.21	Separable	Convex	Scalable
F5	Generalized Rosenbrock's Function	Non-separable	Non-convex	Scalable
F6	Step 2 Function	Separable	Convex	Scalable
F7	Quartic Function with Noise	Separable	Non-convex	Scalable

Characteristics of multimodal benchmarks, that were employed in experiments, are given in Table 2. Functions in the Table 2 with IDs F8–F13 are classical multimodal benchmarks, while the functions with IDs from the range F14–F23 belong to the group of fixed-dimension multimodal tests.

Table 2. Characteristics of multimodal benchmark functions.

ID	Name of the Benchmark	Separability	Convexity	Scalability
F8	Generalized Schwefel's Problem 2.26	Separable	Convex	Scalable
F9	Generalized Rastrigin's Function	Non-separable	Convex	Scalable
F10	Ackley's Function	Non-separable	Non-convex	Scalable
F11	Generalized Griewank Function	Non-separable	Non-convex	Scalable
F12	Generalized Penalized Functions	Non-separable	Non-convex	Scalable
F13	Generalized Penalized Functions	Non-separable	Non-convex	Scalable
F14	Shekel's Foxholes Function	Non-separable	Non-convex	Scalable

Table 2. *Cont.*

ID	Name of the Benchmark	Separability	Convexity	Scalability
F15	Kowalik's Function	Non-separable	Non-convex	Scalable
F16	Six-Hump Camel-Back Function	Non-separable	Non-convex	Non-scalable
F17	Branin Function	Non-separable	Non-convex	Non-scalable
F18	Goldstein-Price Function	Non-separable	Non-convex	Non-scalable
F19	Hartman's Family (Hartman 3 Function)	Non-separable	Non-convex	Non-scalable
F20	Hartman's Family (Hartman 6 Function)	Non-separable	Non-convex	Non-scalable
F21	Shekel's Family (Shekel 5 Function)	Non-separable	Non-convex	Scalable
F22	Shekel's Family (Shekel 7 Function)	Non-separable	Non-convex	Scalable
F23	Shekel's Family (Shekel 10 Function)	Non-separable	Non-convex	Scalable

In simulations with benchmarks F1–F13 we utilized 30 dimensions ($D = 30$), as in [22]. For more information regarding the dimension size of fixed-dimension multimodal tests (F14–F23), please refer to [22].

5.1.2. The WOA-AEFS Control Parameter Adjustments

In order to objectively evaluate performance improvements of the WOA-AEFS, that is proposed in this paper, against the original WOA, in all tests we utilized a population with 30 individuals ($N = 30$) that have been improving over 500 iterations ($maxIter = 500$) in each run. These settings generate a total number of 15,000 objective function evaluations ($30 \times 500 = 15,000$). The same values were used in [22].

Since the WOA-AEFS utilizes general algorithm parameters (N and $maxIter$), as well as WAO, ABC, and FA specific parameters, we divided the control parameters into four groups: general parameters, WOA parameters, ABC exploration and FA search parameters. Parameters adjustments, as well as the behavior of dynamic parameters are summarized in Table 3.

Table 3. WOA ABC exploration firefly search (WOA-AEFS) control parameters' adjustments

Parameter Name	Value
WOA-AEFS general parameters	
Population size (N)	30
Number of iterations per run ($maxIter$)	500
WOA search parameters	
Initial value of parameter a	2.0
Dynamic behavior of parameter a	according to Equation (28)
ABC exploration parameters	
Parameter $limit$	17 (according to Equation (37))
Initial value of exploration influence parameter eir	100
Dynamic behavior of parameter eir	according to Equation (33)
FA search parameters	
Initial value for randomization parameter α	0.5
Dynamic behavior of parameter α	according to Equation (36)
Attractivnes at r=0 β_0	0.2
Absorption coefficient γ	1.0

It should be noted that we took values for specific FA's search parameters as in [46]. According to the discussion presented in this paper, as well as regarding our previous experience with the FA metaheuristics [40,47,50,52], the best exploitation ability of the FA can be established by utilizing this set of parameters [46]. If other values would be taken, the performance of the FA search exploitation process would decrease significantly.

Similarly, as in the case of the FA, the best performance of the ABC exploration could be established if the value of *limit* parameter depends on the N and $maxIter$ settings [10,39,41,92]. The value of the *limit* in our proposed approach is determined by using the following expression:

$$limit = round(\frac{maxIter}{N}), \tag{37}$$

where the function $round()$ rounds its arguments to the closest integer value.

5.1.3. Comparative Analysis and Discussion

In all simulations we executed WOA-AEFS in 100 independent runs. In each run we initialized a random population of a size N by using the Equation (32). For the purpose of experiments, we implemented our own pseudo-random number generator, and for each run we used a different seed. The WOA-AEFS algorithm is implemented using Java SE Development Kit 11 technology and the IntelliJ environment.

We first compared WOA-AEFS performance with original WOA metaheuristics. Both algorithms were tested for same benchmark instances (F1–F23) and under the same experimental conditions. Results for the WOA were taken from [22]. We note that we have also implemented basic WOA in Java and obtained the same results as reported in [22]. The WOA showed in [22] was implemented in MATLAB software.

We used the same performance metrics as in [22]—best values averaged over 100 independent runs (mean indicator) and the standard deviation (std). In [22], values are averaged over 30 runs. By performing a simulation with more runs, we wanted to obtain more accurate and precise results.

In order to evaluate improvements over the basic WOA implementation, we first give side-by-side comparison between the WOA-AEFS and the original WOA in Table 4. For the sake of easier visualization of performance indicators, better results for every test instance and for each performance metric (mean and std) are marked bold.

When improving an algorithm (metaheuristics), there should always be trade-offs. For example, for one benchmark instance, the results improve, while for some other, the results get worse. However, it is important that on average (by taking into account all benchmark instances), an improved/hybridized version overcomes the original one.

Results presented in Table 4 provide valid proof of the enhancements over the original WOA, that are obtained by hybridizing WOA with ABC and FA algorithms. On average, the WOA-AEFS significantly outperforms WOA. Benchmark instances and indicators where original WOA establishes better performance than the WOA-AEFS only include the following: mean value for the F7 test, std metric for the F16 benchmark, and mean and std indicators for the F19 test. Experiments where both approaches accomplished the same results encompass mean indicator values for the F9, F15, F16, and F18 benchmark instances and std value for the F9 test.

In all other cases, WOA-AEFS completely outperformed basic WOA metaheuristics. In the Table 4, we included an additional row, where we counted for each column the number of times a particular algorithm obtained better results. In 18 out of 23 benchmark instances, the WOA-AEFS for both indicators (mean and std) achieved better convergence speed and result quality compared with the original WOA.

In the original WOA approach, due to the dynamic behavior of parameter $\vec{a}$ (adaptive mechanism), the search process tends to accelerate with the iterations progress [22]. This mechanism performs well in cases, when in early iterations, the algorithm has found the promising region of the search space. However, if this was not the case, the algorithm can become stuck in some of the suboptimal regions, and the whole population (again due to the adaptive mechanism) converges to this suboptimal domain and loses diversity. As already mentioned in Section 4.2, this behavior is known as premature convergence. Premature convergence is particularly emphasized in the F8 and F21 simulations, as was

also noted in [22]. The main cause of premature convergence is an inadequate exploitation–exploration trade-off especially in early iterations of algorithms' execution.

Table 4. Comparative analysis—WOA-AEFS vs. WOA for unconstrained benchmarks.

ID	WOA		WOA-AEFS	
	Mean	*Standard Deviation (std)*	*Mean*	*Standard Deviation (std)*
F1	1.41×10^{-30}	4.9×10^{-30}	$\mathbf{2.55} \times 10^{-31}$	$\mathbf{8.75} \times 10^{-32}$
F2	1.06×10^{-21}	2.39×10^{-21}	$\mathbf{3.97} \times 10^{-23}$	$\mathbf{2.99} \times 10^{-23}$
F3	5.39×10^{-7}	2.93×10^{-6}	$\mathbf{6.72} \times 10^{-10}$	$\mathbf{5.04} \times 10^{-10}$
F4	0.072581	0.39747	**0**	**0**
F5	27.865580	0.763626	**3.29**	**0.012500**
F6	3.116266	0.532429	$\mathbf{1.5} \times 10^{-19}$	$\mathbf{7.59} \times 10^{-20}$
F7	**0.001425**	0.001149	0.001452	**0.000851**
F8	−5080.76	695.7968	**−13239.3**	**48.3**
F9	0	0	0	0
F10	7.4043	9.897572	**0.013**	**0.0051**
F11	0.000289	0.001586	**0**	**0**
F12	0.339676	0.214864	$\mathbf{9.29} \times 10^{-15}$	$\mathbf{2.02} \times 10^{-15}$
F13	1.889015	0.266088	$\mathbf{7.56} \times 10^{-14}$	$\mathbf{2.09} \times 10^{-14}$
F14	2.111973	2.498594	**0.998023**	$\mathbf{1.99} \times 10^{-16}$
F15	0.000572	0.000324	**0.000322**	0.000324
F16	−1.03163	$\mathbf{4.2} \times 10^{-7}$	−1.03163	2.87×10^{-6}
F17	0.397914	2.7×10^{-5}	**0.397887**	$\mathbf{1.83} \times 10^{-10}$
F18	3	4.22×10^{-15}	3	$\mathbf{2.35} \times 10^{-15}$
F19	**−3.85616**	**0.002706**	−3.85529	0.003812
F20	−2.98105	0.376653	**−3.31952**	**0.025821**
F21	−7.04918	3.629551	**−10.1532**	$\mathbf{3.02} \times 10^{-12}$
F22	−8.181780	3.829202	**−8.925003**	**3.352013**
F23	−9.34238	2.414737	**−10.5364**	$\mathbf{3.5} \times 10^{-13}$
Better	2	2	18	18

By using the ABC exploration mechanism, our proposed WOA-AEFS algorithm avoids the premature convergence, as was empirically proved. In cases of F8 and F21 benchmarks, the WOA-AEFS successfully managed to avoid trapping in local optimum regions. However, by using the ABC exploration, some good solutions could be wasted and our WOA-AEFS compensates this by utilizing dynamic parameter *eir* and the very efficient FA's search equation (please refer to Section 4.3). Moreover, WOA-AEFS still employs adaptive shrinking mechanism (parameter $\vec{a}$) of the original WOA and the search process is accelerating with the increase of iterations.

Convergence speed graphs of WOA and WOA-AEFS for F8 and F21 benchmark instances are given in Figure 1.

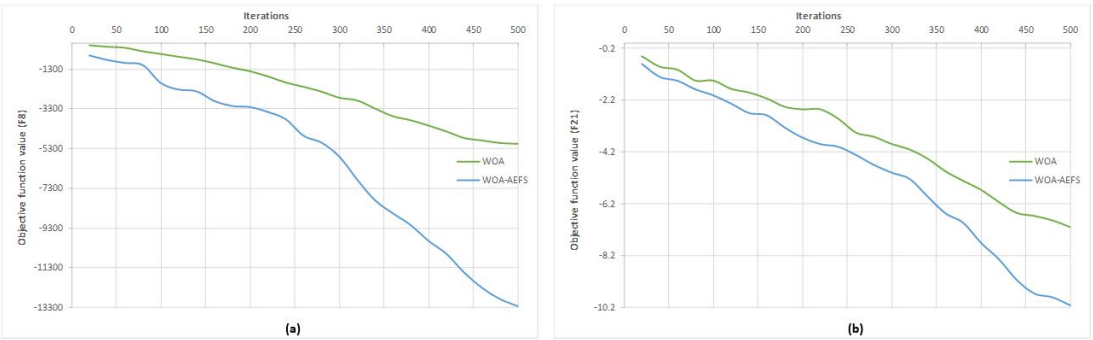


Figure 1. Convergence speed graphs: F8 benchmark (**a**) –F21 benchmark (**b**).

From the Figure 1, some interesting conclusions about the algorithms' behavior can be drawn. First, it should be noted that the both algorithms accelerate with the increase of iteration number due to the shrinking behavior mechanism. Secondly, it can be concluded that the WOA-AEFS after approximately 300 and 400 iterations obtains the value that the basic WOA generates after 500 iterations, when observing F8 and F21 benchmarks, respectively. Also, it can be noticed that the basic WOA in the F21 simulation gets stuck somewhere between the 80th and 120th, and between the 200th and 220th iterations. At the other side, the WOA-AEFS shows non-volatile convergence during the whole execution.

As a general conclusion, by conducting empirical experiments and by performing theoretical analysis, it can be concluded that the WOA-AEFS considerably improves basic WOA version by addressing its deficiencies of inadequately established balance between the exploitation and exploration and by avoiding the premature convergence behavior. However, as stated above, there always must be some kind of a compromise. The basic WOA utilizes fewer number of controls parameters. In order to control the ABC exploration and the FA's search process, the WOA-AEFS employs three more control parameters, two dynamic (*eir* and α) and one static (*limit*).

In addition to the comparative analysis with the basic WOA, we also wanted to see how the WOA-AEFS relates to other state-of-the-art metaheurisitcs, which results in the same benchmark instances could be found in the modern computer science literature. With this objective, we performed another comparative analysis between the WOA-AEFS and the particle swarm optimization (PSO) [93], gravitational search algorithm (GSA) [94], differential evolution (DE) [95] and fast evolutionary programming (FEP) [96]. Results for all approaches are taken from [22,97].

As in the first comparative analysis, we used the same performance metrics as in [22]—the best values averaged over 100 independent runs (mean indicator) and the standard deviation (std). All algorithms are tested under the same experimental conditions (N = 30 and $maxIter$ = 500).

Comparative analysis between the WOA-AEFS and the above mentioned approaches is given in Table 5. We note that we also added basic WOA in comparative analysis because we wanted to evaluate its performance against other state-of-the-art algorithms. Similarly as in the previous comparative analysis, we formatted the best results for each category of tests with bold style. We have also included an additional row, where we counted for each column the number of times a particular algorithm obtained the best results.

Based on the comparative analysis with other state-of-the-art algorithms that is shown in Table 5, it can be stated that the WOA-AEFS on average performed better than all other approaches included in analysis. For example, for the standard deviation indicator, the WOA-AEFS obtained the best results even for 10 benchmarks, while in the case of the mean indicator, the WOA-AEFS performed the best for six benchmarks. The second best algorithm is DE, while the GSA is ranked as the third best algorithm included in the analysis.

Table 5. Comparative analysis—WOA-AEFS vs. other state-of-the-art algorithms for unconstrained benchmarks. Particle swarm optimization (PSO), gravitational search algorithm (GSA), differential evolution (DE), fast evolutionary programming (FEP) and the basic whale optimization algorithm (WOA).

ID	PSO		GSA		DE		FEP		WOA		WOA-AEFS	
	mean	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*	*mean*	*std*
F1	0.000136	0.000202	2.53×10^{-16}	9.67×10^{-17}	8.2×10^{-14}	5.90×10^{-14}	0.00057	0.00013	1.41×10^{-30}	4.9×10^{-30}	$\mathbf{2.55} \times 10^{-31}$	$\mathbf{8.75} \times 10^{-32}$
F2	0.042144	0.045421	0.055655	0.194074	1.5×10^{-9}	9.9×10^{-10}	0.008100	0.000770	1.06×10^{-21}	2.39×10^{-21}	$\mathbf{3.97} \times 10^{-23}$	$\mathbf{2.99} \times 10^{-23}$
F3	70.125620	22.119240	896.5347	318.9559	$\mathbf{6.8} \times 10^{-11}$	7.4×10^{-11}	0.016	0.014	5.39×10^{-7}	2.93×10^{-6}	6.72×10^{-10}	$\mathbf{5.04} \times 10^{-10}$
F4	1.086481	0.317039	7.35487	1.741452	0	0	0.3	0.5	0.072581	0.39747	0	0
F5	96.71832	60.11559	67.54309	62.22534	**0**	**0**	5.06	5.87	27.865580	0.763626	3.29	0.012500
F6	0.000102	8.28×10^{-5}	2.5×10^{-16}	1.74×10^{-16}	0	0	0	0	3.116266	0.532429	1.5×10^{-19}	7.59×10^{-20}
F7	0.122854	0.044957	0.089441	0.04339	0.00463	0.0012	0.1415	0.3522	**0.001425**	0.001149	0.001452	**0.000851**
F8	−4841.29	1152.814	−2821.07	493.0375	−11080.1	574.7	−12,554.5	52.6	−5080.76	695.7968	**−13239.3**	**48.3**
F9	46.70423	11.62938	25.96841	7.470068	69.2	38.8	0.046	0.012	0	0	0	0
F10	0.276015	0.50901	0.062087	0.23628	9.7×10^{-8}	4.2×10^{-8}	0.018	**0.0021**	7.4043	9.897572	**0.013**	0.0051
F11	0.009215	0.007724	27.70154	5.040343	0	0	0.016	0.022	0.000289	0.001586	0	0
F12	0.006917	0.026301	1.799617	0.95114	$\mathbf{7.9} \times 10^{-15}$	8×10^{-15}	9.2×10^{-6}	3.6×10^{-6}	0.339676	0.214864	9.29×10^{-15}	$\mathbf{2.02} \times 10^{-15}$
F13	0.006675	0.008907	8.899084	7.126241	$\mathbf{5.1} \times 10^{-14}$	4.8×10^{-14}	0.00016	0.000073	1.889015	0.266088	7.56×10^{-14}	$\mathbf{2.09} \times 10^{-14}$
F14	3.627168	2.560828	5.89838	3.831299	0.998004	3.3×10^{-16}	1.22	0.56	2.111973	2.498594	0.998023	$\mathbf{1.99} \times 10^{-16}$
F15	0.000577	0.000222	0.003673	0.001647	4.5×10^{-14}	0.00033	0.0005	0.00032	0.000572	0.000324	**0.000322**	0.000324
F16	−1.03163	6.25×10^{-16}	−1.03163	$\mathbf{4.88} \times 10^{-16}$	−1.03163	3.1×10^{-13}	−1.03	4.9×10^{-7}	−1.03163	4.2×10^{-7}	−1.03163	2.87×10^{-6}
F17	0.397887	0	0.397887	0	0.397887	9.9×10^{-9}	0.398	1.5×10^{-7}	0.397914	2.7×10^{-5}	0.397887	$\mathbf{1.83} \times 10^{-10}$
F18	3	1.33×10^{-15}	3	4.17×10^{-15}	3	$\mathbf{2} \times 10^{-15}$	3.02	0.11	3	4.22×10^{-15}	3	2.35×10^{-15}
F19	−3.86278	2.58×10^{-15}	−3.86278	$\mathbf{2.29} \times 10^{-15}$	N/A	N/A	−3.86	0.000014	−3.85616	0.002706	−3.85529	0.003812
F20	−3.26634	0.060516	−3.31778	**0.023081**	N/A	N/A	-3.27	0.059	−2.98105	0.376653	**−3.31952**	0.025821
F21	−6.8651	3.019644	−5.95512	3.737079	−10.1532	0.0000025	-5.52	1.59	−7.04918	3.629551	−10.1532	$\mathbf{3.02} \times 10^{-12}$
F22	−8.45653	3.087094	−9.68447	2.014088	**−10.4029**	$\mathbf{3.9} \times 10^{-7}$	−5.53	2.12	−8.181780	3.829202	−8.925003	3.352013
F23	−9.95291	1.782786	−10.5364	$\mathbf{2.6} \times 10^{-15}$	−10.5364	1.9×10^{-7}	−6.57	3.14	−9.34238	2.414737	−10.5364	3.5×10^{-13}
Best	0	0	0	4	4	1	0	1	1	0	6	10

5.2. Cloud Computing Resource Scheduling Simulations

In this section we show the empirical results that our proposed WOA-AEFS algorithm obtained when tackling the resource scheduling problem in cloud computing environments. In order to more accurately measure the performance of our proposed approach, we employed two types of simulations: first with a real data set, and second with an artificial data set.

In simulations with a real data set, we used the single-objective model that was described in Section 3.1, while for the purpose of simulations with an artificial data set, we utilized the multi-objective model that was in detail presented in Section 3.2.

Both simulations were conducted in a CloudSim 3.0.3 environment, which is a self-contained framework that provides an extensible toolkit for cloud environments simulations [98]. For more information about CloudSim, please refer to [98].

Moreover, we have executed both simulations on the computing platform with Intel CoreTM i7-4770K processor at 4 GHz with 32GB of RAM memory, Windows 10 Professional x64 operating system and Java Development Kit 11 (JDK 11) and IntelliJ integrated development environment (IDE). Since the basic CloudSim 3.0.3 is not CUDA^{TM} technology enabled, we have executed algorithms on the central processing unit (CPU).

A potential solution is encoded in both simulations as a set of tasks, where for each task, an available VM (resource) is mapped. Thus, the length of a solution is the total number of submitted tasks to the cloud system.

In the following two subsections we show our empirical results, along with comparative analysis and discussion, for both conducted simulations.

5.2.1. Simulations with Real Data Set

By utilizing real-world task data generated from a large-scale supercomputing facilities, we evaluate performance of a resource scheduling metaheuristic. In this subsection we present results that the WOA-AEFS obtained when a dataset from a real computing environment was used. The same resource scheduling model and the same dataset was utilized in [75].

The tasks utilized in the conducted simulations were generated from NASA Ames iPSC/860 log [99] in Feitelson's Parallel Workloads Archive (PWA). The NASA iPSC is located in the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center. It comprises the data of a sanitized account record for the 128-node iPSC/860 for three months (October 1993–December 1993). The log contains data about 42264 tasks and 128 resources (CPUs). The data was retrieved from the the workload trace log file *NASA-iPSC-1993-3.swf* from the URI: https://www.cse.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/. From this log we retreived the task ID, length of each task expressed in MIPS and the requested processing elements (PEs).

For the purpose of simulations, we generated cloud infrastructure by using the CloudSim framework. In all simulations we utilized 10 homogeneous VMs, that are located on two physical hosts within one data center. The data center was created with default CloudSim characteristics.

The characteristics of hosts and VMs used in experiments are summarized in Tables 6 and 7, respectively.

Table 6. CloudSim hosts configurations for simulations with a real dataset.

Host Identifier	Parameter	Value
Host1	RAM	3 (units: GB)
	CPU type	Intel Core 2 Extreme X6800
	Number of cores (PEs)	2
	CPU ability	27,079 (units: MIPS)
	storage capacity	1 (units: TB)
	bandwidth	10 (units: Gbps)
	VMs scheduling policy	space-shared

Table 6. *Cont.*

Host Identifier	Parameter	Value
Host2	RAM	3 (units: GB)
	CPU type	Intel Core i7 Extreme Edition 3960X
	Number of cores (PEs)	6
	CPU ability	177,730 (units: MIPS)
	storage capacity	1 (units: TB)
	bandwidth	10 (units: Gbps)
	VMs scheduling policy	space-shared

Table 7. CloudSim VMs configurations for simulations with a real data set.

Parameter	Value
number of VMs	10
CPU processing power	9726 (units: MIPS)
RAM memory	512 (units: MB)
bandwidth capacity	1000 (units: Mbps)
storage capacity	10 (units: GB)
task scheduling policy	time-shared
VMM (Hypervisor)	Xen
Operating System	Linux
number of CPUs	1
CPU type	Pentium 4 Extreme Edition

In order to evaluate the robustness and scalability of the proposed WOA-AEFS in terms of the number of tasks, we performed experiments with a small task set (from 100 to 600 tasks) and with a large task set (with 1000 and 2000 tasks).

We adjusted the basic WOA-AEFS parameters as follows: size of population to 30 ($N = 30$) and maximum number of iterations in one run to 1000 ($maxIter = 1000$). Parameter $limit$ was set to 33 ($round(1000/30)$), while the dynamic parameters $\vec{a}$, eir, and α were adjusted during the course of one run by expressions (28), (33) and (36), respectively. The other WOA-AEFS parameters were adjusted as shown in Table 3.

The same CloudSim environment, as well as the parameters for population size and maximum iterations were utilized in [75]. In this way, we wanted to perform a more objective comparative analysis between our proposed WOA-AEFS and other approaches that were presented in [75].

We executed tests for eight instances, with 100, 200, 300, 400, 500, 600, 1000, and 2000 number of tasks. For each experiment instance, the WOA-AEFS was executed in 100 independent runs, and we calculated the averages of the makespan (MS) objective and the DI (Equation (11)). Details of the resource scheduling model that was employed can be retrieved from Section 3.1. We note that we have also adapted the basic WOA and performed experiments against this model, because we wanted to see comparison between the basic and hybridized version.

We first wanted to compare the performance of WOA-AEFS with the basic WOA. Comparative analysis between WOA-AEFS and the original WOA for both sets, small number of tasks (from 100 to 600) and large number of tasks (1000 and 2000) is shown in Figure 2.

From the presented diagram, it is clear that the WOA-AEFS outperformed WOA in all test instances, except in the simulation with 400 tasks. In this case, the original WOA obtained slightly better MS value (around 1%). The most significant performance difference can be noticed in simulations with 600 tasks, where WOA-AEFS outperformed original WOA by more than 25%.

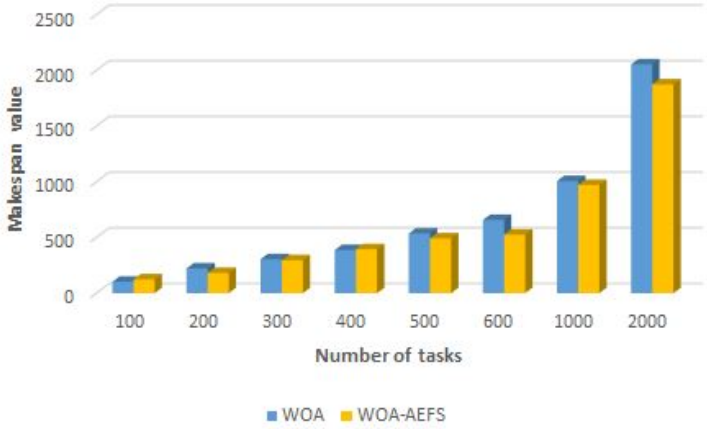


Figure 2. Comparative analysis—WOA-AEFS vs. WOA using NASA iPSC real trace for small and large number of tasks.

Convergence speed graph obtained for WOA-AEFS and WOA in test instance with 200 tasks is given in Figure 3.

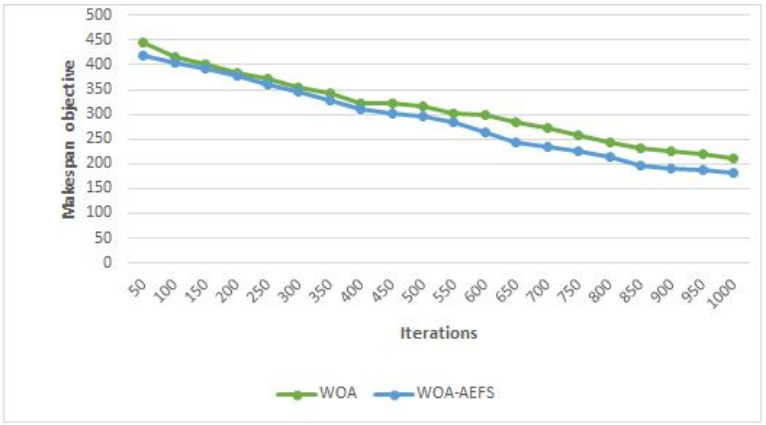


Figure 3. Convergence speed graph WOA-AEFS vs. WOA using NASA iPSC real trace for 200 tasks.

In order to further assess the robustness of WOA-AEFS, we performed comparative analysis with the moth search differential evolution (MSDE) and PSO algorithms, that are presented in [75]. We excluded from comparative analysis heuristics due to the fact that heuristics like round robin (RR) generate high MS value. In [75], results for the original moth search (MS) are also presented. However, we have also excluded this algorithm from comparative analysis, since the MSDE significantly outperforms the basic MS algorithm. However, in order to see how basic WAO compares to other state-of-the-art approaches, it was included in comparison.

We note that the results of the basic WOA were also shown in [75]. However, we did not take results from this paper, instead we performed simulations with our own WOA implementation.

Results with smaller numbers of tasks (from 100 to 600) and with large task sets (1000 and 2000) are shown in Figures 4 and 5, respectively.

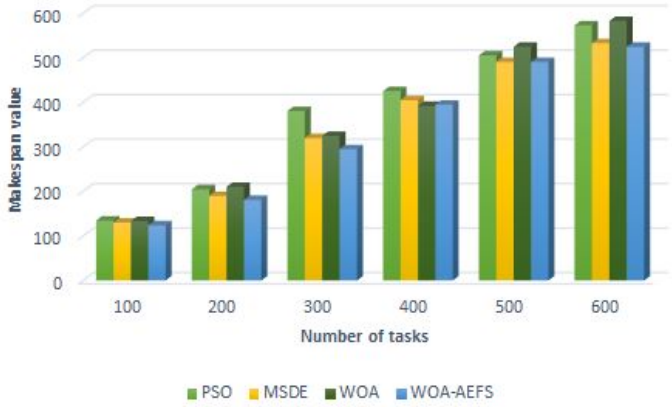


Figure 4. Comparative analysis—WOA-AEFS vs. other state-of-the-art algorithms using NASA iPSC real trace for small numbers of tasks.

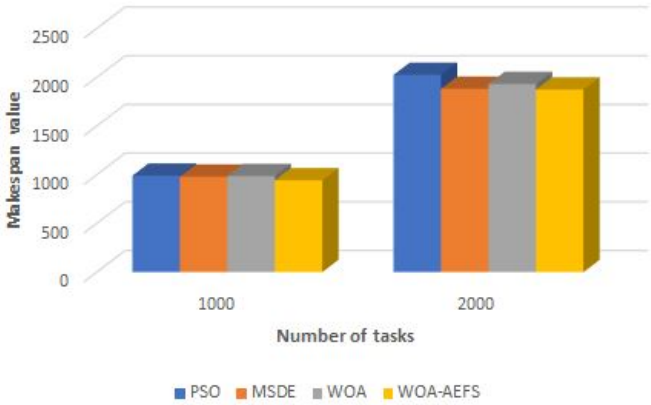


Figure 5. Comparative analysis—WOA-AEFS vs. other state-of-the-art algorithms using NASA iPSC real trace for large numbers of tasks.

Based on the result visualizations from the diagrams presented in Figures 4 and 5, it can be concluded that on average, WOA-AEFS establishes a better MS value than the other state-of-the-art approaches included in comparative analysis. As stated above, only WOA in the test instance with 400 tasks obtained slightly better performance than WOA-AEFS (around 1%). From the diagrams, it can also be noticed that the WOA-AEFS shows the same performance as the MSDE in simulation with 500 tasks. In simulations with a larger number of tasks (1000 and 2000), in both cases WOA-AEFS outperforms all other approaches.

In order to generate a more precise analysis of obtained results, improvements of the MS objective, expressed in percentage, of the WOA-AEFS over other state-of-the-art algorithms are given in Table 8.

Table 8. Improvements of the WOA-AEFS over other state-of-the-art algorithms in simulations with NASA iPSC real trace data set.

Number of Tasks	PSO	MSDE	WOA
100	+8.60%	+4.83%	+7.63%
200	+13.23%	+4.97%	+16.16%
300	+29.08%	+8.47%	+10.18%
400	+7.71%	+2.53%	−0.70%
500	+2.99%	+0.02%	+6.96%
600	+9.19%	+1.52%	+11.08%
1000	+1.89%	0.92%	1.58%
2000	+8.10%	+0.42%	+3.08%

Finally, we wanted to perform comparative analysis between WOA-AEFS and WOA, MSDE and PSO by using the DI indicator (Equation (11)). For more information, please refer to Section 3.1. Comparative analysis of the DI is shown in Table 9 (results for MSDE and PSO were taken from [75]). The best results from each category of test instances are marked bold.

Table 9. Comparative analysis of the DI indicator in simulations with NASA iPSC real trace data set.

Number of Tasks	PSO	MSDE	WOA	WOA-AEFS
100	0.974118	0.972773	1.004538	**0.935002**
200	1.375412	1.138444	1.337259	**1.373259**
300	1.051463	**0.905947**	1.032713	0.983205
400	1.050034	**0.878397**	1.063955	0.883302
500	1.077579	0.942521	1.311571	**0.941056**
600	1.097044	0.983239	1.270371	**0.976200**
1000	**1.013558**	1.040253	1.093521	1.0387226
2000	**0.974118**	1.011911	0.991087	0.9835211
Better	2	2	0	3

Based on the analysis of DI indicator results shown in Table 9, it can be stated that, on average, the WOA-AEFS obtained the best results. The PSO proved to be capable of establishing effective load balancing between VMs in environments with large number of tasks (1000 and 2000), and in these simulations this approach showed the best performance. In simulations with 300 and 400 tasks, the MSDE outperformed WOA-AEFS, as well as other algorithms. Our proposed WOA-AEFS showed superior quality of results in small test instances with 100, 200, 500, and 600 numbers of tasks.

As a general conclusion, in comparative analysis with both a small and large number of tasks, the WOA-AEFS shows better performance than other outstanding metaheuristics. The second best is the MSDE approach, while the original WOA and the PSO on average obtained similar results.

Considering the possibility of implementing metaheuristics-based resource scheduling techniques in a real cloud environment, CPU time required for obtaining promising solution represent an important indicator. Since in [75], details of the computation platform are not given, we could not compare the required CPU time of the WOA-AEFS with the CPU time for the MSDE and PSO. However, since we have implemented and tested both original WOA and the WOA-AEFS on the same computing platform, in this case we were able to perform such comparison. The CPU time comparison is presented in Table 10.

Table 10. Comparative analysis of the CPU time in simulations with NASA iPSC real trace data set.

Algorithm	Number of Tasks							
	100	200	300	400	500	600	1000	2000
WOA	**2730**	4122	6541	**8567**	11,005	**15,349**	**35,152**	118,782
WOA-AEFS	2855	**3983**	**6374**	8750	**10,730**	15,765	36,110	**109,351**

From the presented results it can be stated that in average WOA-AEFS and the original WOA have obtained a similar performance regarding the simulation time. This has important implications for the cloud resource scheduling problem, as well as on the metaheuristics performance, since the complexity of the WOA-AEFS over the original WOA does not have significant influence on the execution time.

5.2.2. Resource Scheduling Simulations with Artificial Data Set

In the second resource scheduling simulation, we used a multi-objective model with performance and budget constraints. In this experiment, the objective was to minimize the MS along with the total cost of CPU and memory resources. For more information about the model, please refer to Section 3.2.

For this experiment, we used an artificial dataset. The simulation environment is generated in the CloudSim 3.0.3 platform, as in experiments with the real data set. For each algorithm run, we generated a random environment with 50 heterogeneous VMs implemented under hosts located in data centers. We have considered varying the number of heterogeneous tasks (from 100 to 500) with different lengths. Moreover, we have also considered various tasks arrival rates that were set to 10 and 40. The same simulation environment was utilized in [64].

The characteristics of tasks, VMs, hosts and data centers are shown in Tables 11 and 12.

Table 11. CloudSim VM, hosts and data centers configurations for simulations with artificial data set.

Entity ame	Parameter	Value
VM	Number of VMs	50
	RAM	512 (units: MB)
	CPU processing power	1860, 2660 (units: MIPS)
	storage capacity	1 (units: GB)
	bandwidth capacity	1000 (units: Mbps)
	Task scheduling policy	time-shared
	VMM (Hypervisor)	Xen
	Operating System	Linux
	Number of CPUs	1
Host	RAM	2 (units: GB)
	storage capacity	10 (units: GB)
	bandwidth capacity	1 (units:Gbps)
	VMs scheduling policy	space-shared
Data center	Number of data centers	10
	Number of hosts	10

Table 12. CloudSim tasks characteristics for simulations with real data set.

Parameter	Value
number of tasks	100–500
tasks length	400–1000 (units: MI)
file size	200–1000 (units: MB)
output size (memory)	20–40 (units: MB)

We have adjusted the basic WOA-AEFS parameters as in the experiments with the real dataset: the size of population was set to 30 ($N = 30$) and the maximum number of iterations in one run was adjusted to 1000 ($maxIter = 1000$). The parameter $limit$ was set to 33 ($round(1000/30)$), while the dynamic parameters $\vec{a}$, eir and α were adjusted during the course of one run by the expressions (28), (33), and (36), respectively. The other WOA-AEFS parameters were adjusted as shown in Table 3.

Similarly as in [64], we have separately evaluated MS, cost, and deadline violation and performed comparative analysis with Min-Min and first come first serve (FCFS) heuristics, as well as with cuckoo search particle swarm optimization (CPSO) and performance budget ACO (PBACO) metaheuristics. Moreover, to more precisely evaluate performance of our proposed WOA-AEFS, we have also included original WOA in the comparative analysis.

Simulation results for Min-Min and FCFS heuristics and CPSO and PBACO metaheuristics were taken from [64], while we have performed simulations for the basic WOA approach using our own implementation in Java. In [64], implementation of state-of-the-art CPSO was shown.

For each simulation, WOA and WOA-AEFS are executed in 100 independent runs and average results are reported. For each run, a new random cloud environment was generated.

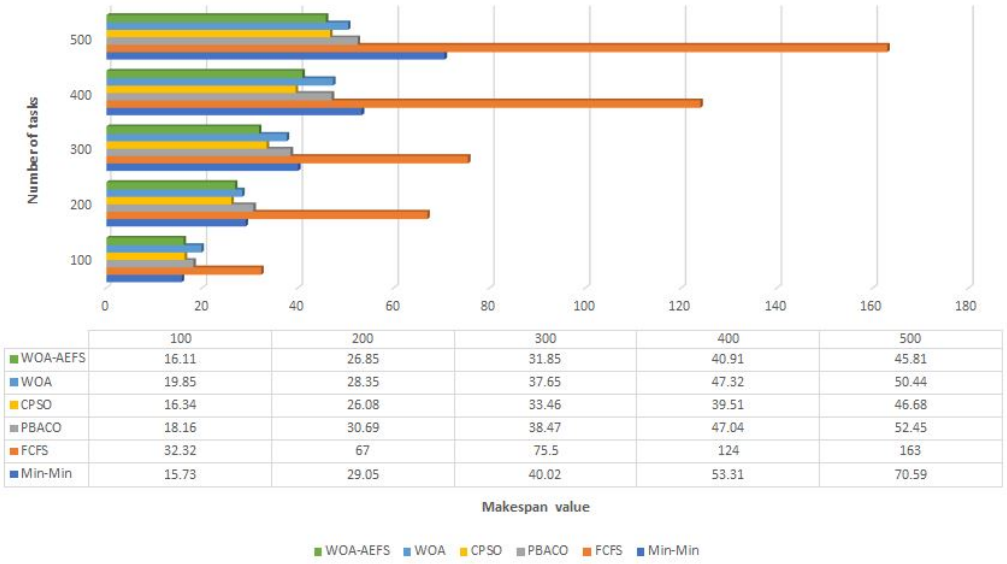
Makespan Analysis

First we show comparative analysis between the WOA-AEFS and other above mentioned heuristics and metaheuristics for the MS value. The result diagrams along with the data table when the tasks arrival rate was set to 10, is shown in Figure 6.

According to the presented results, on average WOA-AEFS obtained the best value for the MS indicator compared to all other approaches included in analysis. It achieved better performance than the CPSO metaheuristics, which proved to be the second best, and much better than the original WOA. The WOA-AEFS established better results than CPSO algorithm in test instances with 100, 300, and 500 tasks. In benchmarks with 200 and 400 tasks, CPSO established slightly better MS value than our proposed WOA-AEFS.

Moreover, based on the presented results, it can be concluded that the WOA-AEFS significantly outperformed basic WOA in all test cases. The largest performance increase could be observed in the benchmark with 100 tasks, where the WOA-AEFS outperformed WOA by around 19%. In tests with 200, 300, 400 and 500 tasks, WOA-AEFS improved the MS value of the original WOA by around 5%, 15%, 13%, and 9%, respectively.

Finally, it can be also concluded that the basic WOA performs similarly to PBACO metaheuristics.

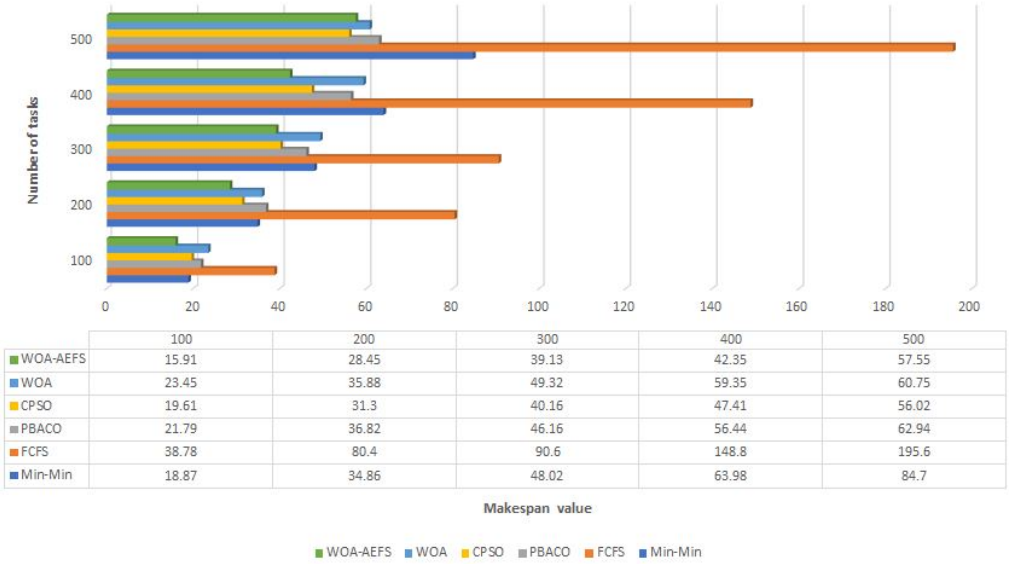


	100	200	300	400	500
WOA-AEFS	16.11	26.85	31.85	40.91	45.81
WOA	19.85	28.35	37.65	47.32	50.44
CPSO	16.34	26.08	33.46	39.51	46.68
PBACO	18.16	30.69	38.47	47.04	52.45
FCFS	32.32	67	75.5	124	163
Min-Min	15.73	29.05	40.02	53.31	70.59

Figure 6. Comparative analysis—WOA-AEFS vs. others using artificial dataset (makespan at arrival rate of 10).

The result diagrams along with the data table when the tasks arrival rate was set to 40 is shown in Figure 7.

In the experiments with tasks arrival rate of 40, the WOA-AEFS obtained better results than state-of-the-art CPSO in four out of five test instances. Only in tests with 500 tasks, did CPSO outperform WOA-AEFS by an insignificant 2%. The most significant performance difference between these two algorithms can be observed in the test with 100 tasks, where the WOA-AEFS obtained better results by around 19%.



	100	200	300	400	500
WOA-AEFS	15.91	28.45	39.13	42.35	57.55
WOA	23.45	35.88	49.32	59.35	60.75
CPSO	19.61	31.3	40.16	47.41	56.02
PBACO	21.79	36.82	46.16	56.44	62.94
FCFS	38.78	80.4	90.6	148.8	195.6
Min-Min	18.87	34.86	48.02	63.98	84.7

Figure 7. Comparative analysis—WOA-AEFS vs. others using artificial dataset (makespan at arrival rate of 40.

As in simulations with a task arrival rate of 10, the WOA-AEFS proved to be the best metaheuristic, while the CPSO took second place. In all test instances, WOA-AEFS completely outperformed the basic WOA approach, which showed similar results to PBACO.

As expected in simulations with both task arrival rates of 10 and 40, heuristics algorithms showed the worst performance. However, the Min-Min heuristics could compete with metaheuristic approaches in some test instances. For example, the Min-Min heuristics obtained a better MS value than the PBACO in tests with 100 and 200 tasks and arrival rates of 10 and 40. On the other hand, Min-Min heuristics achieved, on average, two times better performance than FCFS heuristics.

Cost Objective Analysis

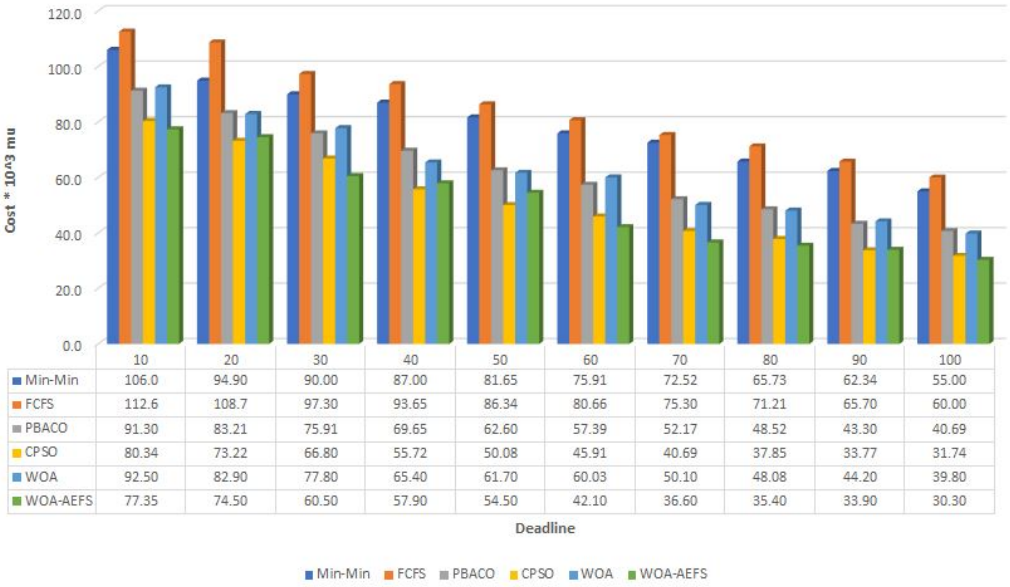
As in [64], cost objective evaluation was performed by considering sets of 200, 400, and 500 tasks with deadlines varying from 10 to 100. Results of cost objective comparative analysis with 200, 400, and 500 tasks with varying deadline constraints are shown in Figure 8–10, respectively.

In all presented graphs, vertical axis represents cost expressed in monetary units (mu), while the horizontal axis is used for plotting deadlines.

When analyzing results presented in Figures 8–10, similarly as in analysis of the MS objective, it can be concluded that on average our proposed WOA-AEFS establishes the best performance. The second best approach is CPSO [64], that only in few test instances outperformed our WOA-AEFS. Some of these instances include: 200 tasks with deadlines of 20, 40, 50, and 90, 400 tasks with deadlines of 40, 50, 70, and 90 and 500 tasks with deadlines of 10, 70, and 100. It should be emphasized that in these benchmarks, CPSO showed only slightly better results than our WOA-AEFS.

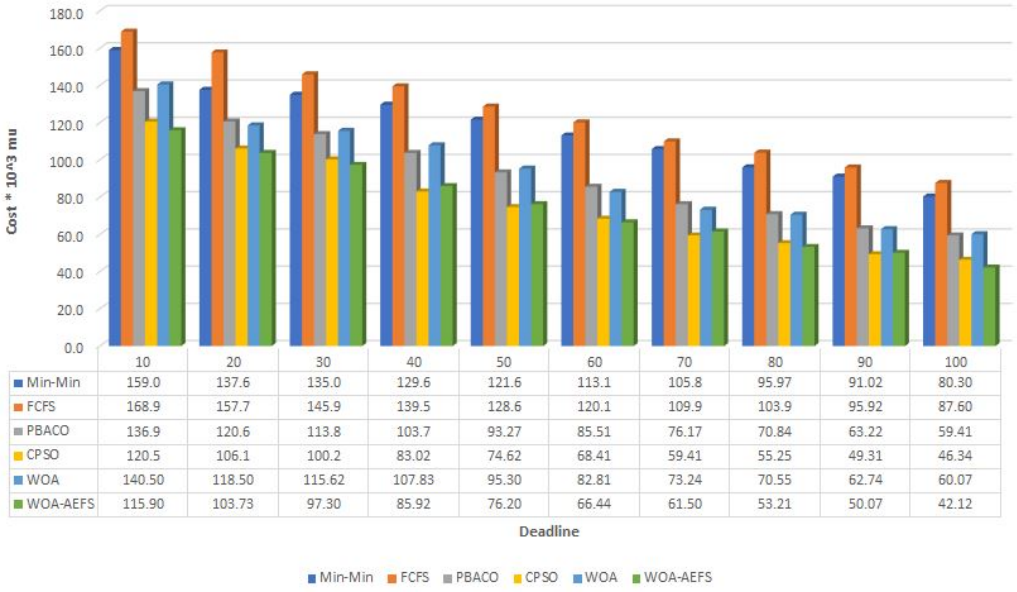
Also, similar to the previous analysis, the proposed WOA-AEFS in all simulation instances achieved much better results that the original WOA, by establishing better solution quality along with the convergence speed. Again, the basic WOA established similar performance metrics to PBACO metaheuristics.

Regarding the cost objective with varying deadline constraints, both heuristics included in comparative analysis showed substantially worse results than all other approaches.



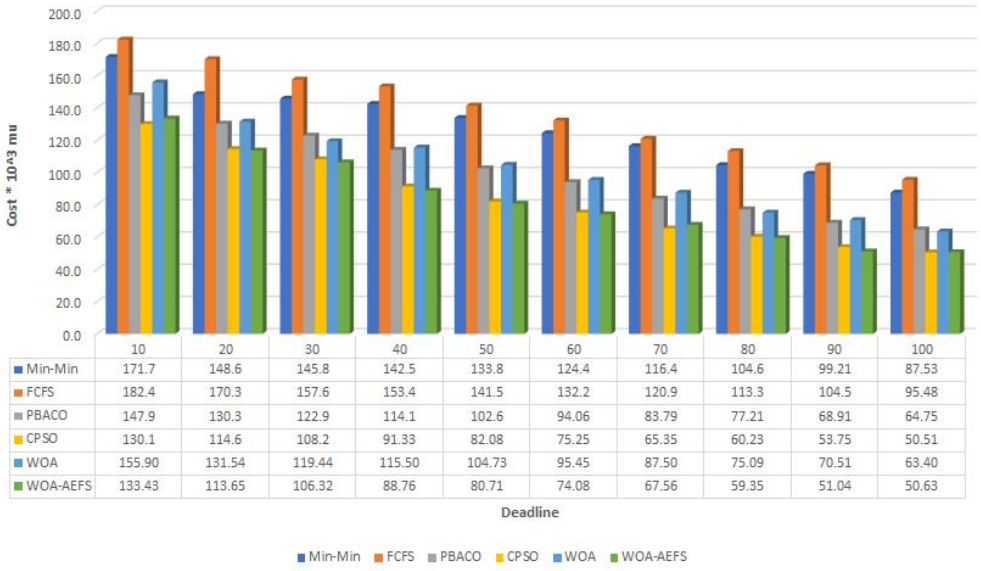
	10	20	30	40	50	60	70	80	90	100
Min-Min	106.0	94.90	90.00	87.00	81.65	75.91	72.52	65.73	62.34	55.00
FCFS	112.6	108.7	97.30	93.65	86.34	80.66	75.30	71.21	65.70	60.00
PBACO	91.30	83.21	75.91	69.65	62.60	57.39	52.17	48.52	43.30	40.69
CPSO	80.34	73.22	66.80	55.72	50.08	45.91	40.69	37.85	33.77	31.74
WOA	92.50	82.90	77.80	65.40	61.70	60.03	50.10	48.08	44.20	39.80
WOA-AEFS	77.35	74.50	60.50	57.90	54.50	42.10	36.60	35.40	33.90	30.30

Figure 8. Comparative analysis—WOA-AEFS vs. others using artificial dataset (costs with 200 tasks and varying deadlines.



	10	20	30	40	50	60	70	80	90	100
Min-Min	159.0	137.6	135.0	129.6	121.6	113.1	105.8	95.97	91.02	80.30
FCFS	168.9	157.7	145.9	139.5	128.6	120.1	109.9	103.9	95.92	87.60
PBACO	136.9	120.6	113.8	103.7	93.27	85.51	76.17	70.84	63.22	59.41
CPSO	120.5	106.1	100.2	83.02	74.62	68.41	59.41	55.25	49.31	46.34
WOA	140.50	118.50	115.62	107.83	95.30	82.81	73.24	70.55	62.74	60.07
WOA-AEFS	115.90	103.73	97.30	85.92	76.20	66.44	61.50	53.21	50.07	42.12

Figure 9. Comparative analysis—WOA-AEFS vs. others using artificial dataset (costs with 400 tasks and varying deadlines.



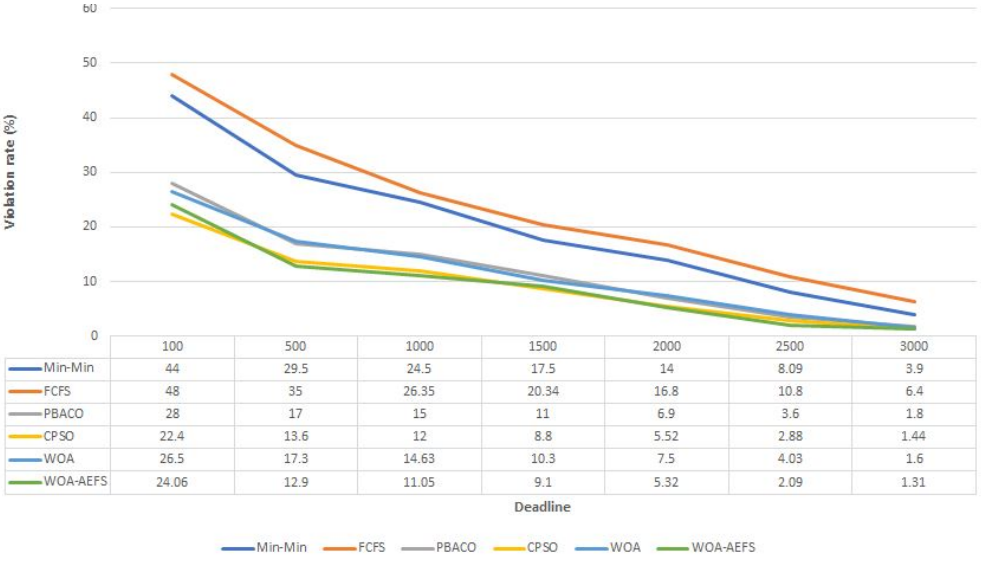
	10	20	30	40	50	60	70	80	90	100
Min-Min	171.7	148.6	145.8	142.5	133.8	124.4	116.4	104.6	99.21	87.53
FCFS	182.4	170.3	157.6	153.4	141.5	132.2	120.9	113.3	104.5	95.48
PBACO	147.9	130.3	122.9	114.1	102.6	94.06	83.79	77.21	68.91	64.75
CPSO	130.1	114.6	108.2	91.33	82.08	75.25	65.35	60.23	53.75	50.51
WOA	155.90	131.54	119.44	115.50	104.73	95.45	87.50	75.09	70.51	63.40
WOA-AEFS	133.43	113.65	106.32	88.76	80.71	74.08	67.56	59.35	51.04	50.63

Figure 10. Comparative analysis—WOA-AEFS vs. others using artificial dataset (costs with 500 tasks and varying deadlines.

Deadline Violation Rate Analysis

Finally, in the last analysis in simulations with artificial dataset, the deadline violation rate has been evaluated in order to validate the QoS of the proposed scheduling metaheuristics. Simulations were conducted with 200, 400, and 500 tasks and varying deadlines (from 100 to 3000) as in [64].

Results of deadline violation comparative analysis with 200, 400, and 500 tasks with varying deadline constraints are shown in Figures 11–13, respectively. In all presented graphs, violation rate was plotted expressed in percents on the ordinate axis, while the deadlines were represented on abscissa axis.



	100	500	1000	1500	2000	2500	3000
Min-Min	44	29.5	24.5	17.5	14	8.09	3.9
FCFS	48	35	26.35	20.34	16.8	10.8	6.4
PBACO	28	17	15	11	6.9	3.6	1.8
CPSO	22.4	13.6	12	8.8	5.52	2.88	1.44
WOA	26.5	17.3	14.63	10.3	7.5	4.03	1.6
WOA-AEFS	24.06	12.9	11.05	9.1	5.32	2.09	1.31

Figure 11. Comparative analysis—WOA-AEFS vs. others using artificial dataset (deadline violation with 200 tasks and varying deadlines).

	100	500	1000	1500	2000	2500	3000
Min-Min	70.4	47.2	39.2	32.725	29.4	20.225	12.48
FCFS	76.8	56	42.16	38.0358	35.28	27	20.48
PBACO	44.8	27.2	24	20.57	14.49	9	5.76
CPSO	35.84	21.76	19.2	16.45	11.59	7.2	4.6
WOA	45.09	26.66	24.05	20.07	15.33	8.26	6.12
WOA-AEFS	33.82	20.92	19.65	16.03	12.07	6.77	4.51

Figure 12. Comparative analysis—WOA-AEFS vs. others using artificial dataset (deadline violation with 400 tasks and varying deadlines).

	100	500	1000	1500	2000	2500	3000
Min-Min	79.2	53.1	44.1	36.75	35	25.079	14.82
FCFS	86.4	63	47.43	42.714	42	33.48	24.32
PBACO	50.4	30.6	27	23.1	17.25	11.16	6.84
CPSO	40.32	24.48	21.6	18.48	13.8	8.92	5.47
WOA	51.99	30.07	26.55	24.42	19.73	10.66	6.92
WOA-AEFS	40.07	22.98	21.85	18.31	12.9	8.74	5.39

Figure 13. Comparative analysis—WOA-AEFS vs. others using artificial dataset (deadline violation with 500 tasks and varying deadlines).

The results shown in Figures 11–13 prove the superiority of our WOA-AEFS approach over other metaheuristics and heuristics. Results are similar as in the previous two simulations.

On average, the WOA-AEFS is the best approach included in comparative analysis, followed by the CPSO that obtained the second best performance in deadline violation tests. The WOA again showed similar performance as PBACO, while all four metaheuristics included in comparative analysis established much better results than Min-Min and FCFS heuristics.

Artificial Data Set Simulations Final Conclusion

According to the analysis of results obtained for the MS and cost objectives, as well as for deadline violation rate, in simulations with artificial data set, our overall conclusion is that the proposed WOA-AEFS algorithm, on average, obtained the best results and proved to be robust and efficient optimization technique for tackling resource scheduling NP hard problem.

The WOA-AEFS in all experiment instances outperformed basic WOA, PBACO and Min-Min and FCFS heuristics. The second best approach, the CPSO only in some cases showed better performance metrics than our WOA-AEFS.

6. Conclusion and Future Research

In this paper an implementation of the improved WOA swarm intelligence metaheuristics adapted for tackling the resource scheduling challenge in cloud computing environments was proposed. The original WAO approach was enhanced by using hybridization as a promising technique that is widely adopted for improving swarm algorithms. By using hybridization, the best components of two or more algorithms are combined.

The main objective of the research proposed in this paper was to improve the resource scheduling problem in cloud computing environments by using swarm intelligence algorithms. The secondary objective of the proposed research was to improve WOA by hybridization with other state-of-the-art swarm algorithms to address the observed deficiencies of the original WOA implementation.

The proposed hybrid algorithm, WOA-AEFS addresses deficiencies of inappropriate balance adjustments between exploitation and exploration and the premature convergence of the original WOA version by adapting exploration mechanism from the ABC algorithm and by incorporating exploitation search procedure from the FA metaheuristic.

In the presented research, a simulation in the standard environment with classic benchmark instances was utilized as the research methodology. Resource scheduling simulations were conducted in the CloudSim framework environment.

To establish and validate performance of the proposed WOA-AEFS approach, three types of tests (simulations) have been performed. The devised approach was first tested on a standard set of bound-constrained benchmarks and the obtained solution quality and the convergence speed were compared with the results generated by the original WOA implementation and PSO, GSA, DE, and FEP metaheuristics, that were all tested on the same benchmark instances.

The last two simulations were performed by using two cloud computing resource scheduling models. In simulations with the first resource scheduling model (single-objective model with MS objective), real data set obtained from NASA iPSC super computing environment was employed and performance of the WOA-AEFS with the original WOA has been compared, as well as with the PSO and the MSDE state-of-the-art metaheuristics.

The second resource scheduling model falls into the category of multi-objective optimization, where the MS and budget costs objectives with task performance and task deadline constraints were considered. In these tests, the WOA-AEFS generated solution quality has been compared with the original WOA, CPSO, and PBACO metaheuristics, as well as with FCFS and Min-Min heuristics.

In all conducted simulations (bound-constrained benchmarks and cloud computing resource scheduling), the WOA-AEFS showed substantially better performance than all approaches included in the comparative analysis. In this way, a hybrid algorithm was devised that overcomes deficiencies of the original WOA metaheuristic, and improvements in solving resource scheduling problem in cloud computing has also been established.

Future research could implement and validate other swarm intelligence algorithms in original, as well as in modified/hybridized versions to other challenges and problems in cloud computing, such as load balancing, migration and VMs provisioning, energy consumption reduction, etc. It would also be interesting to see how efficiently the proposed WOA-AEFS is able to tackle large-scale global optimization challenges.

Author Contributions: I.S. and N.B. proposed the idea. I.S., N.B. and E.T. have implemented and adapted algorithms. The entire research project was conceived and supervised by M.T. The original draft was written by I.S., N.B. and E.T.; review and editing was performed by M.T. All authors participated in the conducted experiments and in discussion of the experimental results.

Funding: This research is supported by the Ministry of Education and Science of Republic of Serbia, Grant No. III-44006.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Abbas, H.; Shaheen, S.; Elhoseny, M.; Singh, A.K.; Alkhambashi, M. Systems thinking for developing sustainable complex smart cities based on self-regulated agent systems and fog computing. *Sustain. Comput. Inform. Syst.* **2018**, *19*, 204–213. [CrossRef]
2. Kalra, M.; Singh, S. A review of metaheuristic scheduling techniques in cloud computing. *Egypt. Inform. J.* **2015**, *16*, 275–295. [CrossRef]
3. Zhan, Z.H.; Zhang, G.Y.; Lin, Y.; Gong, Y.J.; Zhang, J. Load Balance Aware Genetic Algorithm for Task Scheduling in Cloud Computing. In *Simulated Evolution and Learning*; Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 644–655.
4. Strumberger, I.; Tuba, M.; Bacanin, N.; Tuba, E. Cloudlet Scheduling by Hybridized Monarch Butterfly Optimization Algorithm. *J. Sens. Actuator Netw.* **2019**, *8*, 44. [CrossRef]
5. Strumberger, I.; Beko, M.; Tuba, M.; Minovic, M.; Bacanin, N. Elephant Herding Optimization Algorithm for Wireless Sensor Network Localization Problem. In *Technological Innovation for Resilient Systems*; Camarinha-Matos, L.M., Adu-Kankam, K.O., Julashokri, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 175–184.
6. Tuba, E.; Tuba, M.; Beko, M. Mobile wireless sensor networks coverage maximization by firefly algorithm. In Proceedings of the 27th International Conference Radioelektronika, Brno, Czech Republic, 19–20 April 2017; pp. 1–5.
7. Strumberger, I.; Bacanin, N.; Beko, M.; Tomic, S.; Tuba, M. Static Drone Placement by Elephant Herding Optimization Algorithm. In Proceedings of the 24th Telecommunications Forum (TELFOR), Belgrade, Serbia, 21–22 November 2017. [CrossRef]
8. Dolicanin, E.; Fetahovic, I.; Tuba, E.; Capor Hrosik, R.; Tuba, M. Unmanned combat aerial vehicle path planning by brain storm optimization algorithm. *Stud. Inform. Control* **2018**, *27*, 15–24. [CrossRef]
9. Tuba, E.; Strumberger, I.; Bacanin, N.; Tuba, M. Optimal Path Planning in Environments with Static Obstacles by Harmony Search Algorithm. In *Advances in Harmony Search, Soft Computing and Applications*; Kim, J.H., Geem, Z.W., Jung, D., Yoo, D.G., Yadav, A., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 186–193.
10. Bacanin, N.; Tuba, M.; Strumberger, I. RFID Network Planning by ABC Algorithm Hybridized with Heuristic for Initial Number and Locations of Readers. In Proceedings of the 2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim), Cambridge, UK, 25–27 March 2015; pp. 39–44. [CrossRef]
11. Tuba, E.; Tuba, M.; Beko, M. Support Vector Machine Parameters Optimization by Enhanced Fireworks Algorithm. In Proceedings of the 7th International Conference (ICSI 2016), Bali, Indonesia, 25–30 June 2016; pp. 526–534.
12. Tuba, M.; Bacanin, N.; Alihodzic, A. Multilevel image thresholding by fireworks algorithm. In Proceedings of the 2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 21–22 April 2015; pp. 326–330. [CrossRef]
13. Tuba, E.; Tuba, M.; Dolicanin, E. Adjusted Fireworks Algorithm Applied to Retinal Image Registration. *Stud. Inform. Control* **2017**, *26*, 33–42. [CrossRef]
14. Tuba, M.; Tuba, E. Generative Adversarial Optimization (GOA) for Acute Lymphocytic Leukemia Detection. *Stud. Inform. Control* **2019**, *28*, 245–254. [CrossRef]
15. Tuba, E.; Tuba, M.; Jovanovic, R. An algorithm for automated segmentation for bleeding detection in endoscopic images. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, Alaska, 14–19 May 2017; pp. 4579–4586.
16. Strumberger, I.; Bacanin, N.; Tuba, M. Constrained Portfolio Optimization by Hybridized Bat Algorithm. In Proceedings of the 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS),Bangkok, Thailand, 25–27 January 2016; pp. 83–88. [CrossRef]

17. Subotic, M.; Tuba, M. Parallelized Multiple Swarm Artificial Bee Colony Algorithm (MS-ABC) for Global Optimization. *Stud. Inform. Control* **2014**, *23*, 117–126. [CrossRef]
18. Tuba, M.; Brajevic, I.; Jovanovic, R. Hybrid Seeker Optimization Algorithm for Global Optimization. *Appl. Math. Inf. Sci.* **2013**, *7*, 867–875. [CrossRef]
19. Tuba, E.; Dolicanin, E.; Tuba, M. Chaotic brain storm optimization algorithm. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Guilin, China, 30 October–1 November 2017; pp. 551–559.
20. Subotic, M.; Tuba, M.; Stanarevic, N. Parallelization of the artificial bee colony (ABC) algorithm. In Proceedings of the 11th WSEAS International Conference on Neural Networks, Iasi, Romania, 13–15 June 2010; Volume 10, pp. 191–196.
21. Subotic, M.; Tuba, M.; Bacanin, N.; Simian, D. Parallelized cuckoo search algorithm for unconstrained optimization. In Proceedings of the 5th WSEAS Congress on Applied Computing Conference, Faro, Portugal, 2–4 May 2012; Volume 10, pp. 151–156.
22. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
23. Mafarja, M.M.; Mirjalili, S. Hybrid Whale Optimization Algorithm with simulated annealing for feature selection. *Neurocomputing* **2017**, *260*, 302–312. [CrossRef]
24. Ling, Y.; Zhou, Y.; Luo, Q. Lévy Flight Trajectory-Based Whale Optimization Algorithm for Global Optimization. *IEEE Access* **2017**, *5*, 6168–6186. [CrossRef]
25. Bozorgi, S.M.; Yazdani, S. IWOA: An improved whale optimization algorithm for optimization problems. *J. Comput. Des. Eng.* **2019**, *6*, 243–259. [CrossRef]
26. Sreenu, K.; Sreelatha, M. W-Scheduler: Whale optimization for task scheduling in cloud computing. *Cluster Comput.* **2017**. [CrossRef]
27. Le, L.T.; Nguyen, H.; Dou, J.; Zhou, J. A Comparative Study of PSO-ANN, GA-ANN, ICA-ANN, and ABC-ANN in Estimating the Heating Load of Buildings' Energy Efficiency for Smart City Planning. *Appl. Sci.* **2019**, *9*, 2630. [CrossRef]
28. Memari, A.; Ahmad, R.; Jokar, M.R.A.; Rahim, A.R.A. A New Modified Firefly Algorithm for Optimizing a Supply Chain Network Problem. *Appl. Sci.* **2019**, *9*, 7. [CrossRef]
29. Mishra, S.K.; Sahoo, B.; Parida, P.P. Load balancing in cloud computing: A big picture. *J. King Saud Univ. Comput. Inf. Sci.* **2018**. [CrossRef]
30. Kumar, M.; Sharma, S. PSO-COGENT: Cost and energy efficient scheduling in cloud environment with deadline constraint. *Sustain. Comput. Inform. Syst.* **2018**, *19*, 147–164. [CrossRef]
31. Strumberger, I.; Minovic, M.; Tuba, M.; Bacanin, N. Performance of Elephant Herding Optimization and Tree Growth Algorithm Adapted for Node Localization in Wireless Sensor Networks. *Sensors* **2019**, *19*, 2515. [CrossRef]
32. Strumberger, I.; Bacanin, N.; Tuba, M. Hybridized Elephant Herding Optimization Algorithm for Constrained Optimization. In *Hybrid Intelligent Systems*; Abraham, A., Muhuri, P.K., Muda, A.K., Gandhi, N., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 158–166.
33. Strumberger, I.; Tuba, E.; Bacanin, N.; Tuba, M. Dynamic Tree Growth Algorithm for Load Scheduling in Cloud Environments. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 65–72. [CrossRef]
34. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology*; Computer Security DivisionInformation Technology LaboratoryNational Institute of Standards and Technology: Gaithersburg, MD, USA, 2011.
35. Chaudhary, D.; Kumar, B. Cloudy GSA for load scheduling in cloud computing. *Appl. Soft Comput.* **2018**, *71*, 861–871. [CrossRef]
36. Kumar, M.; Sharma, S.; Goel, A.; Singh, S. A comprehensive survey for scheduling techniques in cloud computing. *J. Netw. Comput. Appl.* **2019**, *143*, 1–33. [CrossRef]
37. Magliani, F.; Sani, L.; Cagnoni, S.; Prati, A. Genetic Algorithms for the Optimization of Diffusion Parameters in Content-Based Image Retrieval. In Proceedings of the 13th International Conference on Distributed Smart Cameras (ICDSC 2019), Trento, Italy, 9–11 September 2019; ACM: New York, NY, USA, 2019; pp. 14:1–14:6. [CrossRef]
38. Harifi, S.; Khalilian, M.; Mohammadzadeh, J.; Ebrahimnejad, S. Emperor Penguins Colony: A new metaheuristic algorithm for optimization. *Evol. Intell.* **2019**, *12*, 211–226. [CrossRef]

39. Karaboga, D. *An Idea Based on Honey Bee Swarm for Numerical Optimization*; Technical Report–TR06; Department of Computer Engineering, Engineering Faculty, Erciyes University: KayserİI, Turkey, 2005; pp. 1–10.
40. Tuba, M.; Bacanin, N. Artificial bee colony algorithm hybridized with firefly metaheuristic for cardinality constrained mean-variance portfolio problem. *Appl. Math. Inf. Sci.* **2014**, *8*, 2831–2844. [CrossRef]
41. Bacanin, N.; Tuba, M. Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators. *Stud. Inform. Control* **2012**, *21*, 137–146. [CrossRef]
42. Tuba, M.; Bacanin, N.; Beko, M. Multiobjective RFID Network Planning by Artificial Bee Colony Algorithm with Genetic Operators. In *Advances in Swarm and Computational Intelligence*; Tan, Y., Shi, Y., Buarque, F., Gelbukh, A., Das, S., Engelbrecht, A., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 247–254.
43. Babu, D.; Krishna, P. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Appl. Soft Comput.* **2013**, *13*, 2292–2303.
44. Rastkhadiv, F.; Zamanifar, K. Task Scheduling Based On Load Balancing Using Artificial Bee Colony In Cloud Computing Environment. *Int. J. Adv. Biotechnol. Res.* **2016**, *7*, 1058–1069.
45. Jena, R.K. Task scheduling in cloud environment: A multi-objective ABC framework. *J. Inf. Optim. Sci.* **2017**, *38*, 1–19. [CrossRef]
46. Yang, X.S. Firefly algorithms for multimodal optimization. *Stoch. Algorithms Found. Appl. LNCS* **2009**, *5792*, 169–178.
47. Strumberger, I.; Bacanin, N.; Tuba, M. Enhanced Firefly Algorithm for Constrained Numerical Optimization, IEEE Congress on Evolutionary Computation. In Proceedings of the IEEE International Congress on Evolutionary Computation (CEC 2017), San Sebastian, Spain, 5–8 June 2017; pp. 2120–2127.
48. Tuba, M.; Bacanin, N. Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems. *Neurocomputing* **2014**, *143*, 197–207. [CrossRef]
49. Capor Hrosik, R.; Tuba, E.; Dolicanin, E.; Jovanovic, R.; Tuba, M. Brain Image Segmentation Based on Firefly Algorithm Combined with K-means Clustering. *Stud. Inform. Control* **2019**, *28*, 167–176. [CrossRef]
50. Bacanin, N.; Tuba, M. Firefly Algorithm for Cardinality Constrained Mean-Variance Portfolio Optimization Problem with Entropy Diversity Constraint. *Sci. World J.* **2014**, *2014*, 16. [CrossRef]
51. Donati, L.; Iotti, E.; Mordonini, G.; Prati, A. Fashion Product Classification through Deep Learning and Computer Vision. *Appl. Sci.* **2019**, *9*. [CrossRef]
52. Strumberger, I.; Tuba, E.; Bacanin, N.; Zivkovic, M.; Beko, M.; Tuba, M. Designing Convolutional Neural Network Architecture by the Firefly Algorithm. In Proceedings of the 2019 International Young Engineers Forum (YEF-ECE), Caparica, Lisbon, 8 May 2019; pp. 59–65. [CrossRef]
53. SundarRajan, R.; Vasudevan, V.; Mithya, S. Workflow scheduling in cloud computing environment using firefly algorithm. In Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, India, 3–5 March 2016; pp. 955–960. [CrossRef]
54. Kaur, G.; Kaur, K. An Adaptive Firefly Algorithm for Load Balancing in Cloud Computing. In Proceedings of the *Sixth International Conference on Soft Computing for Problem Solving*; Deep, K., Bansal, J.C., Das, K.N., Lal, A.K., Garg, H., Nagar, A.K., Pant, M., Eds.; Springer: Singapore, 2017; pp. 63–72.
55. Yang, X.S. A new metaheuristic bat-inspired Algorithm. *Stud. Comput. Intell.* **2010**, *284*, 65–74.
56. Yang, X.S.; Deb, S. Cuckoo search via Levy flights. In Proceedings of the World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), Coimbatore, India, 9–11 December 2009; pp. 210–214.
57. Tuba, M.; Alihodzic, A.; Bacanin, N. Cuckoo Search and Bat Algorithm Applied to Training Feed-Forward Neural Networks. In *Recent Advances in Swarm Intelligence and Evolutionary Computation*; Springer International Publishing: Cham, Switzerland, 2015; pp. 139–162. [CrossRef]
58. Bacanin, N. Implementation and performance of an object-oriented software system for cuckoo search algorithm. *Int. J. Math. Comput. Simul.* **2010**, *6*, 185–193.
59. Tuba, M.; Bacanin, N. Hybridized Bat Algorithm for Multi-objective Radio Frequency Identification (RFID) Network Planning. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015), Sendai, Japan, 25–28 May 2015; pp. 499–506
60. Kaur, N.; Singh, S. A Budget-constrained Time and Reliability Optimization BAT Algorithm for Scheduling Workflow Applications in Clouds. *Procedia Comput. Sci.* **2016**, *98*, 199–204. [CrossRef]

61. Raghavan, S.; Sarwesh, P.; Marimuthu, C.; Chandrasekaran, K. Bat algorithm for scheduling workflow applications in cloud. In Proceedings of the 2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV), Shillong, India, 29–30 January 2015; pp. 139–144. [CrossRef]
62. Xu, B.; Sun, Z. A Fuzzy Operator Based Bat Algorithm for Cloud Service Composition. *Int. J. Wire Mob. Comput.* **2016**, *11*, 42–46. [CrossRef]
63. Agarwal, M.; Srivastava, G.M.S. A Cuckoo Search Algorithm-Based Task Scheduling in Cloud Computing. In *Advances in Computer and Computational Sciences*; Bhatia, S.K., Mishra, K.K., Tiwari, S., Singh, V.K., Eds.; Springer: Singapore, 2018; pp. 293–299.
64. Prem Jacob, T.; Pradeep, K. A Multi-objective Optimal Task Scheduling in Cloud Environment Using Cuckoo Particle Swarm Optimization. *Wirel. Personal Commun.* **2019**. [CrossRef]
65. Wang, G.G.; Deb, S.; Cui, Z. Monarch Butterfly Optimization. *Neural Comput. Appl.* **2015**, 1–20. [CrossRef]
66. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Modified Monarch Butterfly Optimization Algorithm for RFID Network Planning. In Proceedings of the 2018 6th International Conference on Multimedia Computing and Systems (ICMCS), Rabat, Marocco, 10–12 May 2018; pp. 1–6. [CrossRef]
67. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Monarch butterfly optimization algorithm for localization in wireless sensor networks. In Proceedings of the 2018 28th International Conference Radioelektronika (RADIOELEKTRONIKA), Prague, Czech Republic, 19–20 April 2018; pp. 1–6. [CrossRef]
68. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Modified and Hybridized Monarch Butterfly Algorithms for Multi-Objective Optimization. In *Hybrid Intelligent Systems*; Madureira, A.M., Abraham, A., Gandhi, N., Varela, M.L., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 449–458.
69. Cheraghalipour, A.; Hajiaghaei-Keshteli, M.; Paydar, M.M. Tree Growth Algorithm (TGA): A novel approach for solving optimization problems. *Eng. Appl. Artif. Intell.* **2018**, *72*, 393–414. [CrossRef]
70. Strumberger, I.; Tuba, E.; Zivkovic, M.; Bacanin, N.; Beko, M.; Tuba, M. Dynamic Search Tree Growth Algorithm for Global Optimization. In *Technological Innovation for Industry and Service Systems*; Camarinha-Matos, L.M., Almeida, R., Oliveira, J., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 143–153.
71. Strumberger, I.; Tuba, E.; Bacanin, N.; Jovanovic, R.; Tuba, M. Convolutional Neural Network Architecture Design by the Tree Growth Algorithm Framework. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8. [CrossRef]
72. Lal, A.; Rama Krishna, C. Critical Path-Based Ant Colony Optimization for Scientific Workflow Scheduling in Cloud Computing Under Deadline Constraint. In *Ambient Communications and Computer Systems*; Perez, G.M., Tiwari, S., Trivedi, M.C., Mishra, K.K., Eds.; Springer: Singapore, 2018; pp. 447–461.
73. Zuo, L.; Shu, L.; Dong, S.; Zhu, C.; Hara, T. A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing. *IEEE Access* **2015**, *3*, 2687–2699. [CrossRef]
74. Gao, R.; Wu, J. Dynamic Load Balancing Strategy for Cloud Computing with Ant Colony Optimizatio. *Future Int.* **2015**, *7*, 465–483. [CrossRef]
75. Elaziz, M.A.; Xiong, S.; Jayasena, K.; Li, L. Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowl.-Based Syst.* **2019**, *169*, 39–52. [CrossRef]
76. Anwar, N.; Deng, H. A Hybrid Metaheuristic for Multi-Objective Scientific Workflow Scheduling in a Cloud Environment. *Appl. Sci.* **2018**, *8*. [CrossRef]
77. Wang, G.G.; Deb, S.; Coelho, L.D.S. Elephant Herding Optimization. In Proceedings of the 2015 3rd International Symposium on Computational and Business Intelligence (ISCBI), Washington, DC, USA, 7–9 December 2015; pp. 1–5.
78. Bacanin, N.; Tuba, M. Fireworks Algorithm Applied to Constrained Portfolio Optimization Problem. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC 2015), Sendai, Japan, 25–28 May 2015; pp. 1242–1249.
79. Tuba, E.; Strumberger, I.; Bacanin, N.; Tuba, M. Bare Bones Fireworks Algorithm for Capacitated p-Median Problem. In *Advances in Swarm Intelligence*; Tan, Y., Shi, Y., Tang, Q., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 283–291.
80. Tuba, M.; Bacanin, N.; Beko, M. Fireworks algorithm for RFID network planning problem. In Proceedings of the 2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 21–22 April 2015; pp. 440–444. [CrossRef]

81. Tuba, E.; Strumberger, I.; Zivkovic, D.; Bacanin, N.; Tuba, M. Mobile Robot Path Planning by Improved Brain Storm Optimization Algorithm. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8. [CrossRef]
82. Goldbogen, J.A.; Friedlaender, A.S.; Calambokidis, J.; McKenna, M.F.; Simon, M.; Nowacek, D.P. Integrative Approaches to the Study of Baleen Whale Diving Behavior, Feeding Performance, and Foraging Ecology. *BioScience* **2013**, *63*, 90–100, [CrossRef]
83. Kaveh, A.; Ghazaan, M.I. Enhanced whale optimization algorithm for sizing optimization of skeletal structures. *Mech. Based Des. Struct. Mach.* **2017**, *45*, 345–362. [CrossRef]
84. zhen Sun, W.; sheng Wang, J.; Wei, X. An Improved Whale Optimization Algorithm Based on Different Searching Paths and Perceptual Disturbance. *Symmetry* **2018**, *10*, 210. [CrossRef]
85. Rodriguez, F.; Garcia-Martinez, C.; Lozano, M. Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: Taxonomy, comparison, and synergy test. *IEEE Trans. Evol. Comput.* **2012**, *16*, 787–800. [CrossRef]
86. Sulaiman, N.; Mohamad-Saleh, J.; Abro, A.G. A hybrid algorithm of ABC variant and enhanced EGS local search technique for enhanced optimization performance. *Eng. Appl. Artif. Intell.* **2018**, *74*, 10–22. [CrossRef]
87. Ghosh, S.; Kaur, M.; Bhullar, S.; Karar, V. Hybrid ABC-BAT for Solving Short-Term Hydrothermal Scheduling Problems. *Energies* **2019**, *12*, 551. [CrossRef]
88. Strumberger, I.; Tuba, E.; Bacanin, N.; Beko, M.; Tuba, M. Hybridized moth search algorithm for constrained optimization problems. In Proceedings of the 2018 International Young Engineers Forum (YEF-ECE), Caparica, Lisbon, 8 May 2018; pp. 1–5. [CrossRef]
89. Tuba, E.; Tuba, M.; Beko, M. Two Stage Wireless Sensor Node Localization Using Firefly Algorithm. In *Smart Trends in Systems, Security and Sustainability*; Yang, X.S., Nagar, A.K., Joshi, A., Eds.; Springer: Singapore, 2018; pp. 113–120.
90. Tilahun, S.L.; Ngnotchouye, J.M.T. Firefly algorithm for discrete optimization problems: A survey. *KSCE J. Civ. Eng.* **2017**, *21*, 535–545. [CrossRef]
91. Yang, X.S. Firefly algorithm, stochastic test functions and design optimisation. *Int. J. Bio Inspired Comput.* **2010**, *2*, 78–84. [CrossRef]
92. Karaboga, D.; Akay, B. A modified Artificial Bee Colony (ABC) Algorithm for constrained optimization problems. *Appl. Soft Comput.* **2011**, *11*, 3021–3031. [CrossRef]
93. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks (ICNN '95), Perth, Australia, 27 November–1 December 1995; Volume 4; pp. 1942–1948.
94. Rashedi, E.; Nezamabadi-pour, H.; Saryazdi, S. GSA: A Gravitational Search Algorithm. *Inf. Sci.* **2009**, *179*, 2232–2248. [CrossRef]
95. Storn, R.; Price, K. Differential Evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
96. Yao, X.; Liu, Y.; Lin, G. Evolutionary programming made faster. *IEEE Trans. Evol. Comput.* **1999**, *3*, 82–102. [CrossRef]
97. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey Wolf Optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
98. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.F.; Buyya, R. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Softw. Pract. Exper.* **2011**, *41*, 23–50. [CrossRef]
99. Meng, J.; McCauley, S.; Kaplan, F.; Leung, V.J.; Coskun, A.K. Simulation and optimization of HPC job allocation for jointly reducing communication and cooling costs. *Sustain. Comput. Inform. Syst.* **2015**, *6*, 48–57. [CrossRef]