Cilt 11, Sayı 3, Sayfa 35-43, 2022 Volume 11, Issue 3, Page 35-43, 2022

Araştırma Makalesi https://doi.org/10.46810/tdfd.1123962 Research Article

Türk Doğa ve Fen Dergisi
Turkish Journal of Nature and Science
www.dergipark.gov.tr/tdfd

Cloneable Jellyfish Search Optimizer Based Task Scheduling in Cloud Environments

Mücahit BÜRKÜK[1], Güngör YILDIRIM[1*]

[1]Firat University, Engineering Faculty, Department of Computer Engineering, Elazig, Türkiye
Mucahit BÜRKÜK ORCID No: 0000-0002-4974-0590
Güngör YILDIRIM ORCID No: 0000-0002-4096-4838

**Corresponding author: gungor.yildirim@firat.edu.tr*

(Received: 31.05.2022, Accepted: 27.07.2022, Online Publication: 29.09.2022)

Keywords
Task scheduling, Cloud computing, Metaheuristic, Jellyfish algorithm

Abstract: For cloud environments, task scheduling focusing on the optimal completion time (makespan) is vital. Metaheuristic approaches can be used to produce efficient solutions that will provide important cost savings to both the cloud service provider and the clients. On the other hand, since there is a high probability of getting stuck in local minima in metaheuristic solutions due to the type of problem, it may not always be possible to quickly reach the optimal solution. This study, using a metaheuristic approach, proposes a solution based on the Cloneable Jellyfish Algorithm for optimal task distribution in cloud environments. The unique feature of the proposed algorithm is that it allows dynamic population growth to be carried out in a controlled manner in order not to get stuck in local minima during the exploration phase. In addition, this algorithm, which uses a different cloning mechanism so that similar candidates are not generated in the population growth, has made it possible to achieve the optimal solution in a shorter time. To observe the solution performance, cloud environment simulations created in the Cloudsim simulator have been used. In experiments, the success of the proposed solution compared to classical scheduling algorithms has been proven.

Bulut Sistemlerde Denizanası Arama Optimizasyonu Tabanlı Görev Çizelgeleme

Anahtar Kelimeler
Görev Çizelgeleme, Bulut Hesaplama, Metasezgisel, Denizanası Algoritması

Öz: Bulut ortamlar için optimum tamamlanma süresi (makespan) çözümüne odaklanan görev planlamaları hayati öneme sahiptir. Hem bulut servis sağlayıcı hem de müşteriye ciddi maliyet kazancı sağlayacak çözümlerin üretilmesinde meta-sezgisel yaklaşımlar kullanılabilir. Öte yandan problem tipinden dolayı meta-sezgisel çözümlerde lokal minimumlara takılma olasılığı yüksek olduğundan optimum çözüme hızlıca ulaşmak her zaman mümkün olamayabilir. Meta-sezgisel bir yaklaşım kullanan bu çalışma, bulut ortamlarda optimum görev dağılımı için Klonlanabilir Deniz Anası Algoritması temelli bir çözüm önermektedir. Önerilen algoritmanın özgün özelliği, exploration aşamasında lokal minimumlara takılmamak için dinamik popülasyon artışının kontrollü bir şekilde yapılmasına olanak sağlamasıdır. Ayrıca popülasyon artışında benzer adayların üretilmemesi için farklı bir klonlama mekanizması kullanan bu algoritma, optimum çözüme daha kısa sürede ulaşmayı mümkün kılmıştır. Çözüm performansını gözlemlemek için Cloudsim simülatöründe oluşturulan bulut ortam simülasyonları kullanılmıştır. Farklı senaryolar için yapılan deneylerde, önerilen çözümün klasik scheduling algoritmalarına göre başarısı ispatlanmıştır.

1. INTRODUCTION

Cloud computing technology is the sum of virtualized and scalable resources that allow hosting a large amount of data on the Internet and provide users with a pay-per-use model [1]. Many reasons, such as the development and acceleration of the Internet infrastructure, the spread of IoT (Internet of Things) technology, the rapid growth of big data, and advances in artificial intelligence studies, have led to the widespread use of cloud technology. Cloud computing allows the users to access various services and resources (CPU, RAM, storage) anytime and anywhere. A cloud system can provide three types of services related to infrastructure, platform, and software. The first service is IaaS (Infrastructure as a Service), which provides infrastructure services such as a storage system and computational resources. The second service is PaaS (Platform as a Service), which allows clients to create their applications on the provided

platform. The third service, on the other hand, is SaaS (Software as a Service), which allows users to use software directly from the cloud instead of on local machines [2]. Cloud service providers must offer resources and services to their clients in a way that does not violate the SLA (Service Level Agreement) and guarantees a certain QoS (Quality of Service). Optimal use of cloud system resources and maintaining performance at the highest level are vital for both service providers and users. Task scheduling is a factor that directly affects cloud system performance and optimal resource utilization. A task or resource scheduling that has not been designed well can lead to an SLA violation, serious loss of revenue, and performance degradation.

One of the most important mechanisms of cloud computing is virtual machines (VMs). VMs are created from resources on the cloud system in accordance with the needs of clients. Depending on the volume of work, the number of clients' VMs and their features may vary. Clients pay a certain fee to the cloud provider based on the characteristics of these VMs and the duration of their use. Incorrect scheduling of tasks that need to be run on VMs leads to an increase in task completion time (makespan) and, naturally, to an increase in costs for the client. This increase in makespan also indirectly negatively affects the energy consumption and maintenance and repair costs of the cloud provider. Therefore, the use of a good task scheduling algorithm on cloud systems is mandatory for both the customer and the cloud provider [4]. Task scheduling is an NP-hard problem [1]. The use of metaheuristic algorithms instead of deterministic solutions is often preferred in solving such problems in terms of performance [5]. But on the other hand, due to the type of problem, the probability of the fact that random search-based metaheuristic algorithms are stuck in local minima is also high. This probability can further increase as the number of tasks and VMs increases. Therefore, it is necessary that the used metaheuristic algorithms use mechanisms that will overcome this problem. For this problem, this study proposes a solution based on Jellyfish Search Optimizer (JSO) [8], which is an up-to-date metaheuristic algorithm that uses a different approach mechanism. The most unique aspect of the proposed method is that it gets rid of local minima more quickly and allows dynamic population growth with a different similarity control. In this way, a more efficient exploration process is realized in the search space. The performance of the proposed method was proven by trying comparatively for different scenarios in the CloudSim simulator.

Other parts of this article are as follows; In Part II, task scheduling strategies in cloud computing systems and similar studies found in the literature are given. Part III contains the details of the used methodology and JSO algorithm. The experiments and comparative evaluations are given in Part IV, and the conclusion is presented in Part V.

2. BACKGROUND AND RELATED WORKS

In cloud systems, virtual machines (VMs) with different properties can be created on physical servers using virtualization techniques. VMs can collaborate to perform a specific task, as well as work independently of each other. Cloud service providers (CSPs) have broker services that conveniently distribute incoming tasks to VMs. Such services can also be developed by the client and run on a separate VM in the cloud. The Task Scheduling algorithm used in both cases is the most important factor that determines performance. Task scheduling has three main mechanisms: resource finding, resource determination, and task allocation [7]. The availability of resources requires that shareable resources can be questionable. Resource determination makes the selection of the most optimal resources depending on the characteristics of the tasks at the end of the resource query. Task allocation, on the other hand, sends the relevant tasks to the determined resources and performs their follow-up. The cloud system has a heterogeneous structure in terms of both resource and customer diversity. This naturally leads to the emergence of different goals in Task Scheduling optimization. Among these goals, the completion time (makespan), energy consumption, and cost stand out. Given this diversity, the strategy to be used in task scheduling should be chosen correctly. In the task scheduling proposed in the literature, strategies can be classified as shown in Figure.1 [11].

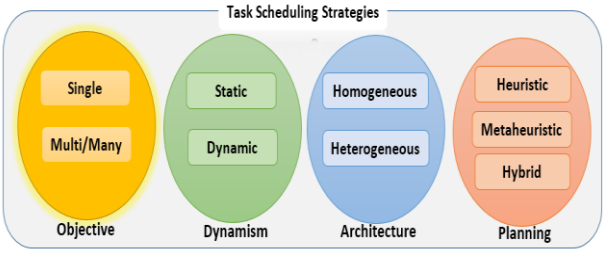


Figure.1 Task scheduling strategies

In the task scheduling strategy by the goal, one of the goals such as makespan, cost, energy consumption, or more than one that contradicts each other can be taken into account. The cloud infrastructure and the incoming task type also determine whether the strategy to be used will be static or dynamic. Static scheduling can be performed in fixed source cloud systems where there is not much workload change. However, since today's cloud technologies have a variable structure, dynamic strategies are usually preferred. This also applies to the heterogeneous structure of the CSP architecture. In terms of planning, heuristic techniques are often used for static scheduling. Metaheuristic algorithms that use heuristics and random search mechanisms together can achieve to create effective solutions for dynamic systems [6, 10, 13]. In this study, the proposed metaheuristic solution was developed for a single objective purpose in a heterogeneous and dynamic CSP system.

Because metaheuristic algorithms contain randomness, initial values are important. In [3], the authors presented a Discrete Symbiotic Organism Search (DSOS)

algorithm, which is a metaheuristic algorithm for the optimal scheduling of tasks in cloud resources. In [9], for the particle swarm optimization (PSO), which is a well-known metaheuristic algorithm, the initial values were found using the heuristic LJFP and MCT algorithms. Thus, success was achieved in makespan and total energy consumption. In [6], makespan optimization conducted with the grey wolf optimizer (GWO) that used the hunting mechanism, CPU, memory, and resource bandwidth parameters were taken into account together. In another study conducted for the purpose of makespan [12], the authors used the Electromagnetism Metaheuristic Algorithm (EMA) and monitored their VM performance comparatively. In [18], the authors used a multi-purpose Task scheduling strategy. For this, they proposed the ICW method that used the metaheuristic whale optimization algorithm (WOA). In [13], transaction cost and makespan optimization were performed with the developed space-shared genetic algorithm, and the superiority of the proposed solution over competitive planning algorithms was proved. Solutions based on ant colony optimization (ACO), which is another well-known metaheuristic algorithm, have also been introduced in the literature. An ACO-based solution taking into account the load balance and the purposes of the makespan was presented in [14]. In [15], a hybrid task scheduling method using the ACO and PSO algorithms was proposed. Another hybrid solution based on ACO was also introduced in [16]. In this study, the authors first ran the GA algorithm to determine the initial values of ACO; thus, they achieved a better execution time. A hybrid solution that performs more metaheuristic algorithm execution in it was proposed by [19]. In this study, GA, ACO, and PSO algorithms were run in the developed framework to obtain the optimum makespan value. In [21], the authors proposed a hyper-heuristic scheduling algorithm by integrating GA, ACO, and PSO into a single framework to reduce the makespan in the cloud. In [22], a minimum makespan task scheduling framework called MMSF and a minimum makespan task scheduling algorithm called MMA were proposed.

In general, in the Metaheuristic Task Scheduling solutions proposed in the literature, it has not been focused on overcoming the emerging local minima problems in a shorter period of time. Unlike the examples in the literature, this study aimed to reach the existing solutions faster with the state-controlled dynamic population variability.

3. MATERIAL AND METHOD

In Task Scheduling, the task scheduler assigns the tasks waiting in the queue to the appropriate VMs according to the output of the JSO algorithm. The used algorithm makes these assignments based on the calculations it makes for certain objective or objectives. In this study, Task Scheduling was performed according to the makespan objective. Makespan is the completion time of a certain number of tasks on the VMs to which they are assigned. The goal of Makespan optimization is to reduce this time to a minimum. As an example, let's assume that 7 different tasks are assigned to 3 different VMs as in Figure.2. In this case, the makespan value will be equal to the task completion time of the second VM.

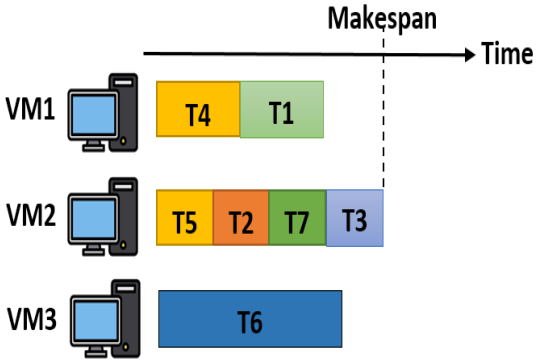


Figure.2 Makespan definition

The JSO tries to reduce makespan to a minimum by trying different assignment variations. In the creation of these variations, calculations made based on the characteristics of tasks and VMs are taken into account. In these calculations, the number of commands of tasks is expressed as a Million Instructions (MI), and the calculation capabilities of VMs are expressed as the number of Million Instructions Per Second (MIPS). Let $T_m = \{MI_0, MI_1, \dots, MI_K\}$ be considered as the tasks assigned to the m[th] virtual machine. In this case, the execution time of the k[th] task on the m[th] virtual machine can be calculated using Eq.1. The total execution time for all tasks is found using Eq.2.

$$ET_{km} = \frac{MI_k}{MIPS_m} \quad (1)$$

$$TET_m = \sum_{k=0}^{K} ET_{km} \quad (2)$$

Considering all VMs, the maximum total execution time determined based on the calculation made with Eq.3 will also give the makespan value. The fitness function of the proposed method is to obtain the minimum makespan value, as expressed in Eq.3.

$$Makepsan = max\{TET_m\}, m \in \{VM_1, VM_2, \dots\} \quad (3)$$
$$F^{obj} = \min\{Makespan\} \quad (4)$$

3.1. Jellyfish Search Optimizer (JSO)

In the proposed method, JSO, the current metaheuristic algorithm of recent years, was used to calculate the optimal Makespan value. JSO is a meta-heuristic optimization algorithm inspired by the movements of jellyfish in the ocean while satisfying their basic needs, such as finding food, and how they affect other individuals in the swarm [8]. In any case, a jellyfish wants to move to a place where the amount of food is more ($the\ best\ F^{obj}$). The amount of food in places visited by jellyfish can vary. In this case, the most favorable location in terms of food is found by comparing the amounts of food. Jellyfish perform two types of movements. These are the movement with the ocean current and the movement within the herd. The transition between these movements is controlled by a

time control mechanism. Since the ocean current contains a lot of nutrients, it always attracts jellyfish. In JSO, the ocean current ($\overrightarrow{OC}$) is found based on the average locations of all candidate solutions (Jellyfish) and the best fitness value. Eq.5 shows the mathematical model of the movement with the ocean current. Here, n_{pop} is the total population and X^* is the jellyfish with the best fitness value so far. In this equation, the random number (r_1) and the hyper-parameter (β) also represent the e_c attractiveness factor of the current ($e_c = \beta \times r_1, \quad r \in [0,1]$).

$$\overrightarrow{OC} = \frac{1}{n_{pop}} \sum \overrightarrow{OC_i} = \frac{1}{n_{pop}} \sum (X^* - e_c X_i) = X^* - \beta \times r_1 \times \frac{\sum X_i}{n_{pop}} \quad (5)$$

In the case of movement relative to the ocean current, the next locations of jellyfish ($X_i(t+1)$) are found by using Eq.6. where $X_i(t)$ represents the current state of the i[th] jellyfish and r_2 represents the number of uniform random.

$$X_i(t+1) = X_i(t) + r_2 \times \overrightarrow{OC} \quad (6)$$

Movements of jellyfish in the swarm are of two types: passive (Type A) and active (Type B) [17]. Initially, most jellyfish exhibit Type A movement while forming a swarm. Over time, Type B movement is exhibited more. In Type A, the jellyfish performs a random movement around its location. The model for this type of movement is given in Eq.7. where γ represents the movement coefficient, and U_b and L_b represent the upper-lower bounds of the search space, respectively.

$$X_i(t+1) = X_i(t) + \gamma \times rand(0,1) \times (U_b - L_b) \quad (7)$$

Type B movement occurs according to the state of the food resources of the i[th] jellyfish as well as j[th] jellyfish which is randomly selected in the swarm. The movement will be towards the jellyfish, where there is more food. The jellyfish performs a random movement in the designated direction. In Type B, the direction of movement and the new location of the jellyfish are calculated by Eq.8 and 9. where $\vec{D}$ is the direction of movement and r_3 is the uniform random value.

$$\vec{D} = \begin{cases} X_j(t) - X_i(t) & if \quad f(X_i) \geq f(X_j) \\ X_i(t) - X_j(t) & if \quad f(X_i < f(X_j) \end{cases} \quad (8)$$

$$X_i(t+1) = X_i(t) + r_3 \vec{D} \quad (9)$$

The movements of jellyfish in the swarm initially begin with type A, and over time they switch to type B. In addition, the movement of the ocean current is also taking place over time. In JSO, a time control mechanism is used for all movements of jellyfish. This control mechanism is modelled by Eq.10. In this model, t is the number of iterations, r_4 is the uniform random coefficient, and $max_{iteration}$ is the maximum number of iterations. The result of the control mechanism is compared with a threshold value (usually 0.5). If the value of the control function is below the threshold value, movement occurs within the swarm, and if it is above the threshold value, movement occurs based on the ocean current. For the movement inside the swarm, the value of $1 - c(t)$ is taken into account. For this purpose, a random value is generated with a uniform generator and this value is compared with $1 - c(t)$. If the generated random value is higher, the type A movement occurs, if not, the type B movement occurs.

$$c(t) = \left| \left(1 - \frac{t}{max_{iteration}}\right) \times (2 \times r_4 - 1) \right| \quad (10)$$

In the fitness tests conducted to determine the coefficients used in the JSO, the effect of the ocean current and the effects of movement types on the results were also observed. As a result of these tests, it was found that the most optimal solutions were obtained for β = 3 and γ = 0.1 [8].

3.2. Cloneable JSO and The Proposed Task Scheduling

Similar to other metaheuristic algorithms, the JSO algorithm also has the risk of getting stuck to the local minima in Task Scheduling problems. To overcome this risk, techniques such as increasing iteration or population number or using different random generator functions can be used. High population and iteration numbers are accompanied by time costs. The technique proposed in this study suggests that the optimum value can be reached in a shorter time by increasing the population growth heuristically and dynamically. Accordingly, if there is no change in the best value during a given number of iteration, the current population is increased at a certain rate at run-time. In fact, it was inspired by the biological characteristics of jellyfish for this feature. In nature, jellyfish are creatures that have the ability to clone themselves in a controlled way. However, the critical point here is the positional values of new population candidates (clones). Adding candidates similar to existing candidates to the population will reduce the likelihood of getting out of the local minimum. For a more effective exploration process, the fact that new individuals differ from existing ones is one of the main points of the proposed Cloneable JSO (C-JSO) algorithm [23]. The high computational cost of the function to be used for similarity checks of new candidates will also increase the time cost of the algorithm, especially in high populations. Therefore, it is necessary to use a fast and effective similarity check function. C-JSO uses a fast and effective similarity function to prevent the generation of similar candidates. Task Scheduling, which is used in doing this, benefits from the discrete nature of the problem solving. In the Task Scheduling algorithm, tasks and VMs are usually encoded with integers. Figure.3 shows an example encoding for two candidate solutions (X_i and X_j). Candidate attributes indicate a task. Each attribute value, on the other hand, is an integer code showing the current VMs. The similarity criterion in Task Scheduling is the number of times the same Tasks are assigned to the same VMs in the current and new candidate. C-JSO calculates

the similarity ratio by performing a match comparison for each task. In this calculation, the similarity is calculated by the ratio of the total number of zeros, obtained as a result of taking the differences in attributes of the two candidates, to the total number of Tasks. This calculation is expressed in Eq.11 and 12. Accordingly, the similarity ratio of the two candidates in Figure.3 is 50%.

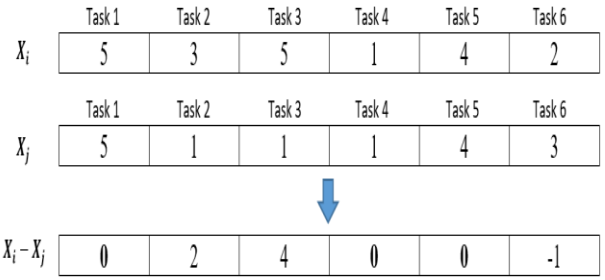


Figure.3 A sample similarity control for task assignment [23]

$$Df = X_i - X_j \quad (11)$$

$$Similarity = \frac{the\ number\ of\ zeroes\ in\ Df}{the\ number\ of\ tasks} \quad (12)$$

In C-JSO, population candidates (jellyfish) have a VM assignment vector as large as the total number of Tasks. Initially, random integer assignments representing VMs are made to these vectors for each candidate. Then the fitness value of each candidate is calculated by Eq.1-4. The candidate with the best fitness value will also represent the ocean current. Throughout the iterations, depending on the value of the time control function, each jellyfish determines its new location within the swarm or relative to the ocean current. In C-JSO, there is a binary variable (φ) that tracks the best value change. If there is no change in the best value in a predefined number (I) at the end of the iteration, this variable gets a value of true. In this case, the current population is increased by a rate of a predefined value (θ). When increasing the population, the similarity of new candidates is calculated by Eq.11-12. As a result of the calculation, candidates below the predefined similarity threshold (ε) are added to the population and the basic steps are repeated. C-JSO's pseudo-code showing these steps is given in Algorithm-1.

Algorithm 1. The pseudo code of C-JSO based task scheduling [23]

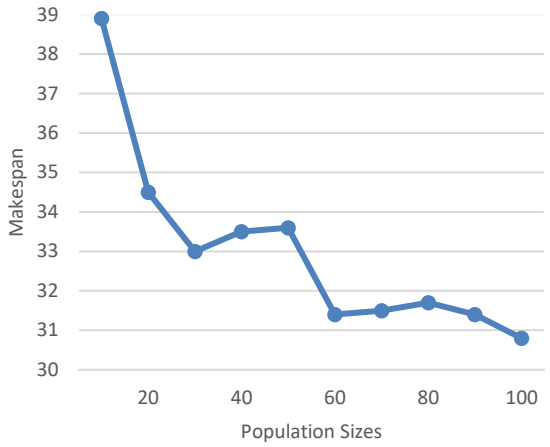
Initialize the population
Specify $\beta, \gamma, \theta, \varepsilon, I$ and $\varphi \rightarrow 0$
While (itr < max iteration)
  Find the fitness values of all candidates by Eq.1-4
  Run time control function by Eq.10
  If c(t) < 0.5 **then** follow $\overrightarrow{OC}$
  Else if 1- c(t) < rand (0,1) **then** make A-type move
  Else make B-type movement
  If there is no change in I iterations ($\varphi \rightarrow 1$) **then** increase the population by θ% considering Equation 11-12
  Else $\varphi \rightarrow 0$
End while
Return the best solution

4. RESULTS AND DISCUSSION

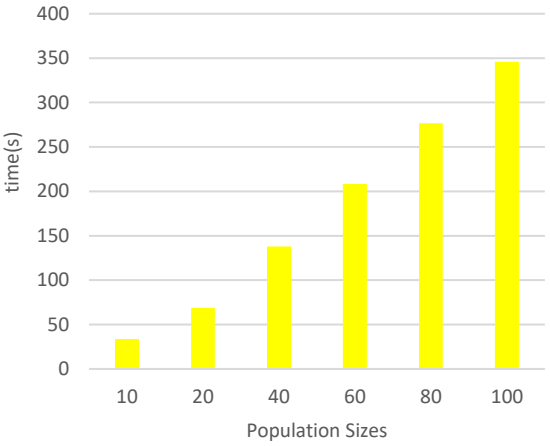
The success of the C-JSO-based Task Scheduling method was tested in the CloudSim simulator [20] for different Cloud scenarios. The success of the C-JSO was shown in comparison with the results of the default CloudSim Task Scheduling algorithm, the classic JSO, and the ACO algorithm. For simulations, a data center was created primarily in CloudSim. This data center has two main physical servers, each has 16 GB of ram, 10 TB of Storage, 1 GB/s of bandwidth, and time-shared VM scheduling. All VMs are distributed equally on these physical servers. The first of these computers has 4-core and the second has dual-core X86-architecture CPUs. The processing capacity of each processor core is 10000 MIPS. There are Linux operating system and Xen VMM on computers. The VMs have 512 MB of ram, 10 GB of Storage, 10 MB/s of bandwidth, and time-shared task scheduling configuration. The processing capacity of VMs ranges from 1000 to 5000 MIPS, and the command length of tasks ranges from 5000 to 20000 MI. The standard task planning method in CloudSim is "*CloudletSchedulerSpaceShared*". The other parameters used in the experiments are given in Table 1. Statistical results were obtained by running each of the experiments, conducted with 100, 250, 500, and 750 task numbers, 10 times in order to observe the performance of different scenarios.

Table 1. Experiment parameters

PARAMETERS	VALUES
Population sizes	10, 20, 30, 50, 60, 80, 100
Initial population size for C-JSO	10
Maximum Iteration	500
Task Sizes	100 - 750
VM number	20
Task MIs	5000 - 20000
VM MIPSs	1000-5000
Increasing Rate for C-JSO (θ)	13 %
Similarity Rate (ε)	90 %



a) Traditional JSO makespan results

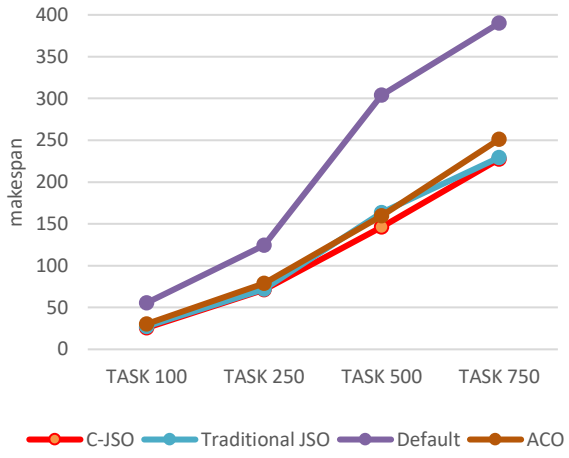


b) Traditional JSO time results

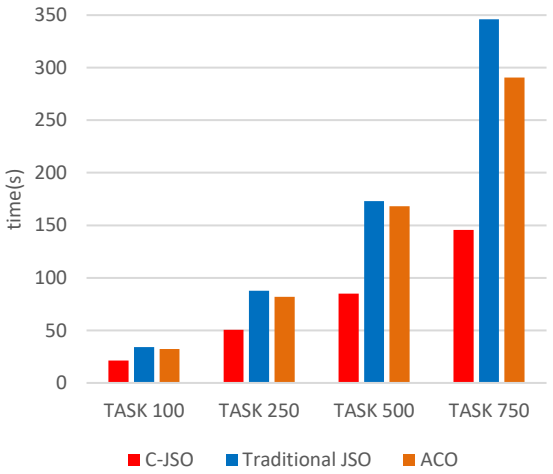
Figure4. The makespan and time results of traditional JSO with different population sizes for 100-task scheduling

In the experiments, first, how population growth affects makespan was examined. For this purpose, classical JSO-based solutions with different population numbers and their durations have been analyzed in a fixed-size Task Scheduling problem. Figure.4a-b shows the results of these experiments. In these experiments, classical JSO algorithms with different population sizes were run for 100 Tasks. As can be seen in Figure.4a, population growth leads to an improvement in the value of makespan. On the other hand, Fig.4b shows that this improvement has a negative effect on the solution time. The proposed method uses dynamic population growth to improve this disadvantage. Thus, at certain task sizes, the optimal result will be achieved in a shorter time with the appropriate number of populations.

Later experiments were conducted for CloudSim scenarios. In these experiments, classical JSO, C-JSO, default Cloud Scheduling, and ACO-based methods were run, and their results were examined. The comparative and statistical results of these experiments are given in Figure.5a-b and Table-2, respectively. In simulation experiments, the worst makespan values were obtained by the default scheduling algorithm. Metaheuristic approaches achieved results that were close to each other in makespan values and about twice as successful results compared to the default scheduling algorithm. Among themselves, on the other hand, JSO and C-JSO were relatively more successful compared to ACO-based Scheduling. When examined in terms of duration, C-JSO was much more successful than other algorithms. While JSO had the highest values in terms of calculation time, C-JSO achieved the shortest time.



a) C-JSO, Traditional JSO and Cloudsim makespan results



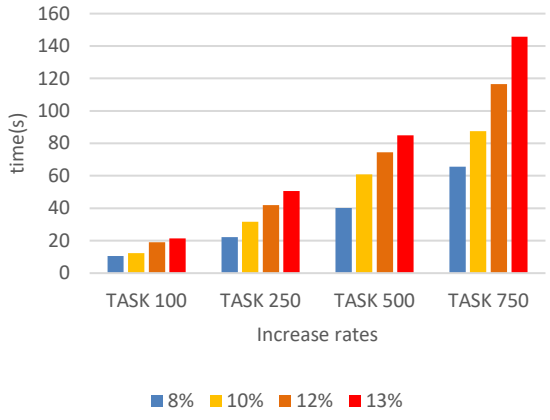
b) C-JSO and Traditional JSO time results

Figure.5. Makespan and time comparison for the methods used

According to statistical results, although the ACO algorithm approached the JSO and C-JSO values in terms of the minimum makespan values, the highest makespan values were also obtained by the ACO. Statistical results revealed that the performances of C-JSO and JSO are close to each other. The main superiority of C-JSO manifested itself in the calculation time.

Table 2. Statistical results for the methods used

	Method	100 Task	250 Task	500 Task	750 Task
Minimum	JSO	27.38	71.92	163.46	229.43
Maximum		31.05	77.92	201.73	257.34
Mean		29.33	73.81	179.72	244.08
Median		29.64	75.76	177.28	244.68
Std.		1.26	4.08	12.54	8.96
Minimum	C-JSO	25.61	71.06	146.63	227.45
Maximum		31.42	77.09	161.66	255.74
Mean		27.96	73.64	154.86	242.86
Median		27.88	74.07	154.72	242.46
Std.		1.48	1.79	4.03	6.94
Minimum	ACO	30.31	79.02	159.54	251.10
Maximum		35.16	87.33	193.47	287.11
Mean		34.53	83.18	171.11	267.01
Median		32.14	84.06	177.76	263.74
Std.		2.29	3.12	10.49	9.88
Minimum	CloudSim	55.51	124.43	304.13	390.08



a) The average time results of C-JSO for different increase rates

b) The average time results of C-JSO for different similarity rates

Figure.6. Effect of increase and similarity rates in C-JSO on makespan

Important hyper-parameters of C-JSO are the population growth rate and the similarity ratio used in the generation of new candidates. For this reason, parameter experiments were performed for different values, and the behaviour of C-JSO was examined. In the experiments, 8%, 10%, 12%, and 13% values were selected for the increase rate. In experiments over 13%, there was no improvement in makespan values, and the calculation time approached JSO. Similarity ratios of 60%, 70%, 80%, and 90% were selected. In the experiments, the best results were obtained at the 13% increase rate and 90% similarity rate. The average time performances of these parameters for different scenarios are given in Figure 6a-b and their effects on makespan are given in Figure 7 and 8.

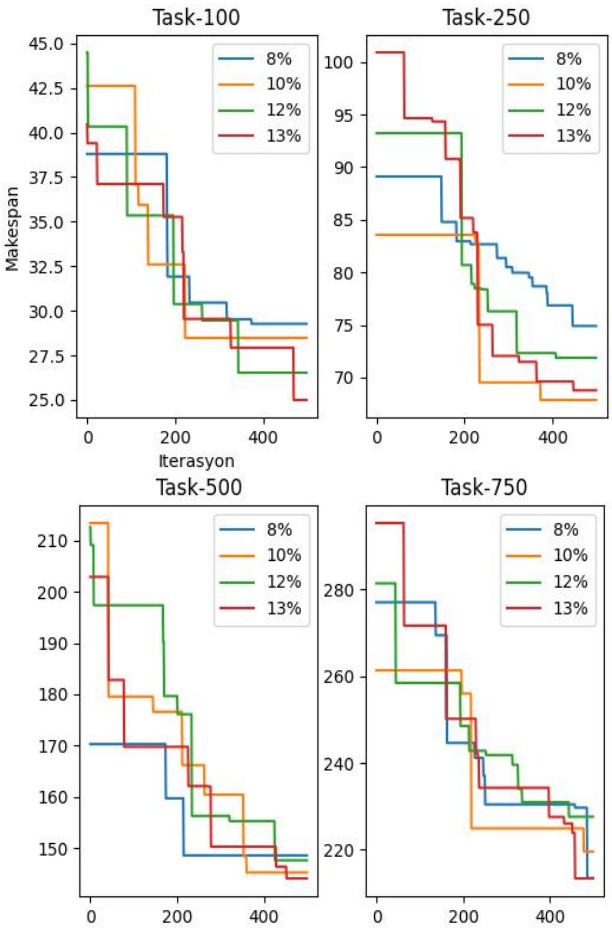


Figure.7. Effect of increase rate in C-JSO on makespan

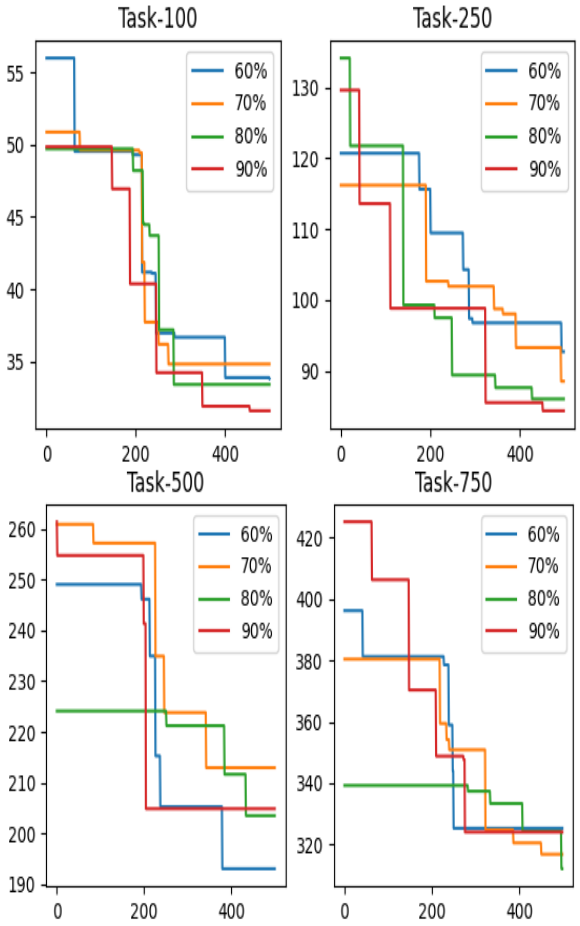


Figure.8. Effect of similarity rate in C-JSO on makespan

4. CONCLUSION

This study focused on the task scheduling process, which is one of the most important problems in cloud computing. To solve this problem, an adapted meta-heuristic algorithm, the C-JSO, which is based on the Jellyfish Search Algorithm (JSO), was developed. C-JSO has functions that can make some mechanisms of traditional JSO more flexible, such as the population structure. The results obtained from the experiments were compared with the CloudSim default task scheduler and the ACO algorithm results. Both classic JSO and C-JSO solutions managed to provide successful results in standard cloud task-sharing methods. It was observed that in Makespan and time comparisons, the C-JSO was more successful than the others. It is clear that cloud systems will remain a topic where different problems will arise for a long time. Therefore, the authors will focus on the solutions of different optimization problems emerging in cloud systems in their next studies.

Acknowledgement

This article is derived from Mucahit BURKUK's master's thesis titled Developing a Metaheuristic Solution Model to Task Scheduling Problems in Cloud Systems.

REFERENCES

[1] Strumberger I., Tuba E., Bacanin N., and Tuba, M. Dynamic tree growth algorithm for load scheduling in cloud environments. In 2019 IEEE Congress on Evolutionary Computation. 2019; p. 65-72.

[2] Avram MG. Advantages and challenges of adopting cloud computing from an enterprise perspective. Procedia Technology.2014; 12, 529-534.

[3] Abdullahi M., Ngadi MA. Symbiotic organism search optimization-based task scheduling in cloud computing environment. Future Generation Computer Systems. 2016; 56, 640-650.

[4] Houssein EH., Gad AG., Wazery YM., Suganthan PN. Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation. 2021; 62, 100841.

[5] Mohamed AB. , Laila AF, Arun KS. Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications; 2018.

[6] Yildirim G., Alatas B. New adaptive intelligent grey wolf optimizer based multi-objective quantitative classification rules mining approaches. Journal of Ambient Intelligence and Humanized Computing. 2021; 12, 9611–9635. https://doi.org/10.1007/s12652-020-02701-9

[7] Pradhan A., Bisoy SK., Das A. A survey on pso based meta-heuristic scheduling mechanism in cloud computing environment. Journal of King Saud University-Computer and Information Sciences; 2021.

[8] Chou JS., Truong TN. A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean, Applied Mathematics and Computation.2021; 389, 125535.

[9] Alsaidy SA., Abbood AD., Sahib MA. Heuristic initialization of PSO task scheduling algorithm in cloud computing. Journal of King Saud University-Computer and Information Sciences; 2020.

[10] Yıldırım S., Yıldırım G., Alatas B."Anlaşılabilir Sınıflandırma Kurallarının Ayçiçeği Optimizasyon Algoritması ile Otomatik Keşfi", Türk Doğa ve Fen Dergisi. 2021; vol. 10, no. 2, pp. 233-241, doi:10.46810/tdfd.976397

[11] Saurav SK., Benedict S. A Taxonomy and Survey on Energy-Aware Scientific Workflows Scheduling in Large-Scale Heterogeneous Architecture. In 2021 6th International Conference on Inventive Computation Technologies (ICICT). 2021; (pp. 820-826). IEEE.

[12] Belgacem A., Beghdad-Bey K., Nacer H. Task scheduling optimization in cloud based on electromagnetism metaheuristic algorithm. In 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS) 2018; (pp. 1-7). IEEE.

[13] Yildirim G., Hallac İR., Aydin G., Tatar Y. "Running genetic algorithms on Hadoop for solving high dimensional optimization problems," 2015 9th International Conference on Application of Information and Communication Technologies

(AICT). 2015; pp. 12-16, doi: 10.1109/ICAICT.2015.7338506

[14] Li K., Xu G., Zhao G., Dong Y., Wang D. Cloud task scheduling based on load balancing ant colony optimization. In 2011 sixth annual ChinaGrid conference. 2011; (pp. 3-9). IEEE.

[15] Liu CY., Zou CM., Wu P. A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing. In 2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science 2014; (pp. 68-72). IEEE.

[16] Chen X., Çeng U., Liu K., Liu Q., Liu J., Ying Mao, Murphy J. A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems. 2020; Volume: 14, Issue: 3, 3117 – 3128. IEEE.

[17] Zavodnik D. Spatial aggregations of the swarming jellyfish Pelagia noctiluca (Scyphozoa), Mar. Biol. 1987; 94, 265–269.

[18] Kıran, MS., Fındık O. A directed artificial bee colony algorithm, Appl. Soft Comput. 2015; 26, 454–462.

[19] Kıran M.S., Gündüz M., Baykan ÖK. A novel hybrid algorithm based on particle swarm and ant colony optimization for finding the global minimum, Appl. Math. Comput, 2012; C. 219, 1515–1521.

[20] Calheiros RN., Ranjan R., Beloglazov A., De Rose CA., Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 2011; 41(1), 23-50.

[21] Tsai CW., Huang WC., Chiang MH., Chiang MC., Yang CS. A hyper-heuristic scheduling algorithm for cloud. IEEE Transactions on Cloud Computing, 2014; 2, 236-250.

[22] Sasikaladevi N. Minimum makespan task scheduling algorithm in cloud computing, International Journal of Advances in Intelligent Informatics ISSN: 2442-6571, 2016; pp. 123-130.

[23] Burkuk M., Developing a Metaheuristic Solution Model to Task Scheduling Problems in Cloud Systems, Master Thesis, Graduate School of Natural and Applied Sciences, Firat University, 2022