KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS VOL. 12, NO. 8, Aug. 2018 3516

Improved Hybrid Symbiotic Organism Search Task-Scheduling Algorithm for Cloud Computing

SongIl Choe[1,2,3,*], Bo Li[2], IlNam Ri[1], ChangSu Paek[3], JuSong Rim[4], SuBom Yun[3]

1 College of Information Science, Kim Il Sung University,
Pyongyang, Democratic People's Republic of Korea
2 College of Management and Economics, Tianjin University, Tianjin 300072, China
3 Department of Information Science, HuiChon Industry University,
HuiChon, Democratic People's Republic of Korea
4 Department of Control Science, University of Science,
Pyongyang, Democratic People's Republic of Korea
* Corresponding author: SongIl Choe (cxl2015316@163.com)

Received April 8, 2018; revised June 25, 2018; accepted July 4, 2018;
published August 31, 2018

Abstract

Task scheduling is one of the most challenging aspects of cloud computing nowadays, and it plays an important role in improving overall performance in, and services from, the cloud, such as response time, cost, makespan, and throughput. A recent cloud task–scheduling algorithm based on the symbiotic organisms search (SOS) algorithm not only has fewer specific parameters, but also incurs time complexity. SOS is a newly developed metaheuristic optimization technique for solving numerical optimization problems. In this paper, the basic SOS algorithm is reduced, and chaotic local search (CLS) is integrated into the reduced SOS to improve the convergence rate. Simulated annealing (SA) is also added to help the SOS algorithm avoid being trapped in a local minimum. The performance of the proposed SA-CLS-SOS algorithm is evaluated by extensive simulation using the Matlab framework, and is compared with SOS, SA-SOS, and CLS-SOS algorithms. Simulation results show that the improved hybrid SOS performs better than SOS, SA-SOS, and CLS-SOS in terms of convergence speed and makespan.

http://doi.org/10.3837/tiis.2018.08.001 **ISSN : 1976-7277**

Keywords: cloud computing, cloud task scheduling, symbiotic organisms search, simulated annealing, chaotic local search

1. Introduction

Cloud computing is a model with rapid growth in recent years, increasing due to technological developments in distributed computing, grid computing, and parallel computation. Cloud computing is a model that can obtain resources quickly from a configurable shared resources pool of servers, storage, networks, services, and applications in real time and based on demand. The supply and release of resources can finish in a shorter time to reduce the load on resource management and to keep the interactions between service providers to a minimum [1].

The basic principles of task scheduling in the cloud are to break down the tasks reported by masses of users into smaller tasks via the network, using multiple computers connected to the network to search, compute, and combine the results, and then send them back to the users. In recent decades, task scheduling has attracted increased attention and has become a very challenging research field. In the process of task scheduling, users submit their jobs to the cloud scheduler, which checks the cloud information service for the status of available resources and their properties and then allocates various tasks to different resources per their requirements. The goal of scheduling is to map tasks to appropriate resources that optimize one or more objectives. Therefore, the task scheduling problem in cloud computing belongs to a category known as NP-hard problems, owing to the large solution space and the dynamic nature of heterogeneous resources [2,3,4]. Thus, it constitutes one of the crucial aspects of a resource management system in cloud computing, which ensures attainment of general quality of service in terms of response time, total execution time (makespan), and throughput, among other things. In addition, appropriate task scheduling is effective in reducing the operational costs of cloud service providers in terms of energy consumption and resource utilization.

Task scheduling problems in the cloud have been tackled using heuristic and metaheuristic algorithms.

Heuristic algorithms provide optimal solutions for small problems; but the solutions produced by these algorithms are far from optimal as the size of the problem increases [5-8].

Metaheuristic algorithms have achieved remarkable success in providing near optimal solutions for task scheduling, and they have since drawn the attention of several researchers [9-11]. However, metaheuristic algorithms still suffer from issues like entrapment in local optima, premature convergence, slow convergence, or imbalances between local and global searches.

Hence, there is scope for further development of task scheduling algorithms in the quest for improved solutions. To solve task scheduling problems now, many metaheuristic algorithms are used, such as ant colony optimization (ACO) [12-15], the genetic algorithm (GA) [18-22], particle swarm optimization (PSO) [25-27], plus variations on, and hybrids of, these methods [16,17,23-31]. A GA simulates natural evolutionary processes [20,22]; PSO simulates the behaviors of flock foraging [25, 29], and ACO imitates the foraging behavior of a real ant colony [12,15]. Recently, some researchers have proposed symbiotic organisms search (SOS) algorithms [31-34]—nature-inspired, swarm-based optimization algorithms imitating the natural symbiotic interactions between different living things. One major advantage of SOS is that it needs only one control variable (eco-size or population size) in comparison with other popular optimization techniques that surfaced earlier [31]. Also, the basic structure of the SOS algorithm is simple and easy to implement. This has made the SOS algorithm popular among many metaheuristic algorithms in recent years, and it has shown improved performance in solving different types of optimization problems [34]. Therefore, the potential for SOS to find a global solution to optimization problems exhibited so far makes it attractive for further investigation and exploration. The quality of solutions and convergence speed obtained by metaheuristic algorithms can be improved by its hybridization with either another metaheuristic algorithm or the local search method, and by generating an initial solution using heuristic search techniques or by modifying the transition operator [6-11]. To the best of our knowledge, none of the aforementioned techniques have been explored to investigate the possible improvement of SOS in terms of convergence speed and the quality of solutions obtained by SOS. In this paper, we study a new task scheduling algorithm using an improved simulated annealing (SA) chaotic local search (CLS) symbiotic organisms search (SA-CLS-SOS). The proposed SA-CLS-SOS algorithm combines the SA method [35-38] and the CLS method [39-43] with the SOS optimization algorithm. In this paper, the basic SOS algorithm is reduced, and CLS is integrated into the reduced SOS to improve the convergence rate of the basic SOS algorithm. Also, SA is combined in order to help SOS avoid being trapped in a local minimum.

The performance of the proposed SA-CLS-SOS algorithm is evaluated via extensive simulations using a Matlab simulation framework, and is compared with SOS, SA-SOS, and CLS-SOS. Simulation results show that the hybrid SOS performs better than SOS, SA-SOS, and CLS-SOS in terms of convergence speed and makespan. The main contributions of this paper are as follows.

• Clearer presentation of SOS, SA, and CLS procedures for scheduling of tasks in a cloud computing environment.

• Proposal of a new cloud task–scheduling method called SA-CLS-SOS.

• Performance comparison of the proposed hybrid method with other algorithms (SOS, SA-SOS, CLS-SOS).

• Descriptive statistical validation of the SA-CLS-SOS results against other selected methods using a significance test.

The remainder of this paper is organized as follows. Metaheuristic algorithms that have been applied to task scheduling problems in the cloud (SOS, SA, CLS) are presented in Section 2. Section 3 describes the task scheduling model in cloud computing, and detailed implementation of the improved hybrid SA-CLS-SOS algorithm for task scheduling in cloud computing is presented in Section 4. The simulation results and discussions are in Section 5. Section 6 presents the conclusion to the paper.

2. Related Work

In computing, scheduling is a method by which work specified by some means is assigned to resources that complete the work. It may be virtual computation elements, such as threads and processors, or data flows which are in turn scheduled onto hardware resources such as processors. Schedulers allow multiple users to share system resources properly, or to achieve good quality of service. Scheduling is fundamental to computation, and is an internal part of the execution model of a computer system. The concept of scheduling makes possible computer multitasking with a single central processing unit. Preference is given to any one of the concerns mentioned above, depending upon the user's needs and objectives.

Cloud computing is a model with rapid growth in recent years, increasing due to technological developments in distributed computing, grid computing and parallel computation. Task scheduling is the main problem in cloud computing. In recent decades, task scheduling has attracted increasing attention and has become a challenging research field. However, task scheduling in the cloud is an NP-hard problem, and thus, it constitutes one of the crucial aspects of a resource management system in cloud computing, which ensures attainment of general quality of service in terms of response time, total execution time (makespan), and throughput, among other things. In addition, appropriate task scheduling is effective in reducing the operational costs of cloud service providers in terms of energy consumption and resource utilization.

Task scheduling problems in the cloud have been tackled using heuristic and metaheuristic algorithms.

Heuristic algorithms provide optimal solutions for small problems, but the solutions produced by these algorithms are far from optimal as the size of the problem increases [6-8].

Metaheuristic algorithms have achieved remarkable success in providing near optimal solutions for task scheduling, and they have since drawn the attention of several researchers [9,10,11]. Metaheuristic methods have been applied to solve task assignment problems in order to reduce makespan and response time. The methods were proved able to find an optimum mapping of tasks to resources, which reduces the cost of computation, improves quality of service, and increases utilization of computing resources.

2.1 Symbiotic Organism Search algorithm

The SOS algorithm was inspired by symbiotic interactions between paired organisms in an ecosystem. Each organism denotes a potential solution to an optimization problem under consideration, and has its position in the solution space. Organisms adjust their positions according to mutualism, commensalism, or parasitism interaction models in the ecosystem. With the mutualistic form of interaction, two interacting organisms both benefit from the relationship; this is applied to the first phase of the algorithm. Commensalism is where one organism benefits from the relationship while other is not harmed. Commensalism is applied to the second phase of the algorithm to fine-tune the solution space. With parasitism, only one organism benefits while the other is harmed. Parasitism interaction is applied in the third phase of the algorithm. The fittest organisms survive in the solution space, whereas unfit ones are eliminated. The best organisms are identified as those that benefit from all three phases of the interaction. The phases of the procedure are continuously applied on the population of "organisms" that represent candidate solutions until the stopping criteria are reached. Each organism within an ecosystem is represented by a vector in the solution plane. Each organism in the search space is assigned a value that suggests the extent of adaptation to the sought objective. The algorithm repeatedly uses a population of the possible solutions to converge to an optimal position where the global optimal solution lies. The algorithm used mutualism, commensalism, and parasitism to update the positions of the solution vector in the search space. SOS is a repetitive process for an optimization problem [30], as given in Definition 2.1. The procedure keeps a population of candidate solutions to the studied problem. The relevant information concerning the decision variables and fitness values is encapsulated into the organism as an indicator of its performance. Essentially, the trajectories of the organisms are modified using the phases of symbiotic association.

Definition 2.1. Given a function $f : D \rightarrow R$ $X' \in D: \forall X \in D f(X') \leq$ or $\geq f(X)$ $\leq (\geq)$ minimization(maximization)
where f is an objective function to be optimized, and D represents the search space, while the elements of D are the feasible solutions. X is a vector of optimization variables, $X = \{x_1, x_2, x_3, \cdots, x_n\}$. An optimal solution is a feasible solution, X', that optimizes f.

The steps of the symbiotic organism search algorithm are given below.

Step 1: Ecosystem initialization

The initial population of the ecosystem is generated, and other control variables, such as ecosystem size and maximum number of iterations, are specified. The positions of the organisms in the solution space are represented by real numbers.

Step 2: Selection

The organism with the best fit objective function is represented as x^{best}.

Step 3: Mutualism phase

In the i'th iteration, an organism, x_j, is randomly selected from the ecosystem to interact with an organism, x_i, for mutual benefit, where $i \neq j$ according to (1) and (2):

$$x_i' = x_i + rand(0,1) \times (x^{best} - Mutual_{vect} \times k_1) \tag{1}$$

$$x_j' = x_j + rand(0,1) \times (x^{best} - Mutual_{vect} \times k_2) \tag{2}$$

The mutual vector is expressed as

$$Mutual_{vect} = \frac{x_i + x_j}{2} \tag{3}$$

The $rand(0,1)$ function is a vector of uniformly distributed random numbers between 0 and 1. The values of benefit factors k_1 and k_2 are determined randomly as either 1 or 2, and represent the level of benefit to each of the two organisms, x_i and x_j (where 1 and 2, respectively, denote an adequate and a huge benefit that can be received by both x_i and x_j in their current mutual symbiotic states). The organism with the best objective or fitness function value in terms of the degree of adaptation in the ecosystem is represented by x^{best}. $Mutual_{vect}$ signifies mutualistic characteristics exhibited between the two organisms to increase their survival advantage. It should be noted that any update for any one of the two organisms is computed only if its new fitness function value, denoted by $f(x_i')$ or $f(x_j')$, is better than the previous solutions, $f(x_i)$ and $f(x_j)$.

Given the above, Eqs. (1) and (2) become

$$x_i' = x_i + rand(0,1) \times (x^{best} - Mutual_{vect} \times k_1), \quad \text{if} \quad f(x_i') > f(x_i) \tag{4}$$

$$x_j' = x_j + rand(0,1) \times (x^{best} - Mutual_{vect} \times k_2), \quad \text{if} \quad f(x_j') > f(x_j) \tag{5}$$

Step 4: Commensalism phase

In this phase, organism x_i (selected randomly from the ecosystem) strives to increase its benefits from the association with x_j. This kind of symbiotic association only places x_i at an advantage over x_j, even though x_j is not harmed in the process. The new solution emanating from the symbiotic relationship is calculated as shown in Eq. (6):

$$x_i^{'} = x_i + \mathrm{rand}(-1,1) \times \left(x^{best} - x_j\right) \quad \text{if} \quad f\left(x_i^{'}\right) > f(x_i) \tag{6}$$

Step 5: Parasitism phase

In the i'th iteration, a parasite vector, x^p, is created by mutating x_i using a randomly generated number in the range of the decision variables under consideration, and organism x_i with $i \neq j$ is selected randomly from the population to serve as host to x^p. If the fitness value $f(x^p)$ is greater than $f(x_j)$, then x^p will replace x_j; otherwise, x^p is discarded.

Steps 2 through 5 are repeated until the stopping criterion is reached.

Step 6: Stopping criterion

The pseudocode of SOS is presented in Algorithm 1.

Algorithm 1. Symbiotic Organism Search Algorithm

Create and initialize the population of organisms in ecosystem $X = \{x_1, x_2, x_3, \cdots, \boldsymbol{x_N}\}$
Set up stopping criterion
iteration_number$\leftarrow \mathbf{0}$
$\boldsymbol{x^{best}} \leftarrow \mathbf{0}$
Do
    $\boldsymbol{iteration\_number} \leftarrow \boldsymbol{iteration\_number} + \mathbf{1}$
    $\boldsymbol{i} \leftarrow \mathbf{0}$
        Do
            $\boldsymbol{i} \leftarrow \boldsymbol{i} + \mathbf{1}$
            For $\boldsymbol{j} = \mathbf{1}\ \boldsymbol{to}\ \boldsymbol{N}$
                $\boldsymbol{If}\ \ \boldsymbol{f}\left(\boldsymbol{x_j}\right) > \boldsymbol{f}\left(\boldsymbol{x^{best}}\right)\ \boldsymbol{Then}\ //\ \boldsymbol{f}(\boldsymbol{x})\ \boldsymbol{is\ fitness\ function}$
                    $\boldsymbol{x^{best}} \leftarrow \boldsymbol{x_j}$
                End if
            End for
//mutualism phase
Randomly select $\boldsymbol{x_j}\ \boldsymbol{with}\ \boldsymbol{i} \neq \boldsymbol{j}$
  $\boldsymbol{k_1} \leftarrow \mathbf{1}\ \boldsymbol{or}\ \mathbf{2}$
  $\boldsymbol{k_2} \leftarrow \mathbf{1}\ \boldsymbol{or}\ \mathbf{2}$
$\boldsymbol{Mutual_{vect}} = \frac{\boldsymbol{x_i}+\boldsymbol{x_j}}{\mathbf{2}}$
 $\boldsymbol{x_i^{'}} = \boldsymbol{x_i} + \boldsymbol{rand}(\mathbf{0},\mathbf{1}) \times (\boldsymbol{x^{best}} - \boldsymbol{Mutual_{vect}} \times \boldsymbol{k_1})$
 $\boldsymbol{x_j^{'}} = \boldsymbol{x_j} + \boldsymbol{rand}(\mathbf{0},\mathbf{1}) \times (\boldsymbol{x^{best}} - \boldsymbol{Mutual_{vect}} \times \boldsymbol{k_2})$
$\boldsymbol{If}\ \ \boldsymbol{f}\left(\boldsymbol{x_i^{'}}\right) > \boldsymbol{f}(\boldsymbol{x_i})\ \ \boldsymbol{Then}$

$\boldsymbol{x_i \leftarrow x_i'}$
End if

$\boldsymbol{If\ \ f(x_j') > f(x_j)\ Then}$

$\boldsymbol{x_j \leftarrow x_j'}$

End if
//commensalism phase
Randomly select $\boldsymbol{x_j\ with\ i \neq j}$
$\boldsymbol{x_i' = x_i + rand(-1,1) \times (x^{best} - x_j)}$
$\boldsymbol{if\ \ f(x_i') > f(x_i)}$ The
$\boldsymbol{x_i \leftarrow x_i'}$
End if
//parasitism phase
Randomly select $\boldsymbol{x_j\ with\ i \neq j}$
Create parasite vector $\boldsymbol{x^p}$ from $\boldsymbol{x_i}$ using random number
$\boldsymbol{If\ \ f(x^p) > f(x_i)\ Then}$
$\boldsymbol{x_j \leftarrow x^p}$
End if
While $\boldsymbol{i <= N}$
While stopping condition is not true

The SOS algorithm is thought to be efficient at solving complex optimization and discrete engineering problems, but it still has a high probability of plunging to the local optimum [30]. Therefore, the SOS-SA algorithm was proposed to overcome this shortcoming.

2.2 Simulated annealing algorithm

Simulated annealing is used to further process the result from SOS to avoid falling into the local optimal solution [33,34].The process begins by considering a solution space, $\boldsymbol{S}$, of a particular tour through the set of given cities or points, $\boldsymbol{x_i | i = 1, 2, \cdots, n}$, with update solutions $\boldsymbol{x_i'}$ created by randomly switching the orders of two cities. The energy function or fitness function, which represents the length of route $\boldsymbol{x_i}$, is denoted by $\boldsymbol{f(x_i)}$. The relative change in cost, $\boldsymbol{\Delta f}$, between $\boldsymbol{x_i}$ and $\boldsymbol{x_i'}$ is expressed as $\boldsymbol{\Delta f = \frac{f(x_i') - f(x_i)}{f(x_i)}}$. Beginning with the initial solution, only the solution that results in a smaller energy value than the previous solution is accepted by the algorithm; in other words, a solution is only accepted with a fitness value of $\boldsymbol{f(x_i') < f(x_i)}$. However, accepting or rejecting a new solution with higher fitness values for $\boldsymbol{x'}$ can be based on the acceptance probability function, given as follows:

$$\boldsymbol{P}(\Delta \boldsymbol{f}, \boldsymbol{T_k}) = \begin{cases} \boldsymbol{e}^{\left(\frac{-\Delta f}{T_k}\right)}, & \Delta \boldsymbol{f} > \boldsymbol{0} \\ \boldsymbol{1}, & \Delta \boldsymbol{f} \leq \boldsymbol{0} \end{cases} \quad \text{for } \boldsymbol{T_k} > \boldsymbol{0} \tag{7}$$

where $\boldsymbol{T_k}$ is the parameter temperature at the $\boldsymbol{k^{th}}$ instance of accepting a new solution route, and for any given T, for $\Delta \boldsymbol{f} > \boldsymbol{0}$, P is greater for smaller values of $\Delta \boldsymbol{f}$, which means that, for the new solution $\boldsymbol{x_i'}$ that is only slightly more costly than the current solution, $\boldsymbol{x_i}$ is more likely to be accepted than a new solution $\boldsymbol{x_i'}$ that is much more costly than current solution $\boldsymbol{x_i}$. The value of T, which is an important control parameter, decreases in proportion to P ; that is, as $\boldsymbol{lim_{T \to 0+}} \boldsymbol{e}^{(\frac{-\Delta f}{T_k})} = \boldsymbol{0}, \Delta \boldsymbol{f} > \boldsymbol{0}$. Therefore, as the value of T decreases, the probability of accepting a degraded route also decreases. In this paper, the following cooling schedule is adopted:

$$\boldsymbol{T_{k+1}} = \boldsymbol{\alpha T_k} \tag{8}$$

where $\boldsymbol{\alpha}$ denotes the cooling coefficient, which is some random constant value between 0 and 1, and it is also the rate at which the temperature is lowered each time new solution $\boldsymbol{x_i'}$ is discovered. The SA procedure is presented in Algorithm 2.

Algorithm 2. Pseudocode for SA.
Input : Initial temperature $\boldsymbol{T_0}$, final temperature $\boldsymbol{T_k}$, cooling rate α, maximum iteration maxiter Output : Best cost 1: Chose a random route $\boldsymbol{x_i}$ and initialize $\boldsymbol{T_0}$T 0 and α 2: For counter = 1 to maxiter 3: Create a new solution $\boldsymbol{x_i'}$ by randomly swapping two cities in neighborhood of $\boldsymbol{x_i}$ 4: Compute $\Delta \boldsymbol{f} = \frac{\boldsymbol{f}(\boldsymbol{x_i'}) - \boldsymbol{f}(\boldsymbol{x_i})}{\boldsymbol{f}(\boldsymbol{x_i})}$ and use the acceptance probability function to either accept or reject the new solution, based on the following conditions: a) if $\Delta \boldsymbol{f} \leq \boldsymbol{0}$, $\boldsymbol{then}\ \boldsymbol{x_i} \leftarrow \boldsymbol{x_i'}$ b) if $\Delta \boldsymbol{f} > \boldsymbol{0}, \boldsymbol{then}\ \boldsymbol{x_i} \leftarrow \boldsymbol{x_i'}$ depending on Eq. (7) 5: Reduce the temperature using Eq. (8) and increment k 6: Update the best solution 7: End for

2.3. Chaotic local search algorithm

Chaos is a deterministic process that is usually found in dynamic and nonlinear systems; it has high sensitivity to initial conditions and parameter changes. Chaos is characterized by randomness, ergodicity, irregularity, and apparent unpredictability. Chaos is known as randomness in a simple dynamic system, which motivates its usage as a source of randomness in optimization theory and other various fields instead of the usual random process. Chaotic sequences have been employed in stochastic optimization techniques to provide population diversity in a search space to ensure global convergence, as well as to avoid local optima entrapment. Chaotic sequences are highly sensitive to their initial values. It is quite important to select the initial value for the chaotic map very precisely. In chaotic PSO (CPSO) [39], the decision variables of PSO are mapped into the chaotic domain with Eq. (9):

$$\boldsymbol{cx_i} = (\boldsymbol{x} - \boldsymbol{x_{min}})/(\boldsymbol{x_{max}} - \boldsymbol{x_{min}}) \tag{9}$$

where $\boldsymbol{cx_i}$ is the initial value of the chaotic sequence, x is the position of the particle, and $\boldsymbol{x_{max}}$ and $\boldsymbol{x_{min}}$ are the search boundaries. But this mapping may lead to ineffectiveness of the chaotic search as the initial value of the chaotic sequence becomes fixed, and hence, the whole chaotic orbit becomes monotonous. CLS is activated when the best solution, obtained by PSO over the entire population, does not change for several times [40]. In this case, u becomes fixed, and hence, the chaotic search orbit will always be the same before the next chaotic search. This will worsen the performance of the chaotic search. To avoid this problem and to maintain the ergodicity of the chaotic search, Saha and Mukherjee suggested usage of a random function to generate the initial value of the chaotic sequence [40]. So, the initial value of the chaotic sequence is

$$\boldsymbol{cx_i} = \boldsymbol{rand}(\mathbf{0}, \mathbf{1}) \tag{10}$$

As chaotic search is most efficient in a small range [38], CLS in the proposed CSOS of the present work is performed over a small radius, r. CLS is only applied to the best organism ($\boldsymbol{x_{best}}$) as achieved after the commensalism phase of the reduced SOS optimizer. This is because ($\boldsymbol{x_{best}}$−r, $\boldsymbol{x_{best}}$ +r) would be the most promising range for the local search. Moreover, it saves more time, compared with the methods that apply chaotic search on all the particles. Chaotic search radius r is defined initially by Eq. (11), and then, it is subsequently decreased in the next generations with the help of a shrinking coefficient, $\boldsymbol{\delta}(\mathbf{0} < \boldsymbol{\delta} < \mathbf{1})$, to shrink the search area [34]:

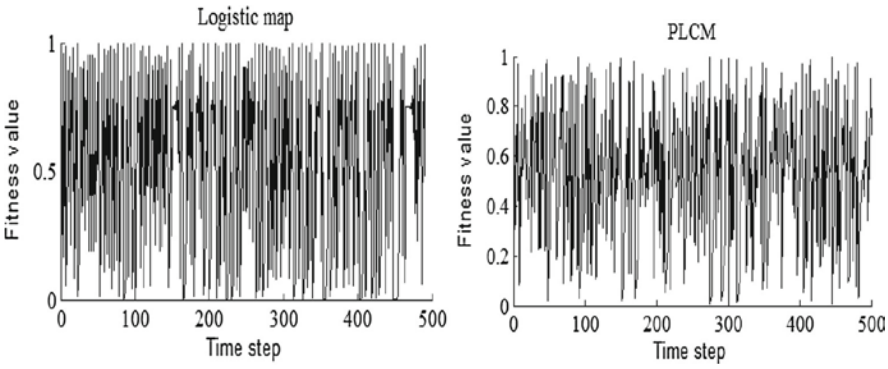
$$\boldsymbol{r} = (\boldsymbol{x_{max}} - \boldsymbol{x_{min}})/\mathbf{2} \tag{11}$$

The initial variable of the chaotic sequence (that is, $\boldsymbol{cx_i}$) is generated by using Eq. (10), and the next variable of that chaotic sequence (i.e. $\boldsymbol{cx_{i+1}}$) is generated by using Piecewise Linear Chaotic Map (PLCM). PLCM is formulated in Eq. (12) [38]:

$$cx_{i+1} = \begin{cases} \frac{cx_i}{q} & cx_i \in (0, q) \\ \frac{(1-cx_i)}{(1-q)} & cx_i \in (q, 1) \end{cases} \tag{12}$$

where q is the control parameter (q∈ 0, 0.5).

The distributions of two different chaotic maps are shown in **Fig. 1** over 500 time steps (Fig. 1a is a logistic map, and Fig. 1b is the PLCM [40]). The chaotic map that is used here to generate the chaotic sequence is the simple PLCM. PLCM is ergodic in nature (see **Fig. 1b**) and has a uniform invariant density function. It is easy to implement, and it depicts very good dynamic behavior, which makes it superior to the well-known logistic map (**Fig. 1a)** [40].



(a) (b)

Fig. 1. Distribution of chaotic maps for (a) a logistic map, and (b) PLCM.

Using Eq. (13), the chaotic variables generated by PLCM are mapped back to the search range around the best organism:

$$x_{i+1} = x_{best} + r(2cx_{i+1} - 1) \tag{13}$$

where x_{i+1} is the position of the best organism over the entire population at the (i +1)th generation of CLS, and x_{best} is the position of the best organism in the ecosystem after the traditional SOS. The fitness value is calculated for organism x_{i+1}, and it will be considered the best organism if it provides better fitness than the previous best organism. The CLS procedure is presented in Algorithm 3.

Algorithm 3. Chaotic Local Search Pseudocode

Set i=0
Initialize chaotic variable $\boldsymbol{cx_i} = \boldsymbol{rand}(\mathbf{0},\mathbf{1})$
Set chaotic search radius r with Eq. (11)
do
  Calculate $\boldsymbol{cx_{i+1}}$ by (12)
  Map $\boldsymbol{cx_{i+1}}$ back to the range around the best organism using $\boldsymbol{x_{i+1}} = \boldsymbol{x_{best}} + \boldsymbol{r}(\mathbf{2}\boldsymbol{cx_{i+1}} - \mathbf{1})$
  Evaluate fitness value for $\boldsymbol{x_{i+1}}$
  while a better solution is found, or the maximum number of iterations is reached
Decrease the radius of the chaotic search space by $\boldsymbol{r} = \boldsymbol{\delta} \times \boldsymbol{r}$
// $\boldsymbol{\delta}$ is a random number between 0 and 1

3. Task Scheduling Model in Cloud Computing

To simplify the complexity of the problem and establish an effective task scheduling model, we make the following assumptions. Tasks submitted by the users are indivisible meta-tasks; furthermore, each task is an independent operation and does not run on priority; the number of tasks submitted by users in cloud computing is far greater than for virtual machines in a cloud datacenter; the execution time of tasks in a virtual machine (VM) can be calculated according to the information processing speed, in millions of instructions per second (MIPS). To establish a mathematical model of facilitated task scheduling, we established the related parameters of the tasks and the virtual machines as follows.

Task set $\mathrm{T} = \{\mathrm{Task_1}, \mathrm{Task_2}, \mathrm{Task_3}, \cdots, \mathrm{Task_i}, \cdots, \mathrm{Task_m}\} = \{\mathrm{Task_i} | \mathrm{i} > 0, i \in [1, \mathrm{m}]\}$, where m is the number of tasks submitted by the users. $\mathrm{Task_i}$ represents the i'th task in the task sequence. The feature of $\mathrm{Task_i}$ is defined as $\{\mathrm{TK_i}, \mathrm{task\_length_i}, \mathrm{Time\_exp_i}, \mathrm{P_i}\}$, in which $\mathrm{TK_i}$ is the serial number of tasks, and $\mathrm{task\_length_i}$ is the instruction length of the task in millions of instructions (MI). $\mathrm{Time\_exp_i}$ refers to the user's expected completion time for $\mathrm{Task_i}$, and $\mathrm{P_i}$ refers to the task priority.

The VM set is $\mathrm{VM} = \{\mathrm{vm_1}, \mathrm{vm_2}, \mathrm{vm_3}, \cdots, \mathrm{vm_j}, \cdots, \mathrm{vm_n}\} = \{\mathrm{vm_j} | \mathrm{j} > 0, j \in [1, n]\}$, where n is the number of virtual machines, and $\mathrm{vm_j}$ denotes the j'th virtual machine resource in the cloud environment. The feature of $\mathrm{vm_j}$ is defined as $\{\mathrm{VM_j}, \mathrm{MIPS_j}\}$, in which $\mathrm{VM_j}$ is the serial number of the virtual machine, and $\mathrm{MIPS_j}$ is the information processing speed in MIPS of the virtual machine.

The tasks are scheduled on the available VMs, and execution of the tasks is done on a first-come first-served basis. Our aim is to schedule tasks on VMs in order to achieve high utilization with a minimal makespan. As a result, the expected time to compute (ETC) of the tasks to be scheduled on each VM will be used by the proposed method to make scheduling decisions. ETC values were determined using the ratio of the MIPS of the VM to the length of the task.

ETC values are usually represented in matrix form, as follows:

$$\mathrm{ETC} = \begin{pmatrix} \mathrm{ETC}_{11} & \cdots & \mathrm{ETC}_{1n} \\ \vdots & \ddots & \vdots \\ \mathrm{ETC}_{m1} & \cdots & \mathrm{ETC}_{mn} \end{pmatrix} \tag{14}$$

where the number of tasks to be scheduled appears in the rows of the matrix, and the number of available VMs appears in the columns of the matrix. Each row of the ETC matrix represents execution times of the given tasks for each VM, while each column represents execution times of the tasks on a given VM. Our objective is to minimize the makespan by finding the best group of tasks to be executed on VMs.

Let $\mathrm{ETC}_{ij}, i = 1,2,\cdots,m, j = 1,2,\cdots,n$ be the execution time of executing the i'th task on the j'th VM.

Then ETC_{ij} is calculated as follows:

$$\mathrm{ETC}_{ij} = \mathrm{task\_length}_i / \mathrm{MIPS}_j \tag{15}$$

The fitness value of each organism is determined using Eq. (16), which determines the strength of the level of adaptation of the organism to the ecosystem:

$$\text{objective function} = \max\left\{\sum_{j=1}^{n} \frac{f(M_j)}{n}\right\} \tag{16}$$

$$f(M_j) = \frac{\mu}{\mathrm{makespan}} \tag{17}$$

$$\mu = \sum_{j=1}^{n} \frac{\lambda_j}{n} \tag{18}$$

$$\lambda_j = \frac{\mathrm{Task}_j}{\mathrm{makespan}} \tag{19}$$

$$\mathrm{makespan} = \max\left\{\mathrm{ETC}_{ij} \middle| i \in T, i = 1,2,3,\cdots,m; j \in VM, j = 1,2,3,\cdots,n\right\} \tag{20}$$

In Eq. (17), $f(M_j)$ is the fitness value of virtual machine j, and μ is the average utilization of virtual machines ready for the execution of tasks. The essence is to support load balancing among VMs, so λ_j defines the utilization of virtual machine j.

For the degree of imbalance, let T_{max}, T_{min}, and T_{avg} denote the sums of the maximum, minimum, and average execution times, respectively, for all VMs. The degree of imbalance (DI) defines the extent of the load distribution among the VMs according to their processing capacities, and is determined with Eq. (21):

$$DI = \frac{T_{max} - T_{min}}{T_{avg}} \tag{21}$$

4. Hybrid SA-CLS-SOS Algorithm for task Scheduling in Cloud Computing

The SA-CLS-SOS algorithm is a hybrid of symbiotic organisms search, simulated annealing, and chaotic local search. CLS is employed after the commensalism phase, replacing the parasitism phase of SOS. In the mutualism phase, two new candidate solutions are generated, whereas during commensalism, one new candidate solution is generated based on the previous best solution or organism in the ecosystem. In both the mutualism and commensalism phases, the new candidate solutions or organisms are accepted if they have better fitness values than the previous best organism, and these newly generated organisms direct the search process over the unvisited portion of the entire search space. In short, the mutualism and commensalism phases provide better exploration of the search space. On the other hand, in the parasitism phase, the current best organism from the commensalism phase is duplicated to act as a parasite vector, and it interacts with a randomly chosen organism from the ecosystem. If the randomly chosen organism has a better fitness value than the parasite vector, it will remain in the ecosystem; otherwise, it will be destroyed. This may lead to loss of a potential solution in case of any improper duplicating of a parasite vector or any ineffective interaction that cannot produce a better solution over a number of generations. This will affect computational efficiency and will take an unnecessarily longer computation time. In contrast, with CLS, the search process is intensified towards a promising region that enhances the exploitation of the search space. As a result, a better solution may be found more quickly. Also, SA is a local-search metaheuristic algorithm widely used for solving both discrete and continuous optimization problems. One of the main benefits of SA lies in its ability to escape the problem of getting stuck in a local minimum by allowing hill-climbing moves to search for a global solution. Therefore, the hybrid approach is proposed by introducing SA to assist SOS in avoiding being trapped in a local minimum, and to increase its level of diversity while searching for the optimum solution in the problem search space. Thus, the new hybrid algorithm (SA-CLS-SOS) is proposed to improve task scheduling optimization in cloud computing.

The steps of the hybrid SA-CLS-SOS algorithm are described in Algorithm 4.

Algorithm 4. SA-CLS-SOS Pseudocode

Input: Initial ecosystem x , ecosystem size eco _ size, initial temperature $\mathbf{T_0}$, final temperature $\mathbf{T_k}$,
cooling rate $\boldsymbol{\alpha}$, maximum iteration maxiter
Initialize chaotic variable $\boldsymbol{cx_i = rand(0,1)}$; set chaotic search radius r with Eq. (11)
Output: best known solution $\boldsymbol{x_{best}}$
1: Create and evaluate new solutions
Generate $\boldsymbol{x_i}$, i = 1 , 2 , . . . , eco _ size
For i = 1 to maxiter
a) Compute cost / fitness function of $\boldsymbol{x_i}$, $\boldsymbol{f(x_i)}$
b) Determine best solution $\boldsymbol{x_{best}}$
d) Compute $\Delta \boldsymbol{f} = \frac{\boldsymbol{f(x_i')} - \boldsymbol{f(x_i)}}{\boldsymbol{f(x_i)}}$
If $\Delta \boldsymbol{f} \leq \mathbf{0}\ \boldsymbol{or}\ \boldsymbol{p} > \boldsymbol{u}$, where p is the acceptance probability from Eq. (7), and u is a random number
between 0 and 1
then update solution by assigning $\boldsymbol{x_{best}} \leftarrow \boldsymbol{x_i}$
End if
For i = 1 to eco _ size
2: Update organism (route) with SA (Algorithm 2) on the two SOS phases in Algorithm 1
For i =1 to eco _ size
a) Modify the organisms according to (1) and (2) in mutualism phase
b) Modify organism $\boldsymbol{x_i}$ with the help of $\boldsymbol{u_j}$ using (6) in commensalism phase
c) Update best organism $\boldsymbol{x_{best}}$
3: Update best organism $\boldsymbol{x_{best}}$ using CLS in Algorithm 3
do
Calculate $\boldsymbol{cx_{i+1}}$ by (12)
Map $\boldsymbol{cx_{i+1}}$ back to the range around the best organism using
$\boldsymbol{x_{i+1} = x_{best} + r(2cx_{i+1} - 1)}$
Evaluate fitness value for $\boldsymbol{x_{i+1}}$
while a better solution is found or maximum number of iterations is reached
Decrease the radius of the chaotic search space by $r = \delta \times r$
4: Update the best x_{best} ever found
5: Update temperature using the cooling schedule given in Eq. (8)
5: End for
6: End for
7: End for

The hybrid algorithm is initialized by random solutions and searches for the optimization solution by the search process intensified towards a promising region that enhances the exploitation of the search space. Also, the hybrid algorithm escapes the problem of getting stuck in a local minimum by allowing hill-climbing moves to search for a global solution. During this course, an evolution of this solution is performed by integrating SA, CLS, and SOS.

The SOS optimization strategy is performed in three search-and-update phases (i.e., mutualism, commensalism, and parasitism) as presented subsequently. In Algorithm 4, step 1 is SA. The SA technique is employed in the solution search procedure of the mutualism and commensalism phases of SOS. This procedure is presented in step 2. Then, CLS is employed after the commensalism phase, replacing the parasitism phase of SOS. In step 3, this procedure is presented.

5. Simulation and Results

In order to test the performance of the proposed method, simulations were carried out using the Matlab R2017a_win64 computing environment on a 3.2 GHz core i5 personal computer with 4 GB of random access memory (RAM).

One datacenter was created containing two hosts. Each host had 20 GB RAM, 1 TB storage, 10 GBps bandwidth, and a time-shared VM scheduling algorithm. One host was a dual-core machine, while the other was a quad-core machine, each with the X86 architecture, a Linux operating system, a Xen virtual machine monitor (VMM), and cumulative processing power of 1,000,000 MIPS. Twenty VMs were created, each with an image size of 10 GB, with 0.5 GB memory, 1 GBps bandwidth, and one processing element. The processing power of the VMs ranged from 100 MIPS to 5000 MIPS. A time-shared cloudlet scheduler and the Xen VMM were used for all the VMs. Task sizes were generated in a uniform distribution, which depicts an equal number of large, medium-size, and small tasks. For the uniform distribution, 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 instances were generated. The larger instances enabled us to gain insight into the scalability of the performance of the algorithms with large problem sizes.

The first experiment was carried out for SA-CLS-SOS, SOS, SA-SOS, and CLS-SOS to evaluate the makespan of the proposed algorithm. The parameter settings of the algorithms are shown in **Table 1**. The comparison results are presented in **Fig. 2** and **Table 2**. The second experiment was carried out to evaluate the degree of imbalance. The results are presented in **Fig. 3** and **Table 3.** The third experiment was carried out to evaluate the quality of the solutions of the SA-CLS-SOS algorithm based on makespan. The results are presented in **Fig. 4 to Fig. 6**.

Table 1. Parameter settings

Algorithm	Parameter	Value
SOS	Number of eco _ sizes	100
	Number of iterations	1000
SA	Initial temperature, T_0	10
	Final temperature, T_k	0.001
	Cooling rate, α	0.9
CLS	Control parameter, p	0.05
	Search boundaries: x_{max}	1.2
	x_{min}	0.2

In order to compare the performance of the proposed SA-CLS-SOS against SOS, SA-SOS, and CLS-SOS, graphs for solution quality, makespan, and response time were plotted against the number of iterations for task sizes from 100 to 1000. **Fig. 2** show the average makespan when executing a task instance 10 times using SOS, SA-SOS, CLS-SOS, and SA-CLS-SOS.

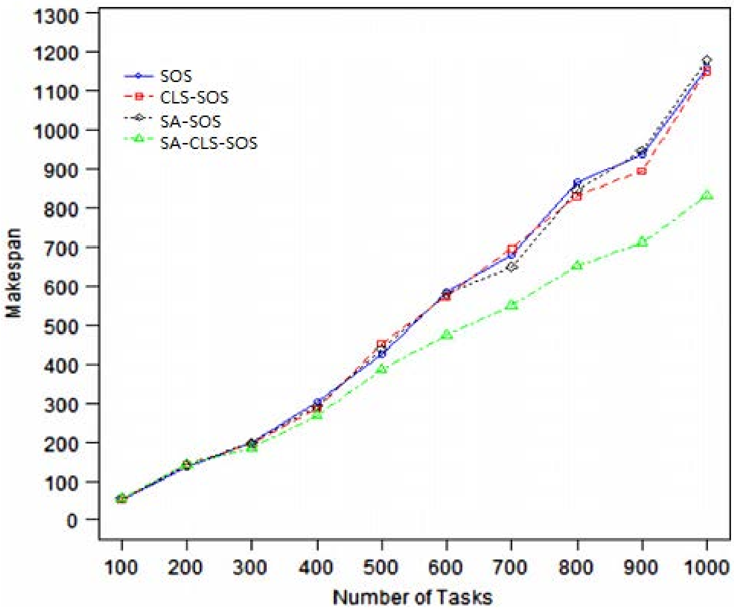


Fig. 2. Makespan comparison between SOS, SA-SOS, CLS-SOS, and SA-CLS-SOS

The figure indicates minimization of makespan when using SA-CLS-SOS, particularly from task instances of 300 upward. For makespan, the percentage improvement with SA-CLS-SOS over SA-SOS is summarized in **Table 2**, showing that the degree of performance of SA-CLS-SOS over SA-SOS increases as the search space increases.

Table 2. Makespan comparison between SA-SOS and SA-CLS-SOS

Number of Tasks	SA-SOS			SA-CLS-SOS			Improvement (%)
	Average	Worst	Best	Average	Worst	Best	
100	312.21	438.57	215.73	278.02	297.00	196.12	10.95
200	807.07	1115.35	507.72	734.91	817.32	522.65	8.94
300	1470.53	1904.61	859.45	1375.71	1484.38	1035.98	6.45
400	2334.28	3309.66	1654.35	1976.99	2168.68	1423.27	15.31
500	3263.91	4396.75	2244.82	2926.87	3099.04	2092.69	10.33
600	4201.65	5391.53	3094.80	3698.41	3847.80	2893.99	11.98
700	5023.15	6118.88	3561.55	4679.92	4956.90	3744.04	6.83
800	6157.11	8487.05	4369.69	5430.88	5800.05	3501.86	11.79
900	7106.45	8446.11	4874.59	6494.58	6825.56	5381.45	8.61
1000	8095.45	9880.74	6171.13	7641.47	7942.74	6680.93	5.61

SA-CLS-SOS also gives a better degree of imbalance among VMs for large problem instances, as can be observed in **Fig. 3**.

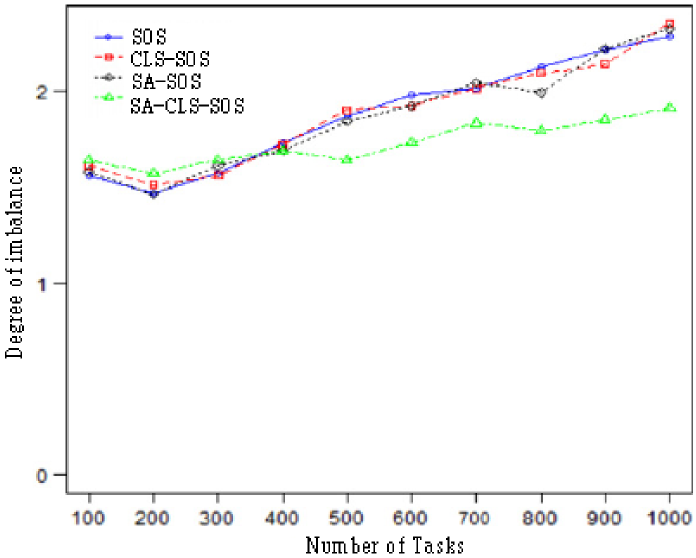


Fig. 3. Degree of imbalance

For the degrees of imbalance, a statistical analysis of SA-SOS and SA-CLS-SOS under different task sizes is presented in **Table 3.** As a result, SA-CLS-SOS produced a better degree of imbalance among VMs, compared to SA-SOS for all task sizes.

Table 3. Comparison of degree of imbalance obtained by SA-SOS and SA-CLS-SOS

Number of Tasks	SA-SOS			SA-CLS-SOS			Improvement (%)
	Average	Worst	Best	Average	Worst	Best	
100	10.94	17.58	10.71	8.47	20.08	11.04	22.61
200	22	43.28	22.71	14.81	41.38	21.59	32.67
300	38.45	66.91	41.26	28.7	62.57	40.68	25.37
400	50.24	86.57	53.19	31.19	86.84	52.58	37.91
500	67.13	103.88	73.45	54.62	112.19	77.6	18.64
600	76.1	120.36	80.93	53.28	121.42	80.47	29.99
700	103.03	148.15	108.92	66.26	150.92	104.74	35.69
800	111.35	167.11	121.6	70.69	174.31	101.93	36.51
900	133.56	191.2	139.55	98.65	181.62	143.7	26.14
1000	139.3	196.2	149.77	120.5	218.16	163.06	13.49

Convergence graphs showing improvement in the quality of solutions for makespan obtained by SA-SOS and SA-CLS-SOS using data instances of 100, 500, and 1000 are presented in **Fig. 4 to Fig. 6**.

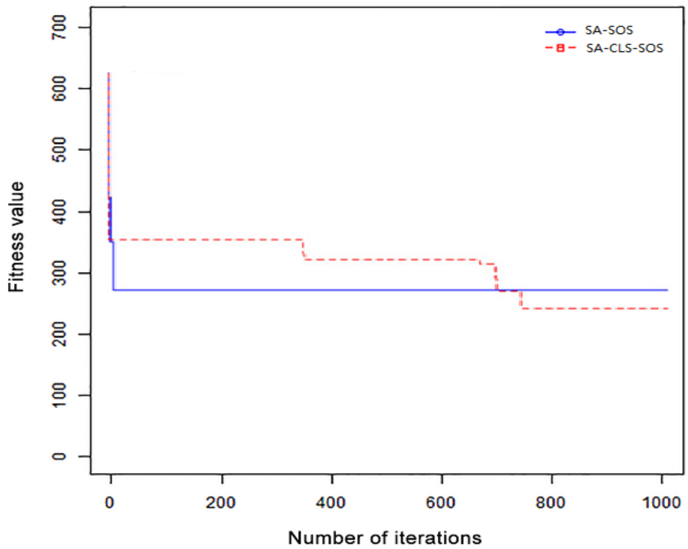


Fig. 4. Convergence graph (100 tasks)

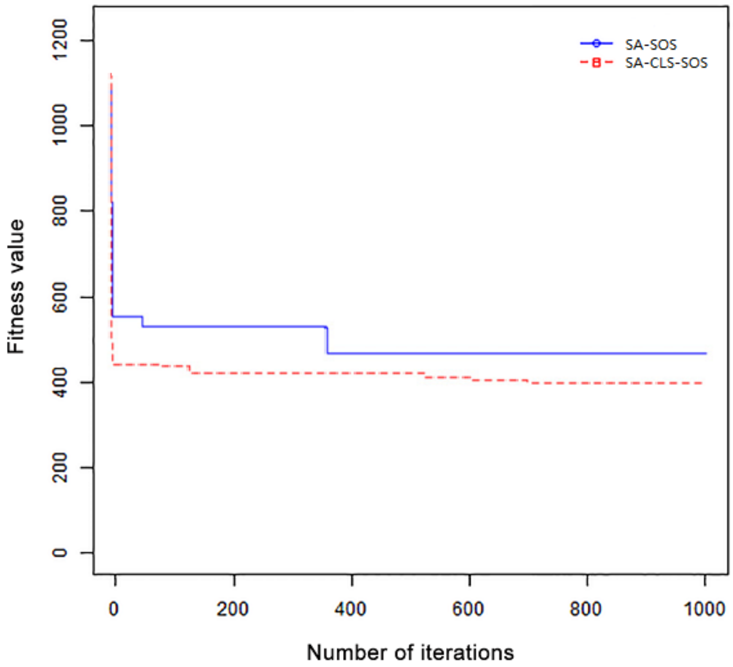


Fig. 5. Convergence graph (500 tasks)

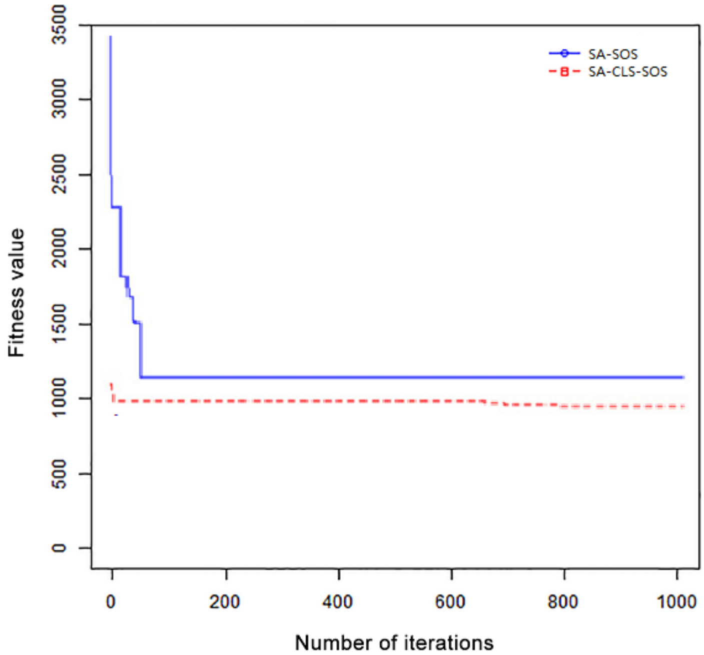


Fig. 6. Convergence graph (1000 tasks)

As can be seen, both methods showed improvement in the quality of solutions at the beginning of the search, but SA-CLS-SOS demonstrated the ability to improve the quality of solutions at a later stage of the search process. The quality of solutions obtained by SA-CLS-SOS is better than with SA-SOS, especially when the problem size is large. As can be seen from the figures, SA-CLS-SOS obtains the lowest makespan, and the quality of the solutions obtained by the SA-CLS-SOS algorithm is better than SOS, SA-SOS, and CLS-SOS. That is, the search direction of SA-CLS-SOS tends to converge to a stable point in fewer iterations. The method is able to improve quality even at a later stage of the search process, which means that SA-CLS-SOS has a higher probability of obtaining a near-optimal solution than SA-SOS.

6. Conclusion

This paper presents a novel SA-CLS-SOS algorithm to decrease makespan and improve the quality of solutions for task scheduling optimization problems in cloud computing. The proposed algorithm employs simulated annealing and a chaotic local search ability in order to improve the speed of convergence and the quality of solutions obtained by the SOS algorithm in terms of makespan. According to the simulation results, SA-CLS-SOS performs better than SOS, SA-SOS, and CLS-SOS in terms of the quality of the solutions obtained and makespan. The proposed method can be used to solve other optimization issues in cloud computing systems and other discrete optimization problems in different domains.

References

[1] XiaoLi He, Yu Song and Ralf Volker Binsack, “The Intelligent Task Scheduling Algorithm in Cloud Computing,” *International Journal of Grid and Distributed Computing*, 9(4), pp. 313-324, April, 2016. Article (CrossRef Link)

[2] S. Balamurugan, Dr.P.Visalakshi, "Strategies for Solving the NP-Hard Workflow Scheduling Problems in Cloud Computing Environments," *Australian Journalof Basic and Applied Sciences*, 8(16), pp. 345-355, October, 2014. http://ajbasweb.com/old/ajbas/2014/October/345-355.pdf

[3] SM Abdulhamid，MS Abd Latiff，G Abdul-Salaam，SH Hussain Madni, "Secure Scientific Applications Scheduling Technique for Cloud Computing Environment Using Global League Championship Algorithm," *Plos One*, 11(7), pp. 1-18, July 12, 2016. Article (CrossRef Link)

[4] T Mathew，KC Sekaran，J Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," *ICACCI*, pp. 658-664, December 2014. Article (CrossRef Link)

[5] F Nzanywayingoma and Y Yang, " Effective Task Scheduling and Dynamic Resource Optimi zation based on Heuristic Algorithms in Cloud Computing Environment," *KSII Transactions on Internet & Information Systems*, 11(12), pp. 5780-5802, December, 2017. Article (CrossRef Link)

[6] Z Wu, X Liu, Z Ni and Y Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *Journal of Supercomputing*, 63(1), pp. 256-293, January, 2013. Article (CrossRef Link)

[7] K Kurowski and A Oleksiak, "Hierarchical scheduling strategies for parallel tasks and advance reservations in grids," *Journal of Scheduling*, 16 (4), pp. 349-368, August, 2013. Article (CrossRef Link)

[8] P Huang， H Peng, P Lin and X Li, "Static strategy and dynamic adjustment: An effective method for Grid task scheduling," *Future Generation Computer Systems*, 25(8), pp. 884-892, September, 2009. Article (CrossRef Link)

[9] Kalra Mala and Singh Sarbjeet, "A review of metaheuristic scheduling techniques in Cloud computing," *Egyption Informatics Journal*, vol. 16, no. 3, pp. 275-295, August, 2015. Article (CrossRef Link)

[10] Young-Choon Lee and Albert Zomaya, "A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems," *IEE Transactions on Parallel and Distributed Systems*, 19(9), pp. 1215-1223, September, 2008. Article (CrossRef Link)

[11] R. Maheswaran and S.G. Ponnambalam, "A meta-heuristic approach to single machine scheduling problems," *The International Journal of Advanced Manufacturing Technology*, 25(7-8), pp. 772–776, April, 2005. Article (CrossRef Link)

[12] U Jaiswal and S A Ggarwal, "Ant Colony Optimization," *International Journal of Scientific & Engineering Research*, 2(7), pp. 2229-5518, July, 2011. https://www.ijser.org/researchpaper/ant_colony_optimization.pdf

[13] M edhat Tawfeek, Arabi Keshk, Ashraf EI-Sisi and Fawzy A. Torket, "Cloud Task Scheduling Based on Ant Colony Optimization," *INTERNATIONAL ARAB JOURNAL OF INFORMATION TECHNOLOGY*, 12(2), pp. 64-69, November, 2013. Article (CrossRef Link)

[14] Gao Ying, Duan Jiajie and Shu Wanneng, "A Novel Ant Optimization Algorithm for Task Scheduling and Resource Allocation in Cloud Computing Environment," *JOURNA OF INTERNET TECHNOLOGY*, 16(7), pp. 1329-1338, January, 2015. Article (CrossRef Link)

[15] LI Li-Fen, YL Zhu and JY Zhang, "A cloud model based multiple ant colony algorithm for the routing optimization of WSN with a long-chain structure," *Comput. Eng. Sci*, 32(11), pp. 10-14, November, 2010. http://en.cnki.com.cn/Article_en/CJFDTOTAL-JSJK201011002.htm

[16] Y Gao, H Guan, Z Qi, Y Hou and L Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci*, 79(8), pp. 1230–1242, December, 2013. Article (CrossRef Link)

[17] Raju, R et al, "Minimizing the makespan using hybrid algorithm for cloud computing," *Adv. Comput. Conf*, 7903, pp. 957–962, February, 2013. Article (CrossRef Link)

[18] Zhang Nan, Yang Xiaolong, Zhang Min and Long Keping, "A genetic algorithm-based task scheduling for cloud resource crowd-funding model," *INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS*, 31(1), September, 2017. Article (CrossRef Link)

[19] Y Xu, K Li, J Hu and K Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci*, 270(6), pp. 255–287, June, 2014. Article (CrossRef Link)

[20] YS Jiang and WM Chen, "Task scheduling for grid computing systems using a genetic Algorithm," *Journal of Supercomputing*, 71(4), pp. 1357-1377, April, 2015. Article (CrossRef Link)

[21] Dasgupta and Kousik, "A genetic algorithm (GA) based load balancing strategy for cloud computing," *Procedia Technol*, December, 2013. Article (CrossRef Link)

[22] M Cuppini, "A genetic algorithm for channel assignment problems," *Eur. Trans. Telecommun*, 5(2), pp. 285–294, March, 2010. Article (CrossRef Link)

[23] Manasrah Ahmad M and Ali Hanan Ba, "Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing," *WIRELESS COMMUNICATIONS & MOBILE COMPUTING*, 3, pp. 1-16, January, 2018. Article (CrossRef Link)

[24] Lin Yang-Kuei and Chong Chin Soon, "Fast GA-based project scheduling for computing resources allocation in a cloudmanufacturing system," *JOURNAL OF INTELLIGENT MANUFACTURING*, 28(5), pp. 1189-1201, June, 2017. Article (CrossRef Link)

[25] Guan T.T. et al, "Application research of multi objective partice swarm optimization in logistics distribution," *Nanchang University, Nanchang*, 2012. Article (CrossRef Link)

[26] Gan Na, Huang Yufeng and Lu Xiaomei, "Niching Particle Swarm Optimization Algorithm for Solving Task Scheduling in CloudComputing," *AGRO FOOD INDUSTRY HI-TECH*, 28(3), pp. 876-879, May, 2017. https://www.researchgate.net/publication/319091663_Niching_particle_swarm_optimization_algorithm_for_solving_task_scheduling_in_cloud_computing

[27] Casas I, Taheri J, Ranjan R and Zomaya AY, "PSO-DS: a scheduling engine for scientific workflow managers," *JOURNAL OF SUPERCOMPUTING*, 73(9), pp. 3924-3947, September, 2017. Article (CrossRef Link)

[28] N Sadhasivam, R Balamurugan and M Pandi, "Cancer Diagnosis Epigenomics Scientific Workflow Scheduling in the CloudComputing Environment Using an Improved PSO Algorithm," *Asian Pacific journal of cancer prevention : APJCP*, 19(1), pp. 243-246, January, 2018. Article (CrossRef Link)

[29] Awad A.I. et al, “Enhanced particle swarm optimization for task scheduling in cloud computing environments,” *Procedia Comput. Sci*, 65, pp. 920–929, December, 2015. Article (CrossRef Link)

[30] Q Cai, D Shan, W Zhao, “Resource scheduling in cloud computer based on improved particle swarm optimization algorithm,” *J. Liaoning Tech. Univ. (Natural Science)*, January, 2016. Article (CrossRef Link)

[31] MY Cheng and D Prayogo, “Symbiotic organisms search: a new metaheuristic optimization algorithm,” *Comput Struct*, 139, pp. 98–112, July, 2014. Article (CrossRef Link)

[32] Abdullahi Mohammed, Ngadi Md Asri and Abdulhamid Shafi’i Muhammad, “Symbiotic Organism Search optimization based task scheduling in cloud computing environment,” *Future Generation Computer Systems*, 56, pp. 640–650, August, 2015. Article (CrossRef Link)

[33] Vincent F.Y , Redi A.P. , Yang C.L , Ruskartina E and Santosa B, “Symbiotic organisms search and two solution representations for solving the capacitated vehicle routing problem,” *Applied Soft Computing, 52, pp.* 657–672, October, 2016. Article (CrossRef Link)

[34] Tejani GG et al, “Adaptive symbiotic organisms search (SOS) algorithm for structural design optimization,” *J.Comput Design Eng*, 3(3), pp. 226–249, February, 2016. Article (CrossRef Link)

[35] Hwang Chii-Ruey, “Simulated annealing: theory and applications,” *Acta Applicandae Mathematicae*, 37(1), pp. 108–111, 1987. Article (CrossRef Link)

[36] Strobl Maximilian AR and Barker Daniel, “On Simulated Annealing Phase Transitionsin Phylogeny Reconstruction,” *Molecular Phylogenetics and Evolution*, 101, pp. 46–55,May, 2016. Article (CrossRef Link)

[37] Absalom El-Shamir Ezugwu, Aderemi Adewumi and Marc Frincu, “Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem,” *Expert Systems With Applications*, 77, pp. 189–210, February, 2017. Article (CrossRef Link)

[38] Abdullahi Mohammed and Ngadi Md Asri, “Hybrid Symbiotic Organisms Search Optimization Algorithm for Scheduling of Tasks on Cloud Computing Environment,” *PLoS One*, 11(6), e0158229, Jun, 2016. Article (CrossRef Link)

[39] M Abdullahi, MA Ngadi and SI Dishing, “Chaotic Symbiotic Organisms Search for Task Scheduling Optimization on Cloud Computing Environment,” in *Proc. of Ict International Student Project Conference on. IEEE*, pp. 1-4, May, 2017. Article (CrossRef Link)

[40] Subhodip Saha and V. Mukherjee, “A novel chaos-integrated symbiotic organisms search algorithm for global optimization,” *Soft Computing*, 4, pp. 1-20, April, 2017. Article (CrossRef Link)

[41] Yang D, Li G and Cheng G, “On the efficiency of chaos optimization algorithms for global optimization,” *Chaos Solitons Fract*, 34(4), pp. 1366–1375, November, 2007. Article (CrossRef Link)

[42] Liu B, Wang L, Jin YH and Huang D, "Improved particle swarm optimization combined with chaos," *Chaos Solitons Fract*, 25(5), pp. 1261–1271, September, 2005.
Article (CrossRef Link)

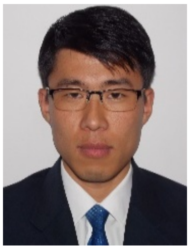
[43] Xiang T, Liao X and Wong K, "An improved particle swarm optimization algorithm combined with piecewise linear chaotic map," *Appl Math Comput*, 190(2), pp. 1637–1645, July, 2007.
Article (CrossRef Link)

SongIl Choe was born in Huichon, Democratic People's Republic of Korea, on August 10, 1986. He received his BSc in Information Science, from Huichon Industry University in 2007 and his MSc from the College of Information Science, Kim Il Sung University, PyongYang in 2011. Currently, he is a teacher at Huichon Industry University and is a PhD candidate at Huichon Industry University. His research interests include Cloud Computing, Artificial Intelligence, Speech Processing, Pattern Recognition, and Multimedia Communications.

Bo Li is a professor with the College of Management and Economics at Tianjin University. She received her bachelor and master degrees in Mathematics and Computer and System Control from Nankai University, China, in 1989 and 1992, respectively. She received her doctorate in Management Science and Engineering from Tianjin University, China, in 2000. Her research interests are Supply Chain Management and Coordination, and Logistics Optimization and Scheduling.

IlNam Ri received his B.S. and M.S. degree in Information Science from College of Information Science, Kim Il Sung University, Pyongyang, Democratic People's Republic of Korea in 1997 and in 2011, respectively. He is currently a professor at the College of Information Science, Kim Il Sung University. His research interests include Computer Network Architecture, Cloud Computing, DBMS, and Web Technology.

ChangSu Paek is a professor of Department of Wireless Communication in Huichon Industry University. He received his Bachelor's and Master's degree in Wireless Engineering from Huichon Industry University Democratic People's Republic of Korea in 1994 and 2000, respectively. He received his Doctor's degree of Wireless Engineering from Huichon Industry University, Democratic People's Republic of Korea in 2013. His research interests are Wireless Communication and Network, Digital Signal Processing, and OFDM.



Jusong Rim received his B.S degree from the Department of Control Scienc e, University of Sciences, Pyongyang, Democratic People's Republic of Korea in 2014. He is currently pursuing his M.S degree with the School of Electri cal and Information Engineering, Tianjin University, Tianjin, China. His curren t research interest includes Artificial Intelligence, and Smart Grid.

Subom Yun received his B.S and M.S degree in the Department of Mechanical Engineering, Huichon Industrial University, Huichon, Democratic People's Republic of Korea in 2007 and 2011, respectively. He is now an associate professor with the Department of Mechanical Engineering, Huichon Industrial University, Huichon, Democratic People's Republic of Korea. Area of his current interest includes Automation Engineering, Mechanical Engineering, and Intelligent Control Engineering.