HEACT: Hybrid Evolutionary Algorithm for the Multi-region Multi-objective Cloud Task Scheduling Problem. A Study of Workflow Scheduling in AWS EC2

Vinicius Renan de Carvalho[1][0000−0002−4623−7244] and Jaime Simão Sichman[1][0000−0001−8924−9643]

Escola Politécnica
Universidade de São Paulo (USP), Brazil
{vrcarvalho,jaime.sichman}@usp.br

Abstract. Cloud computing has revolutionized the provisioning and access of computing resources, offering scalable and flexible alternatives to traditional infrastructure. However, defining how to use these computational resources may be challenging. This paper addresses the challenge of workflow scheduling in cloud environments, focusing on Amazon Web Services (AWS) Elastic Compute Cloud (EC2). We present HEACT, a novel approach that integrates a multi-objective evolutionary algorithm with a specialist scheduling heuristic. The evolutionary algorithm is responsible for generating an initial set of machines (with their performance capability and cost information). The set is sent to the specialist scheduling heuristic for efficient task assignment in these machines. Our approach considers fourteen AWS regions, accurate pricing information from AWS, and employs SimGrid to simulate task execution. The proposed method was benchmarked considering established heuristics (HEFT, PEFT, HSIP, MPEFT) and meta-heuristics (NSGA-II, AGEMOEA2). Results demonstrated that the combinations of AGEMOEA2 with MPEFT and AGEMOEA2 with HEFT yield the best performance, indicating AGEMOEA2's efficacy as a state-of-the-art meta-heuristic for workflow scheduling. [1]

Keywords: Cloud Scheduling · Multi-objective optimization · Evolutionary Algorithms · Scheduling · Heuristics

1 INTRODUCTION

Cloud computing changed how computational resources are provisioned and accessed, offering a scalable and flexible alternative to traditional on-premises infrastructure. Moreover, elasticity allows users to dynamically adjust their re-

[1] This is a post-peer-review, pre-copyedit version of an article published in Lecture Notes in Computer Science, volume 15413. The final authenticated version is available online at: http://dx.doi.org/10.1007/978-3-031-79032-4_4.

source usage based on demand, ensuring that users can scale their applications up or down, paying only for the resources they consume.

Amazon Web Services (AWS) is an Infrastructure as a Service (IaaS) that provides virtualized computing resources over the Internet. In this context, with AWS EC2 [2], users can provision virtual machines (VMs) on demand according to their requirements. AWS EC2 VMs come in various instance types, from smaller to high-performance computing instances. Their computational power correlates with their capabilities and costs.

Effective resource management and scheduling are important in maximizing the benefits of cloud computing, especially when processing workflows. Workflow scheduling in cloud computing, which is known as NP-complete, consists of orchestrating complex sequences of tasks to be processed. Workflows are commonly represented as directed acyclic graphs (DAGs), where nodes represent tasks and edges denote dependencies between tasks.

Several approaches have been proposed over the years in order to schedule tasks to optimize performance. The list-based scheduling algorithm is the most popular scheduling paradigm for the computation of complex workflow applications [1]. Among them, we can cite HEFT [26], PEFT [2], HSIP [27] and MPEFT [20] (all described in section 2) as heuristic specialized in assigning tasks to processors (or machines) focusing on reducing the total time for executing all tasks. This total time is named as *makespan*.

Over the years, genetic algorithms (GA) were also considered for scheduling tasks in this domain. In [15], a single objective genetic algorithm combined with HEFT. In this approach, the GA considers single-point crossover and mutation operators. The chromosome was defined as an ordered list of tasks. As commonly performed in heuristics for this problem, a fixed set of processors was considered.

Nasonov et al. [17] proposed Linewise Earliest Finish Time (LEFT) by combining the HEFT heuristic and the genetic algorithm to optimize the makespan for cloud workload optimization.

In [24], the authors introduced two hybrid heuristics based on genetic algorithms for task scheduling. Both algorithms enhance the initial population of the GA by incorporating guided chromosomes using novel prioritization equations and improved mutation strategies. Similar to [15], these algorithms consider as chromosomes an ordered list of tasks, use two-point crossover and proposed mutation, use makespan as a single-objective function, and consider a specialist heuristic, in that case, HLTSD [25].

These mentioned works are, in fact, effective. However, an overall literature review reveals an evident lack of workflow scheduling algorithms that optimize equally important makespan and monetary cost functions [13]. Despite of that, Durillo et al. [11] introduced MOHEFT, a multi-objective workflow scheduling algorithm based on HEFT. MOHEFT aims to optimize two key objectives simultaneously: minimizing makespan and enhancing energy efficiency.

[2] AWS Elastic Compute 2, the virtual machine product of AWS

However, there are significant challenges in directly applying these algorithms to Cloud environments because most of them are still based on traditional heterogeneous environments such as Grid [30].

In order to tackle this challenge, Zhu et al. [30] proposed using an evolutionary multi-objective optimization algorithm to address the Cloud workflow scheduling problem as a multi-objective problem. This approach generates a series of schedules with different tradeoffs between cost and time so that users can choose acceptable schedules that fit their preferences. The chromosome is defined using three arrays: one responsible for ordering the tasks, the other responsible for assigning the tasks to a machine, and the other for defining the type of the machine. They applied NSGA-II [9], SPEA2 [31], and MOEA/D [29]. The case study revolves around Amazon EC2, utilizing instance specifications and pricing schemes, focusing on the US East region, considering only a single region and a small number of machine types.

Chen and Menzies [7] proposed RIOT (Randomized Instance Order Types). This heuristic method considers aspects from HEFT to group tasks in the workflow, considering a probability model and a surrogate-based method. This paper used the exact chromosome representation considered by [30] and a similar setup.

Qin et al. [21] approach the same problem, focusing on three key objectives: minimizing makespan and cost while maximizing reliability. This problem is situated within the context of multi-cloud systems, where machines from both Azure and AWS were utilized. To address this task, they proposed a variation of the memetic algorithm [16] for multi-objective problems. The studied case study considered the same workflow scheduling problem but in the multi-cloud scenario.

Motivated by research that combines genetic algorithms with specialist heuristics and approaches that consider the problem as a multi-objective problem, this work proposes HEACT, **H**ybrid **E**volutionary **A**lgorithm for the Multi-region Multi-objective **C**loud **T**ask Scheduling Problem.

Contributions: This research has as contributions 1. Proposes a hybrid algorithm that considers a multi-objective evolutionary algorithm and a specialist heuristic in order to solve the multi-objective task scheduling problem considering makespan and cost as objectives; 2. Considers multi-region: instances located in different places in the world have different associated costs. Moreover, regions may have a different set of machine types; 3. Uses realistic AWS information to instantiate the simulator (Simgrid); 4. Considers state-of-the-art scheduling heuristic (specially HSIP and MPEFT) with state-of-art meta-heuristic (especially AGEMOEA2 [19]). As far as we know, MPEFT was never studied together with a GA, while AGEMOEA was never applied to the cloud scheduling problem.

This paper is organized as follows: First, in Section 2, we provide some background necessary for the understanding of our proposal. Then, in Section 3, we present our proposal. Then, in Section 4, we detail our experiments and provide results to demonstrate the effectiveness of our approach. Finally, in Section 5, we present our conclusions and further work.

2 BACKGROUND

In this section, we provide the background necessary for understanding our proposal. First, we define the cloud scheduling problem and then detail the specialized heuristics used to solve it. Finally, we introduce meta-heuristics briefly.

2.1 Scheduling Problem formulation

The cloud task scheduling problem can be modeled as a DAG $G = (V, E)$ composed by a set of nodes $t_i \in V$, which represents an application task to be executed, and a set of edges E, which represents dependency between tasks and have an associated cost. Thus, given $e(i, j) \in E$, a task t_i should be wholly executed before starting task t_j.

The scheduling problem consists of assigning tasks to processors considering dependency between n tasks presented in the *DAG* and information regarding the processors. In this paper, we commonly use the name processor and, in our context, consider a processor to be the same as a virtual machine. Equation 1 calculates the objective function of the scheduling problem.

$$makespan{=}\max\{AFT(t_{exit})\} \tag{1}$$

The Actual Finish Time (AFT) represents when all the tasks and dependencies were executed and considers the last task t_{exit}. In order to calculate an AFT, we have to calculate the *Earliest Starting Time* (EST) and the *Earliest Finish Time* (EFT) for all the tasks.

$$EFT(t_i,p_j){=}w_{t_i,p_j}{+}EST(t_i,p_j) \tag{2}$$

Thus, to calculate when a task will finish, we sum the starting time with the computational cost w_{t_i,p_j}. This is performed for a given task t_i considering a given processor p_j. The computation cost matrix W of size $nt \times np$, where nt is the number of tasks, and np is the number of processors. Each $w_{i,j} \in W$ provides the estimated execution time to complete task t_i on processor p_j.

The *Earliest Starting Time* (EST in the previous equation) is calculated according to Equation 3:

$$EST(t_i,p_j){=}\max\{T_{avaliable}(p_j),\max_{t_m \in pred(t_i)}\{EFT(t_m){+}c_{m,i}\}\} \tag{3}$$

In this equation, $T_{avaliable}(p_j)$ is the time at which processor p_j is available and $pred(t_i)$ represents all tasks that t_i is dependent. We first calculate the latest time we can get all dependencies executed plus communication time. Thus, we guarantee that all dependencies will be respected.

This scheduling problem considers that a given task t_m could depend on another task t_i, which is located in another processor. Hence, it considers how much time will be necessary for the data generated by t_i, which is an input in t_m, arrives at the processor that t_m will be processed. This is what we call as a *communication cost* and it is represented by $c_{m,i}$, which is the communication cost for tasks t_i and t_j and it is defined by:

$$c_{m,i}{=}L_{t_m}+\frac{data_{i,m}}{B_{t_m,t_i}} \tag{4}$$

In this equation, B represents the transfer rates between the processors. This matrix has a size of $np \times np$, where np is the number of processors. L represents the processors' communication startup costs. $data_{i,m}$ represents the amount of data to be transferred and processed. If t_i and t_m are at the same machine, then $c_{m,i} = 0$.

The mono-objective scheduling problem considers only the *makespan* as an objective to be minimized. This is a realistic scenario only when we have a static set of processors waiting to receive tasks to run, so in this case, we are just trying to use the resources we already have (on-premises servers) better. Recently, with the advent of cloud providers, we can use the machines we need. By doing this, costs are reduced when using other companies' computational resources.

In the context of task scheduling, we can also consider the cost to be optimized and then treat this problem as a multi-objective optimization problem (MOP). Equation 5 presents this function; there, we convert the makespan into a fraction of an hour and then multiply it by the cost of every machine (processor) we are considering. Each machine has a type (p_{type}), which defines how powerful the machine is and how much it will cost. The same happens for the region where the machine is located (p_{region}). Usually, the same machine type in different regions may have different prices.

$$TotalCost{=}\sum_{p\in P}\frac{Makespan}{3600}*Cost_{(p_{type},p_{region})} \tag{5}$$

2.2 Heuristics to deal with the problem

Several of the specialized heuristics in the scheduling problem consist of sorting the tasks according to a criterion and then iterating through this ranking in order to assign tasks to processors considering the shorter starting time available. For example, in the Heterogeneous Earliest Finish Time algorithm (HEFT) [26], tasks are ranked decreasingly according to the *upward rank value* (Equation 6), which is based on mean computation and mean communication costs.

$$rank_u(t_i){=}\bar{w_i}{+}\max_{t_j\in succ(t_i)}(\bar{c_{i,j}}{+}rank_u(t_j)) \tag{6}$$

In this equation, *succ* is the set of immediate successors of task t_i $\bar{c_{i,j}}$ is the average communication cost of an edge (i, j) and $\bar{w_i}$ is the average computation cost of task t_i among the processors.

Similarly, the Predict Earliest Finish Time algorithm (PEFT) [2] also ranks tasks but considers the Optimistic Cost Table (OCT) for the ranking phase. The OCT (Equation 7) is a matrix of size $np \times nt$ where each element represents, for a given task t_i and processor p_k, the maximum of the shortest paths of t_i dependencies tasks to the exit node. $\bar{c_{i,j}} = 0$ if $p_w = p_k$.

$$OCT(t_i,p_k)=\max_{t_j\in succ(t_i)}[\min_{pw\in P}\{OCT(t_j,p_w)+w(t_j,p_w)+c_{i,j}\}] \tag{7}$$

The OCT represents the maximum optimistic processing time of the children of task t_i because it considers that dependencies tasks are executed in the processor that minimizes processing time (communications and execution) independently of processor availability [2].

With the values of OCT calculated, we can then rank the values decreasingly, according to Equation 8.

$$rank_{oct}(t_i)=\frac{\sum_{k=1}^{qp} OCT(t_i,p_k)}{qp} \tag{8}$$

In order to select a processor for a task, the algorithm considers the lowest Optimistic EFT ($OEFT$ Equation 9), which sums to the Estimated Finish Time of a given task to the computation time of the longest path to the exit node (OCT). The aim is to guarantee that the tasks ahead will finish earlier, which is the purpose of the OCT table.

$$OEFT(t_i,p_j)=EFT(t_i,p_j)+OCT(t_i,p_j) \tag{9}$$

In [27], the authors proposed the heterogeneous scheduling algorithm with improved task priority (HSIP). This algorithm is considered a prioritization strategy based on computational cost standard deviation and total communication cost for a task (Equation 10). σ_i is the standard deviation of the computational cost of the given task t_i all over the processors while $\bar{w_i}$.

$$rank_u(t_i)=\max_{t_j\in succ(t_i)}\{\bar{w_i}\times\sigma_i+occw(t_i)+rank_u(t_j)\} \tag{10}$$

Moreover, the $occw$ term is defined by Equation 11, where $occw(t_{exit}) = 0$.

$$occw(t_i)=\sum_{t_j\in succ(t_i)} c_{i,j} \tag{11}$$

HSIP also implements an entry task duplication in order to improve the makespan and uses an improved idle time slot insertion-based optimizing policy to make the task scheduling more efficient.

Recently, the Modified Predict Earliest Finish Time algorithm (MPEFT) [20] was proposed. Equation 12 presents how this algorithm ranks tasks.

$$rank_{AP}(t_i)=DCT(t_i)+\sum_{t_j\in OS(t_i)} DCT(t_j) \tag{12}$$

The MPEFT ranking equation considers all direct and indirect successor tasks. Thus, instead of only using $succ$ as the other algorithms, MPEFT also considers all indirect tasks according to Equation 13.

$$OS(t_i)=\begin{cases}\bigcup_{t_j\in succ(t_i)} OS(t_j) \iff t_j \not t_{exit}\\ \emptyset \text{ otherwise}\end{cases} \tag{13}$$

MPEFT ranking considers DCT, where the minimal computational time for a given task t_i (represented by $w^*(t_i)$) is the sum with the summation of the average transfer costs $c_{i,j}$. This is expressed by Equation 14.

$$DCT(t_i) = w^*(t_i) + \sum_{t_j \in succ(t_i)} c_{i,j} \tag{14}$$

Similarly to PEFT, MPEFT does not directly consider the Estimated Finish Time (EFT) but incorporates it into another task's estimated finish, represented by MEFT (Equation 15).

$$MEFT(t_i,p_k)=EFT(t_i,p_k)+OCT(t_j,p_k)\times k(t_i,p_k) \tag{15}$$

The MEFT equation also considers the OCT value (Equation 7) as performed in PEFT but also uses a weight for the OCT value. This weight is calculated according to Equation 16.

$$k(t_i,p_k)=\begin{cases} \frac{rank_{AP}(CPS(t_i,p_k))}{\sum_{t_j \in succ(t_i)} rank_{AP}(t_j)+c_{i,j}} \iff |succ(t_i)| > |P|+1 \\ 1 \text{ otherwise} \end{cases} \tag{16}$$

Where the CPS is essentially the argmax of Equation 7, this is given by Equation 17, which is pretty similar to Equation 7.

$$CPS(t_i,p_k)=\text{argmax}_{t_j \in succ(t_i)}[\min_{pw \in P}\{OCT(t_j,p_w)+w(t_j,p_w)+c_{i,j}\}] \tag{17}$$

In all presented heuristics, tasks are ordered according to the heuristic criterion. Based on that order, tasks are assigned to the processor that is the most early available. The best available processor is found by considering the time that the processor will be available (last task executing in the processor + task duration) + the amount of time to send the task to that processor.

2.3 Meta-heuristics

Meta-heuristics are algorithms, solution methods that orchestrate an interaction between local improvement procedures (heuristics) and higher-level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space [12]. Thus, heuristics usually specializes in solving problems for one particular domain, while meta-heuristics are more generic and adaptive in several domains.

One of the most used meta-heuristic is the Evolutionary Algorithm (EAs) [8]. EAs are meta-heuristics that employ Darwin's theory of the survival of the fittest as their inspiration. This kind of algorithm generates solutions using the crossover and mutation heuristic operators. It employs an objective function to choose which individuals (solutions for the problem) will compose the next population.

Multi-Objective Evolutionary Algorithm (MOEA) has been successfully applied to the solution of MOPs (Multi-objective optimization problems) [5]. These algorithms are heuristic techniques that allow a flexible representation of the solutions and do not impose continuity conditions on the functions to be optimized.

Moreover, MOEAs are extensions of EAs for multi-objective problems that usually apply the concepts of Pareto dominance. These algorithms typically differ in their replacement strategies. In a replacement strategy, a population of current solutions P that was used to generate offspring solutions O is combined, generating $P \cup O$. So, a replacement strategy (RS) to generate a new population of solutions P' by $P' \leftarrow RS(P \cup O)$.

The Non-dominated Sorting Genetic Algorithm-II [9] (NSGA-II) performs the replacement strategy considering Pareto Dominance and Crowding Distance selection. This selection evaluates how close solutions are to their neighbors, giving a better evaluation of large values allows a better diversity in the population. Thus, NSGA-II selects surviving solutions from $P \cup O$, it first takes non-dominated solutions to compose P', and if the set size is smaller than the maximum population size, it iteratively adds dominated solutions with higher Crowding Distance values until P' is complete. If the set size exceeds the maximum population size, NSGA-II discards the solutions with lower Crowding values until P' is complete.

The Adaptive Geometry Estimation based MOEA (AGEMOEA) [18] inherits the overall framework of NSGA-II. However, it replaces the crowding distance with a survival score that combines diversity and proximity of the non-dominated fronts. First, this algorithm uses a fast heuristic with $O(M \times N)$ computational complexity to estimate the geometry of the front in each generation. Then, proximity is computed as the distance between each population member and the ideal point, while the diversity is measured using the distance among population members. The distance used to calculate proximity and diversity corresponds to the Lp norm that is associated with the estimated geometry.

A second version of the AGE Algorithm (AGEMOEA2) [19] incorporated the Newton-Raphson iterative method and the geodesic distance into the AGE-MOEA framework. This was performed by proposing a novel method for non-dominated front modeling using the Newton-Raphson iterative method. Second, they compute the distance (diversity) between each pair of non-dominated solutions using geodesics, which are generalizations of the distance on Riemann manifolds (curved topological spaces).

In this paper, we considered NSGA-II because this algorithm is probably the most popular MOEA in the literature and has had good results in recent works on cloud scheduling. AGEMOEA2 is one state-of-the-art algorithm for continuous multi-objective optimization benchmarks and was never applied to the cloud scheduling problem.

3 PROPOSAL

Figure 1 presents the HEACT workflow. The workflow to be scheduled is sent to the *Data preparation* process. This process is also responsible for getting real data about EC2 instance types from AWS (using the boto3 library). Then, with that information, for each region, the *Data preparation* calculates how long each task $t \in T$ demands to run in each machine. This creates the table W with size

$nr \times np \times nt$, where nr is the number of regions, np is the number of machines (this number varies in each region), and nt is the number of tasks in T.

To create W, we considered two pieces of information about EC2 instances taken from boto3: (1) the network performance and (2) the *Estimated Compute Unit* (ECU) of the instance type. According to AWS, "ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor" [22]. In order to use Simulation of Distributed Computer Systems (SimGrid) [6] as the simulator, we follow [14] and convert ECU to GigaFlops using the formula $GigaFlops = ECU * 4.2$.

Once W is created, the 'Data preparation' process disseminates this information to nr distinct processes, each tasked with finding solutions for a specific region. This is where the MOEA and a specialist scheduling heuristic come into play, running in parallel. MOEAs (eg. NSGA-II) are responsible for defining the set of machines to be used and heuristics (eg. HEFT) are responsible for assigning tasks to the selected machine. After completion, each process sends its final population to the 'Pareto dominance' analysis, eliminating dominated and repeated (same decision variables) solutions. The resulting solution is then stored as a JSON file.

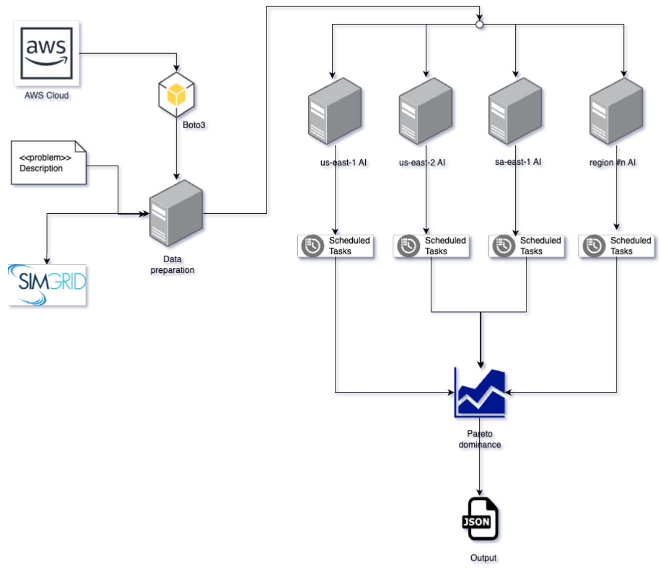


Fig. 1: HEACT Architectural representation

An essential part of an MOEA is how chromosomes are designed. In our approach, we used a fixed-size length integer array. Each array element represents an instance type, numbered from zero to np. When a number is negative, it means that this slot will not be used. That is a logical way to have a variable-size array.

This is important because it allows us to use more straightforward crossover operators in the MOEA. Figure 2 presents two examples of how the solution is defined.

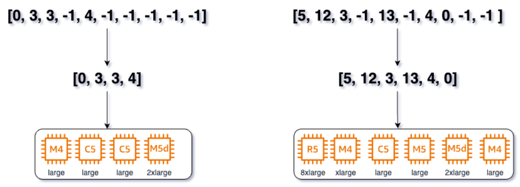


Fig. 2: MOEA chromosomes definition illustration

After generating a solution through crossover and mutation, the partial solution is sent to a specialized heuristic, which will be responsible for allocating tasks in machines specified by the previous partial solution. Thus, every task $t \in T$ must be assigned to a machine considering the performance characteristics of each EC2 instance type, such as network performance and how long it takes to solve a task (available in table W).

Tables 1 and 2 show an example of eight tasks being assigned to the solutions represented in Figure 2. Notice that not every machine will be used; this is something particular to the heuristic because it considers the cost of transferring gigabytes of data through the network or just waiting for a dependency task at the same machine to finish and having a network transfer cost of zero.

Machine	Task	Begin	End
M4 large	T4		
	T6		
	T2		
C5 large	T1	0	10
	T3		
	T5		
	T7		
	T8		
C5 large	-		
M5d 2xlarge	-		

Table 1: Solution #1

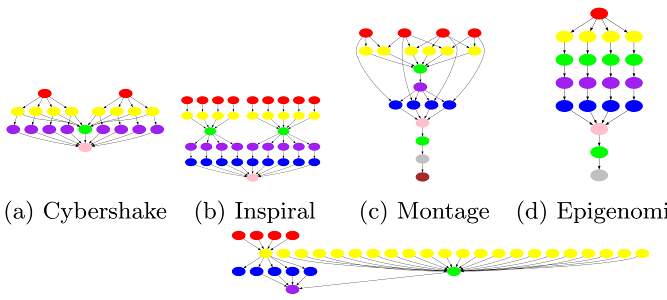
Machine	Task	Begin	End
R5 8xlarge	T4		
	T6		
	T2		
M4 xlarge	T1	0	10
	T3		
	T8		
C5 large	-		
M5 large	-		
M5d 2xlarge	T5		
	T7		
M4 large	-		

Table 2: Solution #2

Finally, the solution is updated by taking all the scheduling information (which task will run in each machine) and which machine will, in fact, be used. Thus, the solution [0,3,3,-1,4,-1,-1,-1,-1,-1] ([0,3,3,4]) becomes [0,3,-1,-1,-1,-1,-1,-1,-1,-1] ([0,3]) and [5,12,3,-1,13,-1, 4,0,-1,-1] ([5,12,3,13,4,0]) becomes [5,12,-1,-1,-1,-1, 4,-1,-1,-1] ([5,12,4]). With the complete solution, scheduling + machine types, we are able to calculate the objectives makespan and cost.

4 EXPERIMENTS

We run experiments to evaluate how HEACT performs. For this purpose, we considered several real-world workflows, such as Montage, CyberShake, Epigenomics, Inspiral, and Sipht [3]. We named the problems in the format problem_nodes. So, Montage_25 is an instance of Montage, which has 25 tasks. Figure 3 shows the different workflows.



(a) Cybershake (b) Inspiral (c) Montage (d) Epigenomics

(e) Sipht

Fig. 3: Real-world workflows used to evaluate HEACT

We created eight different instances by combining the meta-heuristics NSGA-II and AGEMOEA2 with scheduling heuristics HEFT, PEFT, HSIP, and MPEFT. The population size was set as 50. The number of generations was set to 700, totaling 35000 heuristic execution and solution evaluation, defined empirically. We considered 14 AWS regions. These are the regions that have EC2 instances with complete data about ECU. Each AWS region was assigned to a process that would run for 50 generations (or 2500 solution evaluations). For NSGA-II and AGEMOEA2, we considered a two-point crossover with a probability of 90% and flip mutation with a probability of 10%.

All algorithms were coded using Python 3.11. We used pymoo [4] for meta-heuristics, pysimgrid [23] and SimGrid 3.24 for the cloud simulation, and AWS boto3 1.28 to get AWS pricing data.

After running each of the eight different algorithm instances twenty times (for each problem), we took the resulting population and removed all the dominated solutions. After that, as commonly performed in the literature, we calculated the normalized Hypervolume [28] to get the Pareto Front's quality for each execution. After getting twenty hypervolume values (for each problem and each algorithm instance), we calculated the average and submitted them to the Kruskal-Wallis statistical test with 99% confidence, as commonly performed in the literature.

Table 3 presents Hypervolume results; highlighted values mean the highest hypervolume average, while bold values mean that the value is statistically tied with the best value. NS means NSGA-II while AGE means AGEMOEA2. We can see that no single algorithm statistically overcomes all the others. We can also see that instances that use PEFT as a heuristic have, in general, the poorest performance. However, regarding CyberShake_100, AGE+PEFT and NS+PEFT were indeed the best-performing algorithms. NS+HSIP have similar performance compared with PEFT instances by being the best at one problem and have statistically tied results with the best algorithm in one problem. AGE+HSIP is a bit better because it has the best average at one problem, but it is tied with the best in five problems and is overcome in nine problems.

Table 3: Hypervolume Averages

	AGE+HEFT	AGE+HSIP	AGE+MPEFT	AGE+PEFT	NS+HEFT	NS+HSIP	NS+MPEFT	NS+PEFT
Epigenomics_24	**0.9798**	0.9820	0.9790	0.9783	0.9796	**0.9821**	0.9800	0.9782
Montage_25	**0.9963**	**0.9961**	0.9959	0.9962	**0.9964**	0.9962	0.9955	0.9961
CyberShake_30	**0.9984**	0.9981	0.9980	0.9970	**0.9984**	0.9981	0.9980	0.9968
Inspiral_30	0.9980	0.9956	**0.9980**	0.9977	0.9979	0.9941	**0.9981**	0.9977
Sipht_30	0.9499	**0.9511**	0.9499	0.7736	0.9499	0.9511	0.9498	0.8126
Epigenomics_46	**0.9807**	**0.9800**	**0.9794**	0.6488	**0.9801**	0.9776	0.9778	0.6170
CyberShake_50	**0.9972**	**0.9975**	0.9975	**0.9974**	0.9974	0.9973	**0.9975**	0.9974
Inspiral_50	**0.9888**	0.9877	**0.9901**	0.5024	**0.9895**	0.9863	0.9897	0.5024
Montage_50	0.9827	0.9920	**0.9978**	0.9976	0.9923	0.9925	**0.9977**	0.9976
Sipht_60	**0.9805**	0.9761	0.9771	0.9732	**0.9805**	0.9760	0.9773	0.9717
CyberShake_100	0.9894	0.9939	0.9972	**0.9981**	0.9952	0.9920	0.9972	**0.9981**
Epigenomics_100	**0.9894**	**0.9909**	**0.9926**	0.9822	**0.9765**	**0.9885**	**0.9923**	0.9844
Inspiral_100	**0.9905**	0.9744	0.9693	0.0771	**0.9902**	0.9449	**0.9846**	0.1542
Montage_100	0.9790	**0.9966**	**0.9986**	0.9794	**0.9976**	0.9974	0.9978	0.9809
Sipht_100	**0.9862**	0.9852	**0.9850**	0.9801	**0.9856**	0.9836	0.9841	0.9767

Based on the statistical results, we sum how many problems each algorithm instance got the best hypervolume average, tied with the best, and got the averages statistically worse when compared with the best algorithm. Table 4 presents this analysis. Moreover, we follow [10] and perform the ranking of algorithms considering the Friedman ranking.

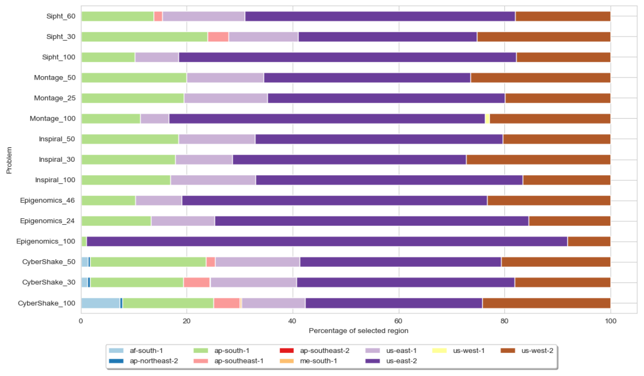
According to Friedman Ranking, AGE+MPEFT is the best algorithm. This happens especially because it is the algorithm that has been the best more times. The ranking states that NS+HEFT, AGE+HEFT, and NS+MPEFT are in sequence. However, an interesting aspect is that NS+HEFT is the fourth algorithm in the ranking, but this is the algorithm with fewer overcomes (Qtd worse column). By this analysis, we can conclude that AGE+MPEFT and AGE+HEFT are the most suitable algorithms in general.

Another interesting aspect to evaluate is how solutions are spreading across the world. Due to this fact, we looked for solutions for each region independently and then selected the surviving ones using the Pareto non-dominance criterion. One region may have more solutions than another. Figure 4 presents this percentual. We can see that always us-east-2 (US East (Ohio)) have more solutions. The second best is us-west-2 (US West (Oregon)) followed by ap-south-1 (Asia Pacific (Mumbai)) and us-east-1 (US East (Virginia)). Of course, these values can change depending on the price AWS assigns to instances.

Table 4: Average Rankings of the algorithms (Friedman)

Algorithm	Qtd. Best	Qtd. tied	Qtd worse	Friedman Ranking
AGE+MPEFT	4	3	8	3.4334
NS+HEFT	3	6	6	3.4667
AGE+HEFT	3	7	5	3.5335
NS+MPEFT	2	3	10	3.9
AGE+HSIP	1	5	9	4.2667
NS+HSIP	1	1	13	5.0667
AGE+PEFT	1	1	13	6.0334
NS+PEFT	0	1	14	6.3001

Fig. 4: Percentual of non-dominated solutions considering regions



5 Conclusions and Further work

Planning the execution of a workflow is a challenging task. Over the years, several approaches have been proposed in order to tackle this problem. This present paper introduced HEACT, a novel approach that finds solutions for workflow scheduling by first running a multi-objective evolutionary algorithm in order to create the set of machines and then submitting this setup to a specialist heuristic responsible for the task assigning. This process was performed considering different regions, each one with its own set of instance types. Our proposal considered accurate pricing information from AWS and simulated the workflow execution considering the simgrid simulator. Moreover, fourteen AWS regions were considered.

We instanced our approach by considering heuristics, such as HEFT, PEFT, HSIP, and MPEFT, as well as meta-heuristic NSGA-II and AGEMOEA2. Results showed that AGEMOEA2 + MPEFT and AGEMOEA2 +HEFT were the best combinations. Thus, AGEMOEA2, a recent state-of-art meta-heuristic, seems promising for this kind of problem. Moreover, MPEFT, a recent work, seems a really interesting heuristic for scientific workload problems, as also verified in [20]. Further work may focus on combining these heuristics and meta-heuristics to create a hyper-heuristic aiming for the best overall performance. This is important because no single algorithm was the best across all the problems. Moreover, information about carbon emissions should also be considered in future work.

Acknowledgments. We gratefully acknowledge financial support from the São Paulo Research Foundation (FAPESP) grant 2023/15949-8.

References

1. Ahmad, D., Gautam, G., Alam, B., Bhati, B.: An analytical review and performance measures of state-of-art scheduling algorithms in heterogenous computing enviornment. Archives of Computational Methods in Engineering (02 2024)
2. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems **25**(3), 682–694 (2014)
3. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: 2008 Third Workshop on Workflows in Support of Large-Scale Science. pp. 1–10 (2008)
4. Blank, J., Deb, K.: pymoo: Multi-objective optimization in python. IEEE Access **8**, 89497–89509 (2020)
5. Boussaid, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237**, 82 – 117 (2013)
6. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014)
7. Chen, J., Menzies, T.: RIOT: A Stochastic-Based Method for Workflow Scheduling in the Cloud. IEEE International Conference on Cloud Computing, CLOUD **2018-July**, 318–325 (2018)
8. Coello, C.: Evolutionary algorithms for solving multi-objective problems. Springer, New York (2007)
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. Evolutionary Computation, IEEE Transactions on **6**(2), 182–197 (Apr 2002)
10. Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation **1**(1), 3 – 18 (2011)
11. Durillo, J.J., Fard, H.M., Prodan, R.: Moheft: A multi-objective list-based method for workflow scheduling. In: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings. pp. 185–192 (2012)
12. Gendreau, M., Potvin, J.Y.: Handbook of Metaheuristics. Springer Publishing Company, Incorporated, Berlin Heidelberg, 2nd edn. (2010)
13. Hosseini Shirvani, M., Noorian Talouki, R.: Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach. Complex & Intelligent Systems **8** (11 2021)
14. Iosup, A., Ostermann, S., Yigitbasi, M.N., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions on Parallel and Distributed Systems **22**(6), 931–945 (2011)

15. Keshanchi, B., Souri, A., Navimipour, N.J.: An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing. Journal of Systems and Software **124**, 1–21 (2017)
16. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. Tech. Rep. 826, Caltech Concurrent Computation Program, C3P (1989)
17. Nasonov, D., Visheratin, A., Butakov, N., Shindyapina, N., Melnik, M., Boukhanovsky, A.: Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment. Journal of Applied Logic **24**, 50–61 (2017), sI:SOCO14
18. Panichella, A.: An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 595–603. GECCO '19, Association for Computing Machinery, New York, NY, USA (2019)
19. Panichella, A.: An improved pareto front modeling algorithm for large-scale many-objective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 565–573. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022)
20. Pu, J., Meng, Q., Chen, Y., Sheng, H.: Mpeft: A novel task scheduling method for workflows. Frontiers in Environmental Science **10** (2023). `https://doi.org/10.3389/fenvs.2022.996483`
21. Qin, S., Pi, D., Shao, Z., Xu, Y., Chen, Y.: Reliability-Aware Multi-Objective Memetic Algorithm for Workflow Scheduling Problem in Multi-Cloud System. IEEE Transactions on Parallel and Distributed Systems **34**(4), 1343–1361 (2023)
22. Services, A.W.: Amazon ec2 instance-types. `https://www.amazonaws.cn/en/ec2/instance-types/` (2024), accessed on 21th May 2024
23. Sukhoroslov, O.: Toward efficient execution of data-intensive workflows. J. Supercomput. **77**(8), 7989–8012 (aug 2021)
24. Sulaiman, M., Halim, Z., Lebbah, M., Waqas, M., Tu, S.: An evolutionary computing-based efficient hybrid task scheduling approach for heterogeneous computing environment. Journal of Grid Computing **19** (03 2021)
25. Sulaiman, M., Halim, Z., Waqas, M.M., Aydın, D.: A hybrid list-based task scheduling scheme for heterogeneous computing. The Journal of Supercomputing **77**, 10252 – 10288 (02 2021)
26. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling forheterogeneous computing. Parallel and Distributed Systems, IEEE Transactions on **13**, 260–274 (04 2002)
27. Wang, G., Wang, Y., Liu, H., Guo, H.: Hsip: A novel task scheduling algorithm for heterogeneous computing. Scientific Programming **2016**, 1–11 (01 2016)
28. While, L., Bradstreet, L., Barone, L.: A fast way of calculating exact hypervolumes. IEEE Trans. Evol. Comput. **16**(1), 86–95 (Feb 2012)
29. Zhang, Q., Liu, W., Li, H.: The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. Tech. Rep. CES-491, School of CS & EE, University of Essex (Feb 2009)
30. Zhu, Z., Zhang, G., Li, M., Liu, X.: Evolutionary Multi-Objective Workflow Scheduling in Cloud. IEEE Transactions on Parallel and Distributed Systems **27**(5), 1344–1357 (2016)
31. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: Evolutionary Methods for

Design Optimization and Control with Applications to Industrial Problems. pp. 95–100. International Center for Numerical Methods in Engineering, Barcelona, Spain (2001)