Contents lists available at ScienceDirect

Array

journal homepage: www.sciencedirect.com/journal/array

An efficient ACO-based algorithm for task scheduling in heterogeneous multiprocessing environments

Jeffrey Elcock [*], Nekiesha Edward

Department of Computer Science, Mathematics and Physics, University of the West Indies, Cave Hill Campus, Bridgetown, Barbados

A R T I C L E I N F O

Keywords:
Task scheduling
Heterogeneous
Ant colony optimization
Directed acyclic graphs

A B S T R A C T

In heterogeneous computing environments, finding optimized solutions continues to be one of the most challenging problems as we continuously seek better and improved performances. Task scheduling in such environments is *N*P-hard, so it is imperative that we tackle this critical issue with a desire of producing effective and efficient solutions. For several types of applications, the task scheduling problem is crucial, and throughout the literature, there are a plethora of different algorithms using several different techniques and varying approaches. Ant Colony Optimization (ACO) is one such technique used to address the problem. This popular optimization technique is based on the cooperative behavior of ants seeking to identify the shortest path between their nest and food sources. It is with this in mind that we propose an ACO-based algorithm, called ACO-RNK, as an efficient solution to the task scheduling problem. Our algorithm utilizes pheromone and a priority-based heuristic, known as the upward rank value, as well as an insertion-based policy, along with a pheromone aging mechanism which aims to avoid premature convergence to guide the ants to good quality solutions. To evaluate the performance of our algorithm, we compared our algorithm with the HEFT algorithm and the MGACO algorithm using randomly generated directed acyclic graphs (DAGs). The simulation results indicated that our algorithm experienced comparable or even better performance, than the selected algorithms.

1. Introduction

As computing hardware continues its amazing pursuit into new frontiers, we must be cognizant that the software aspect ought to keep abreast. Rapid improvements and technological advances in computer architecture have over the years, continue unabated. The ever presence of heterogeneous multiprocessor systems, no doubt, keep gaining increasing popularity because of their varied yet incredible capabilities [1]. This increased parallelism has brought with it diversity, high throughput, along with the potential of higher and even better performances. So to fully exploit these environments, task scheduling continues to be resolutely pursued in order to reap and maximize whatever benefits these computer systems have to offer. Task scheduling, is defined as the assignment of tasks of a parallel application to different processors in a manner that minimizes the overall completion time or schedule length (SL) of the application while ensuring that all constraints or interdependencies are fully satisfied [2]. In a heterogeneous computing environment, scheduling of these interdependent tasks becomes a lot more challenging, because of the different processor speeds along with the varying computational cost associated with each task [3].

A program or parallel application may be modeled by a task graph in the form of a weighted directed acyclic graph (DAG), $G = (V, E)$, where V denotes the set of nodes (n_i) which represent the tasks of the application and E denotes the set of edges that indicate the data dependencies between the various tasks. The weight on each edge represents the communication cost between two nodes while the weight of each task denotes its computational cost which may or may not vary depending of which processor it is executed [4].

In an instance where $(n_i, n_j) \in E$ then n_i is called the parent or immediate predecessor of task n_j, and n_j is called the child or immediate successor of n_i. If a task n_a, has more than one immediate predecessor, then n_a is referred to as a joined task. The immediate successors of n_a is denoted by *Imsuc* (n_a) and is defined as $\{ n_j \mid (n_a, n_j) \in E\}$, while its set of immediate predecessors, denoted by *Impred* (n_a), is defined as $\{ n_j \mid (n_j, n_a) \in E\}$. Before task n_a, can be scheduled, all of its parent tasks or nodes must first be scheduled. Additionally, the critical path of a DAG is the longest path from the entry node to the exit node, considering both the computation and communication costs between the tasks [5]. It is

* Corresponding author.
E-mail address: jeffrey.elcock@cavehill.uwi.edu (J. Elcock).

https://doi.org/10.1016/j.array.2023.100280
Received 1 November 2022; Received in revised form 16 January 2023; Accepted 20 January 2023
Available online 21 January 2023

assumed that there is one entry task (n_{entry}) which has no predecessor nodes, and one exit task (n_{exit}), which is a node with no successors for the DAG. If a DAG contains multiple entry or exit tasks, a dummy entry or exit node with zero computation cost, along with a zero communication cost can be connected, therefore making our algorithm applicable for DAGs of any kind.

The focus of our research is static task scheduling, where information about available resources is known before execution and scheduling may be done at compile time. Whether static or dynamic, task scheduling is classified as an *N*P-Hard problem [4,6]. However, it has been well studied and a number of suboptimal heuristic-based solutions have been proposed. These solutions may be categorized as list scheduling, clustering, task duplication and guided random search methods.

List scheduling algorithms [7–10] approach the scheduling problem by first prioritizing tasks, based on various criteria, into an ordered list before mapping each onto a processor with the earliest completion time. Generally, list scheduling algorithms have good performance-to-cost tradeoffs; however the choice of rank can affect the makespan. Clustering algorithms [4,11–13] attempt to reduce the communication cost associated with dependent tasks by grouping communicating nodes together into clusters with the intention of scheduling them onto the same processor. The rationale of this approach is the fact that when two inter-dependent tasks are placed on the same processor, the communication cost between them becomes negligible. However, if groupings are not carefully selected, this approach can lead to inefficient schedule lengths. The basis of task duplication algorithms [14–16] is to, where possible, remove the cost associated with inter-processor communication, and thus reduce the overall schedule length, by duplicating some predecessor nodes [4]. Task duplication, like clustering, takes advantage of the zero or negligible communication cost when two dependent tasks are placed on the same processor. The redundancy in task duplication can however, lead to the inefficient use of the processors. Guided random search algorithms [5,17–20] on the other hand, examine multiple solutions in the search space and converge to an efficient solution. Such algorithms, like Genetic Algorithms and Ant Colony Optimization (ACO) have been applied to the task scheduling problem producing some very good results.

Given the versatility of ACO algorithms, we present an ACO-based algorithm. Our proposed algorithm incorporates the upward ranking concept, found in the HEFT algorithm [8], in our prioritization methodology; an insertion-based policy and a pheromone aging mechanism which seeks to create new opportunities, to produce efficient schedules. Our research investigates the application of an efficient solution to the static task scheduling problem in a heterogeneous environment.

For our scheduling system model, the target computing environment consists of a set of processors P, where P = { $p_1, p_2, p_3, \ldots p_{|P|}$ }, and |P| denotes the number of processors. Our model assumes heterogeneous non-preemptive processors in a fully connected topology and the inter-processor communication is contention-free. The main objective of the task scheduling problem is to determine a mapping of tasks of a given application to processors that minimizes the overall schedule length or makespan.

The remainder of the paper is organized as follows: We describe the ACO metaheuristic in Section II and outline our proposed algorithm in Section III. Results obtained from a performance comparison of the HEFT [8] and MGACO [21] with our proposed work are discussed in Section IV and we present our conclusions and future works in Section V.

2. The aco metaheuristic

Ant Colony Optimization (ACO) algorithms were first introduced by Dorigo and his colleagues in the early 1990s [22]. These algorithms form part of a wider research area known as Swarm Intelligence, which models solutions to combinatorial, and optimization problems, based on the behavior of certain species found in nature. ACO is inspired by the indirect communication of a foraging ant colony, where colony survival governs the ants' behavior and not simply individual survival. This indirect communication, known as stigmergy, enables ants to move in a coordinated manner to find very short paths between food sources and their nest [23].

In the initial stages of food searching, ants explore randomly. When food is encountered, the quality and quantity is assessed and pheromone from the food source to back to the nest is deposited. Subsequent foraging ants utilize these said pheromone trails to guide them to the food, with the high probability of using paths marked with strong pheromone concentrations, which reinforces the pheromone density and thus increasing their attractiveness for later ants. This reinforcement leads to convergence to the most attractive or shortest path since evaporation of pheromones on the trails provides the limiting mechanism for this positive feedback, so less frequented paths have decreased pheromone concentration.

The ACO metaheuristic (Fig. 1) incorporates the foraging behavior of natural ants in a computational environment and iteratively constructs solutions using artificial pheromone and local heuristics to guide the artificial agents (ants) through the investigated search space. The pheromone trails influences future agents toward high quality solutions, until a termination condition is satisfied.

Contrary to foraging ants in nature which deposit continuous pheromone trails, ACO approaches have implemented various alternatives [24]. For example, in the original Ant System (AS) [22] ants deposit pheromone only to completed solutions. Alternatively, the Ant Colony System (ACS) [25] makes step-by-step online (local) pheromone deposits by every agent during the development or building of solutions and introduces a further offline (global) update of pheromones to the best solution of the iteration. Furthermore, some kind of evaporation mechanism or technique is implemented, allowing the ants to consider new search areas within the search space [24]. Additionally, some ACO techniques employ local and global optimization strategies to further increase the quality of their solutions.

The ACO mechanism has been applied to various optimization and scheduling problems [26]. It has been integrated with other random search algorithms, for example, the Genetic Algorithm and Tabu Search. ACO has also been combined or incorporated with list scheduling, for instance, the ANT-LS algorithm [24] and the ACO-TMS [21]. This combination of pheromone trails and list scheduling heuristics further enhances the possibility of good quality schedules.

3. The proposed algorithm

3.1. Overview of the algorithm

Our proposed algorithm (Fig. 2) is known as the rankUP-Ant Colony System (ACO-RNK). It combines the foundation of the Ant Colony System (ACS) with the heuristic function of list scheduling algorithm HEFT along with some random searches whenever stagnation or aging presents itself. ACS exhibits some flexibility with the utilization of the

```
The ACO Metaheuristic

Set parameters, initialize pheromone trails
while termination condition not satisfied do
    Construct Ant Solutions
    Apply Local Optimization (optional)
    Pheromone Update
end while
```

Fig. 1. The aco metaheuristic.

```
Begin
Set DAG parameters, initialize pheromone trails & ready
list
while termination conditions not satisfied do
    # CONSTRUCT ANT SOLUTIONS
    For each iteration
        Repeat for each ant
            Initialise RdyList
            While RdyList NOT empty
                if (first iteration or AGing)
                    select task from RdyList randomly
                    selects ProC randomly
                else
                    select task from RdyList using
                    STask Rule
                    selects ProC using SProc Rule
            Update RdyList
            end while
        end Repeat
        unset AGing
      PHEROMONE UPDATE
        Local pheromone update using LoPU Rules
        Global pheromone update using GoPU Rules

    If AGING MECHANISM applied
        set AGing
  end iteration
end while
End
```

Fig. 2. The aco-rnk algorithm.

offline pheromone update and HEFT has yielded good results as a list scheduling algorithm, with the use of the upward rank value in its prioritization.

First we identify and initialize our two matrices for our pheromone representation. $V \times P$, which we denote as τ, and $P \times V$ which we denote as σ. The entry $\tau(i,p)$ indicates the pheromone on the edge between task i and processor element p, whereas $\sigma(p,j)$ indicates the pheromone on the edge between processor element p and task j. Therefore, if $n_{entry} \rightarrow p_2 \rightarrow n_3 \rightarrow p_1 \rightarrow n_2 \rightarrow p_{|P|} \rightarrow \ldots . \rightarrow n_{exit} \rightarrow p_1$ is a possible solution or priority queue (a complete mapping of the task graph where an ant, starts at the entry node (n_{entry}) moves from task to processor and from processor to task, until a processor has been selected for the exit node (n_{exit})), then τ (n_3, p_1) ε $V \times P$ and σ (p_1, n_2) ε $V \times P$. Initially, a small pheromone deposit is made to all elements of each matrix and the ready list (RdyList) is initialized with the entry node.

Our iterative algorithm executes as follows: for each ant, in each iteration, create an ant list of length V that stores both a task and its selected processor. The ant selects a task from the ready list using the state transition (STask) rule (1) and a processor using the state transition (SProc) rule (4) to construct a schedule. The selected task is removed from the ready list, and appended, along with the processor, to the ant list. The ready list is then updated to contain those children nodes whose parents have already been scheduled. This process is repeated until all the tasks have been mapped. During the first iteration, our algorithm ACO-RNK, does not employ the state transition rules to select either task or processor – they are both selected in a random manner – thus mimicking the ants' natural environment. Throughout the execution of the algorithm, an insertion-based policy is employed whereby the task is checked to see if it can be scheduled earlier on the chosen processor, thus affording our approach, the opportunity to achieve shorter schedules.

After each iteration, an online or local pheromone update is applied to the best q ants ($q \leq \bar{A}$) where $\bar{A}$ is the number of ants per iteration, according to the local pheromone updating (*LoPU*) rule using (6), (7) and (8). Following this update, there is also an offline or global update to the best ant solution according to the global pheromone updating (*GoPU*) rule using (9), (10) and (11).

Our algorithm also attempts to alleviate stagnation by employing a pheromone aging mechanism (represented as ₱). This condition monitors how the best solution changes over the course of the execution of the algorithm. If the value for the best schedule length remains unchanged after a predetermined number of iterations, a deliberate evaporation of the pheromone trail of that schedule is invoked. When invoked, a random value is generated for ₱, which is always less than or equal to the initial pheromone, and applied. Also for the next iteration, all ants are allowed to freely select both task and processor without state transition rules. Repetition of these. Conditions is dependent on the random generation of a value which is equal to, or less than the total number of iterations of the algorithm. This deliberate evaporation and random selection facilitate the increased probability of exploring new, and possibly better quality solutions.

3.2. Criteria for task and processor selection

The two transition rules which govern our task and processor selections as used by our proposed algorithm are as follows:

Task Selection Rule (STask): For each ant, a task (i) is selected from the ready list (RdyList) according to the probability as follows:

$$P_{rob}(i) = \frac{[\tau(i,p)]^{\delta} \cdot [\eta(i)]^{\theta}}{\sum_{t \in RdyList} [\tau(t,p)]^{\delta} \cdot [\eta(t)]^{\theta}} \tag{1}$$

where the edge, $\tau(i,p)$, indicates the pheromone between task (i) and processor element (p) while δ and θ indicate the influence of the heuristic function and pheromone such that $\theta \in \{0,1,2\}$ and $\delta \in \{0,1,2\}$ and $\eta(i)$ is the list scheduling heuristic function evaluated by

$$\eta(i) = \frac{1}{Urank(i)} \tag{2}$$

The upward rank, $Urank(i)$ is the longest path from a particular node to the exit node, and is recursively defined as

$$Urank(i) = \overline{wt_i} + \max_{j \in \mathrm{Im}succ(i)} \left[\overline{c_{i,j}} + Urank(j)\right] \tag{3}$$

where $\mathrm{Im}succ(i)$ denotes the immediate successors of node n_i, $\overline{wt_i}$ is the average computation cost of n_i and $\overline{c_{i,j}}$ is the average communication on edge *(i,j)*.

Processor Selection Rule (SProc): An ant selects a processor element (v) for a chosen task (j) based on a probability derivied by

$$P_{rob}(v) = \frac{[\sigma(v,j)] \cdot [\eta_*(v)]^{\psi}}{\sum_{p \in P} [\sigma(p,j)] \cdot [\eta_*(p)]^{\psi}} \tag{4}$$

where

$$\eta_*(v) = \frac{1}{AComC(j)} \cdot \frac{1}{EST(j,v)} \tag{5}$$

and *AComC(j)* is the average computation cost of the node j on the processors, and the *EST (j,v)* is the estimated start time of the node j on processor v. Ψ, such that $\Psi \in \{0,1,2\}$, is the influence on the probability and p is a component of the set of processors (P).

3.3. Criteria for pheromone update

After construction of the ant solutions a local and a global pheromone update is applied to the best ant solution of the iteration.

Local Pheromone Update (LoPU): Our algorithm applies pheromone to edges *(i, p)* and *(v, j)* based on the following functions

$$\tau(i,p) = (1-\rho)\cdot\tau(i,p) + \Delta\tau \tag{6}$$

$$\sigma(v,j) = (1-\rho)\cdot\sigma(v,j) + \Delta\sigma \tag{7}$$

where

$$\Delta\sigma = \Delta\tau = \frac{1}{FinT_{(ant)}} \tag{8}$$

and $FinT_{(ant)}$ is the completion time of the ant while ρ is the pheromone decay parameter $(0 < \rho < 1)$.

Global Pheromone Update (GoPU): Application of pheromone to the edges *(i, p)* and *(v, j)* of the ant with the best solution based on the following functions:

$$\tau(i,p) = (1-a)\cdot\tau(i,p) + a\cdot\Delta\tau \tag{9}$$

$$\sigma(v,j) = (1-a)\cdot\sigma(v,j) + a\cdot\Delta\sigma \tag{10}$$

where

$$\Delta\sigma = \Delta\tau = \frac{1}{FinT_{(best)}} \tag{11}$$

and $FinT_{(best)}$ is the best finish time of the iteration while a denotes the pheromone decay or evaporation parameter $(0 < a < 1)$.

4. An ilustrative example

For the DAG in Fig. 3(a), the HEFT algorithm has a scheduling order of {n_1, n_4, n_2, n_3, n_6, n_5, n_9, n_8, n_7, n_{10}}. Using this scheduling order, Table 1 shows the actual schedule with a length of 93. The MGACO algorithm, which generated its best priority queue of $n_1 \rightarrow p_3 \rightarrow n_2 \rightarrow p_3 \rightarrow n_6 \rightarrow p_3 \rightarrow n_4 \rightarrow p_2 \rightarrow n_3 \rightarrow p_1 \rightarrow n_7 \rightarrow p_1 \rightarrow n_5 \rightarrow p_2 \rightarrow n_8 \rightarrow p_1 \rightarrow n_9 \rightarrow p_2 \rightarrow n_{10} \rightarrow p_2$ shows in Table 2, with an actual schedule of length 89. Our algorithm ACO-RNK generated its best priority queue of, $n_1 \rightarrow p_3 \rightarrow n_2 \rightarrow p_3 \rightarrow n_5 \rightarrow p_3 \rightarrow n_4 \rightarrow p_1 \rightarrow n_6 \rightarrow p_1 \rightarrow n_3 \rightarrow p_2 \rightarrow n_7 \rightarrow p_2 \rightarrow n_9 \rightarrow p_2 \rightarrow n_8 \rightarrow p_1 \rightarrow n_{10} \rightarrow p_2$ with Table 3 showing the actual schedule of length 83. The use of the upward rank value and our anti-aging approach, influence the ants' construction, in facilitating searches for solutions in areas of the search space containing high quality solutions.

Table 1
The schedule of the HEFT algorithm for the DAG in Fig. 3(a).

NODE	1	2	3	4	5	6	7	8	9	10
Proc	3	1	3	2	3	2	3	1	2	2
StartT	0	31	11	20	34	30	46	65	62	84
FinishT	11	46	34	30	46	48	59	72	72	93

Table 2
The schedule of the MGACO algorithm for the DAG in Fig. 3(a).

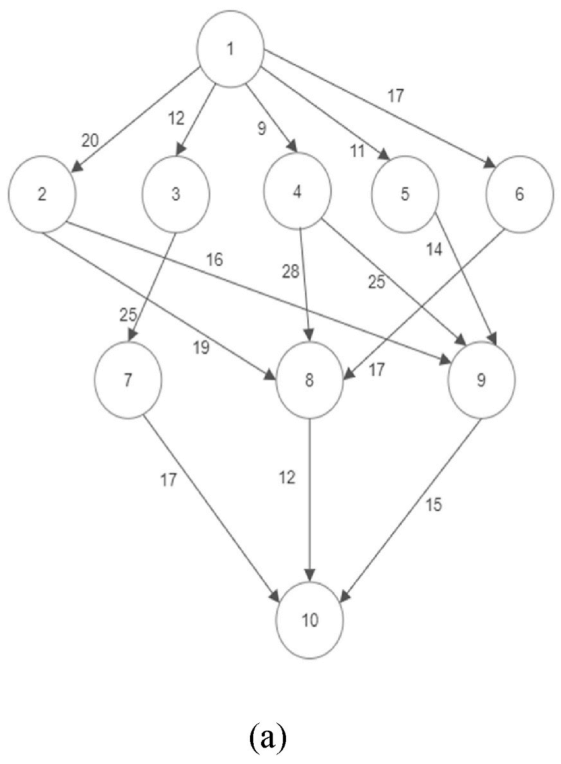
NODE	1	2	3	4	5	6	7	8	9	10
Proc	3	3	1	2	2	3	1	1	2	2
StartT	0	11	23	20	30	33	36	61	49	80
FinishT	11	33	36	30	45	44	45	68	63	89

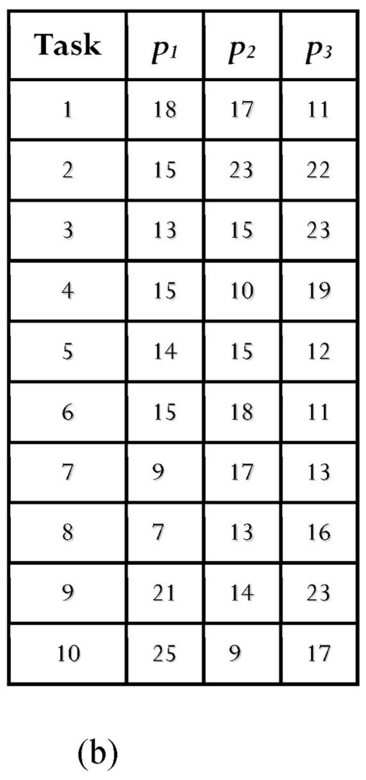
Task	p_1	p_2	p_3
1	18	17	11
2	15	23	22
3	13	15	23
4	15	10	19
5	14	15	12
6	15	18	11

(*continued on next page*)

Table 3
The schedule of the ACO-RNK algorithm for the DAG in Fig. 3(a).

NODE	1	2	3	4	5	6	7	8	9	10
Proc	3	3	2	1	3	1	2	1	2	2
StartT	0	11	23	20	33	35	38	52	60	74
FinishT	11	33	38	35	45	50	55	59	74	83





Task	p_1	p_2	p_3
1	18	17	11
2	15	23	22
3	13	15	23
4	15	10	19
5	14	15	12
6	15	18	11
7	9	17	13
8	7	13	16
9	21	14	23
10	25	9	17

(b)

Fig. 3. Illustrates (a) an arbitrary DAG representing a task graph (an application); (b) the computation cost of each task on the different processors.

(*continued*)

7	9	17	13
8	7	13	16
9	21	14	23
10	25	9	17

5. Performance evaluation

The performance of the proposed algorithm (ACO-RNK) was evaluated by comparing it with its progenitor, the HEFT algorithm [8], and the MGACO algorithm [21]. For this comparison, a number of random directed acyclic graphs (DAGs) of varying attributes were generated and executed by the algorithms.

5.1. Comparative metrics

1. **Speedup**: This value is defined as the ratio between both the sequential and the parallel execution time of a process or application. The sequential time is calculated by sequentially adding the computational cost of each task in the graph.

This summation is obtained for each processor and the smallest value is used. The parallel execution time, on the other hand, also referred to as the Makespan or Scheduled Length (SL), is the completion time of the graph. Therefore

$$Speedup = \frac{\min_{p_k \in P}\left\{\sum_{n_i \in V} \omega(n_{i,k})\right\}}{SL} \tag{12}$$

where $\omega(n_{i,k})$ denotes the computational cost of task n_i on processor p_k.

2. **Schedule Length Ratio**: For a scheduling algorithm, its main performance measure is the Schedule Length (SL). Our experiment uses a large set of DAGs with various properties, and

Therefore, it becomes necessary to normalize the schedule length to the lower bound. This is known as the Schedule Length Ratio (SLR). The SLR value is defined as follows:

$$SLR = \frac{SL}{\sum_{n_i \in CriP_{MIN}} \min_{p_k \in P}\{\omega(n_{i,k})\}} \tag{13}$$

The $CriP_{MIN}$ value in the denominator is calculated using the following method. Firstly, each task (n_i), is set to its minimum computational cost, and secondly, the length of the Critical Path (|CP|) is then derived by summing these values.

5.2. Attributes of the DAGs

5.2.1. Randomly generated DAGS

In our experiment, the following input parameters were utilised for the generation of the task graphs, which were also used in Ref. [8].

- Number of tasks in the DAG ($|V|$).
- OutDegree of a node (O_{deg}). This is the maximum number of children or immediate predecessors of a node.
- Shape parameter of the graph (α).
- Communication to computation ratio (*CCR*). It is the ratio between the average communication cost and the average computation cost.
- Range percentage of computation costs on processors (β). It is the heterogeneity factor for processors. A higher percentage value indicates a significant difference in the computation cost across the processors, while lower values are indicative of more subtle differences in computation costs.

For the random generated graphs used in our experiment, the values were assigned from the sets given below.

- Number of Ants ($\bar{A}$) = {min ((*avg* (O_{deg}) × |V|, 100)}
- Number of Iterations = {100}
- Number of processors = {3}
- Set of Nodes (V) = {100, 200, 300, 400}
- Set of CCR (*CCR*) = {0.1, 0.5, 1.0, 5.0, 10.0}
- Set of Alpha (α) = {0.5, 1.0, 2.0}
- Set of OutDegree (O_{deg}) = {1, 2, 3, 4, 5}
- Set of Beta (β) = { 0.25, 0.5, 0.75, 1.0}

5.3. Performance comparison using randomly generated DAGs

For the comparison with HEFT and the MGACO a total of 13,500 random graphs with the varying characteristics were generated and executed.

Comparative analysis of the algorithms was then done based on the average makespan attained with the varying shape parameters. DAGs generated in this experiment were of varying degrees of parallelism. From the results in Fig. 4, ACO-RNK outperformed the other two algorithms for short graphs of high parallelism (larger α values). When α ≥ 1, it was 6.5% better than MGACO and 24% better than HEFT. With α < 1, where the graphs have greater depth and a low parallelism, it shows that HEFT is a good candidate as it was slightly better than ACO-RNK while ACO-RNK performed 4.5% than MGACO.

For our next experiment we examined the variation of the average SLR as the number of nodes of the DAGs was increased. Fig. 5 shows as the number of nodes increases, the performance of the average SLR values, when compared to our proposed algorithm and that of the MGACO and HEFT, shows a steady increase. This indicates that our proposed algorithm, for large applications with more tasks when compared to smaller applications, performs better. ACO-RNK is better than MGACO by, on average, 4.8% and the HEFT by 14.5%.

The quality of schedules generated by the algorithms for varying communication to computation cost ratio (CCR) values, were observed in the third experiment. These results are presented in Fig. 6. For the lower CCR values, between 0.5 and 1.0, where graphs are indicative of being of high computational intensity, our proposed algorithm performed better than MGACO. The HEFT algorithm, on the other hand, yielded higher schedule lengths, as shown by the higher SLR values. However, as the CCR values increased indicating more communication intensive graphs, it was found that our proposed work experienced lower SLR values than both the MGACO and HEFT algorithms. Overall, our algorithm outperformed the other algorithms for graphs in all the CCR value categories. Our work performed better than the MGACO algorithm by 5.5% and 19.4% better than the HEFT algorithm.

In Fig. 7, we investigate the average speedup, as the DAG size increases. Our proposed algorithm experienced a steady increase in the average speedup, indicating an outperformance of both the HEFT and

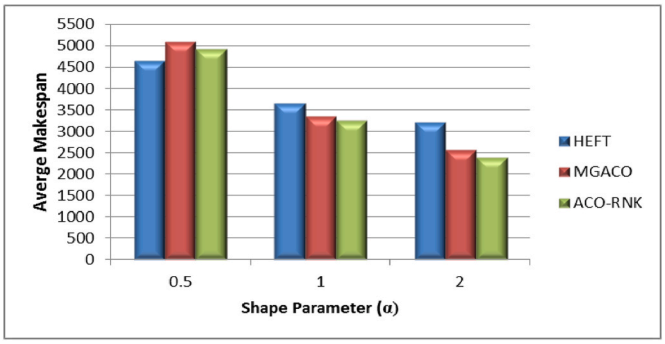


Fig. 4. Results for average makepan for varied dag structure.

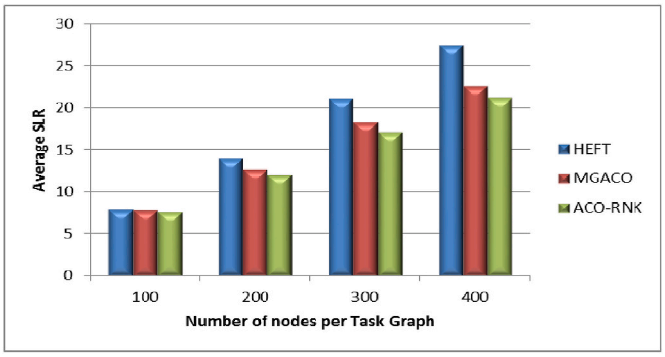


Fig. 5. Results for average slr for varying number of nodes.

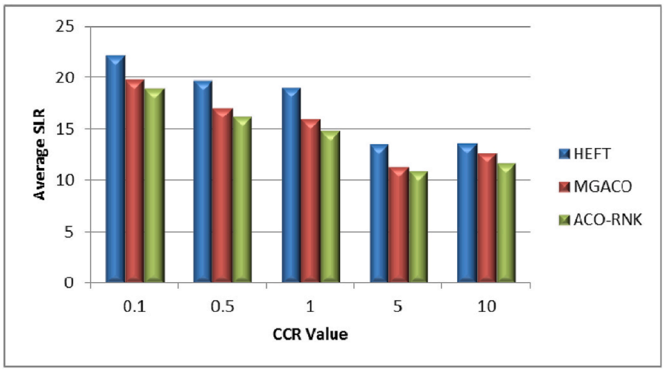


Fig. 6. Results for average slr for varying ccr values.

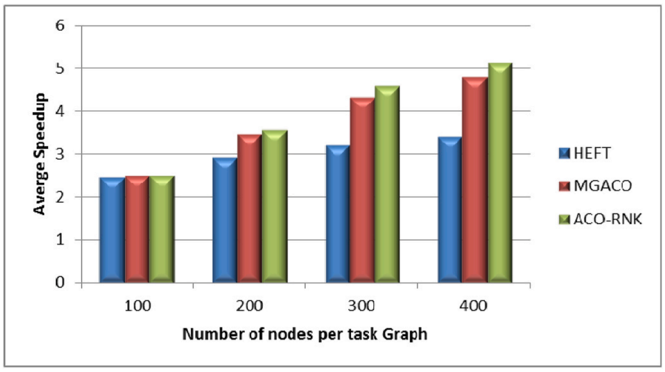


Fig. 7. Results for average speedup for varying number of nodes.

the MGACO algorithms. The HEFT experienced or showed minimal increase in the speedup throughout this experiment. The average speedup experienced by the MGACO was steady however it was not as pronounced as the ACO-RNK. This shows, as the number of nodes of the DAG was increased our proposed algorithm continue to yield the smallest schedule lengths. Overall, our proposed algorithm was better than HEFT and MGACO by 24 and 4.7% respectively.

6. Conclusions

Robust, versatile and efficient scheduling strategies are required and being continuously sought out, if we are to fully exploit the high-performance potential of heterogeneous multiprocessor computing environments. With this in mind, we proposed an ACO-based algorithm (ACO-RNK), which utilizes an upward ranked value of the nodes or tasks, an insertion-based policy; an anti-aging or stagnation policy that incorporates random searches, to further guide the ants toward quality solutions. We compared our proposed algorithm to the HEFT and the MGACO algorithms using a set of various randomly generated task graphs. Our experimental study showed that the ACO-RNK outperformed these algorithms yielding better results when it came to the average speedup and average SLR for varying DAG sizes as well as the varying DAG shapes. For future work, our plan is to investigate our proposed algorithm with a least weight or maximum weight, in our ranking strategy; incorporate a downward rank, and use a range of iterations as well a wider range of processors. We also intend to investigate and add both local optimization and stagnation strategies to further increase its efficiency as an algorithm for tackling the static task scheduling problem.

Credit author statement

All persons who meet authorship criteria are listed as authors, and all authors certify that they have participated sufficiently in the work to take public responsibility for the content including participation in the concept, design, analysis, writing or revision of the manuscript. Furthermore, each author, certifies that this material or similar material has not been and will not be submitted to or published in any other publication before its appearance in the **Array**.

Category 1:Jeffrey Elcock, Concept and design of study, Nekiesha Edward, Concept and design of study, Acquisition of data: Jeffrey Elcock, Acquisition of data, Nekiesha Edward, Analysis and interpretation of data: Jeffrey Elcock, Analysis and interpretation of dataNekiesha Edward, **Category 2**:Drafting the manuscript: Jeffrey Elcock, Drafting the manuscriptNekiesha Edward, Revising the manuscript critically: Jeffrey Elcock, Nekiesha Edward, **Category 3**:Approving the version of the manuscript to be published: Jeffrey Elcock, Nekiesha Edward, This statement is signed by all the authors:Jeffrey Elcock jelcock 12/01/2023, Nekiesha Edward Nedward 12/01/2023.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.array.2023.100280.

References

[1] Dogan A, Ozguner F. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Trans Parallel Distr Syst 2002;13(3):308–23.
[2] Chaudhuri P, Elcock J. Scheduling DAG-based applications in multicluster environments with background workload using task duplication. Int J Comput Math 2010;87(11):2387–97.
[3] Ijaz S, et al. Efficient scheduling strategy for task graphs in heterogeneous computing environment. Int Arab J Inf Technol 2013;10(5):486–92.
[4] Lin W-M, Gu Q. An efficient clustering-based task scheduling algorithm for parallel programs with task duplication. J Inf Sci Eng 2007;23(2):589–604.
[5] Manudhane KA, Wadhe A. Comparative study of static task scheduling algorithms for heterogeneous systems. Int J Comput Sci Eng 2013;5(3):166–73.
[6] Kashani M, Sarvizadeh R. A novel method for task scheduling in distributed systems using max-min ant colony optimization. In: 2011 3rd international conference on advanced computer control (ICACC). Harbin, China: IEEE; 2011. p. 422–6.

[7] Guoqi X, Renfa L, Keqin L. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messaging on networked embedded systems. Journal of Parallel and Distributing 2015;83:1–12.
[8] Topcuoglu H, Hariri S, Wu M-y. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distr Syst 2002;13 (3):260–74.
[9] Mihaela-Andreea V, et al. Resource-aware hybrid scheduling algorithm in heterogenous distributed computing. Future Generat Comput Syst 2015;51:61–71.
[10] Maurya AK, Tripathi AK. On benchmarking task scheduling algorithms for heterogeneous computing systems. J Supercomput 2018;74(7):3039–70.
[11] Qiao T, et al. A hybrid task scheduling algorithm based on task clustering. Mobile Network Appl 2020;25(4):1518–27.
[12] Papadimitriou CH, Yannakakis M. T Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms. J Syst Software 2017;133:1–16.
[13] Sakellariou R, Zhao H. A hybrid heuristic for dag scheduling on heterogeneous systems. In: 18th international parallel and distributed processing symposium, vol. 2004. NM, USA: IEEE; 2004.
[14] Mei J, Li K, Li K. A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems. The Journal of Supercomputing Systems 2014;68:1347–77.
[15] Ranaweera S, Agrawal DP. A task duplication based scheduling algorithm for heterogeneous systems. In: Parallel and distributed processing symposium, 2000. IPDPS 2000. Proceedings. 14th international. IEEE; 2000.
[16] Sinnen O, Andrea T, Manpreet K. Contention-aware scheduling with task duplication. J Parallel Distr Comput 2011;71(1):77–86.
[17] Gupta A, Garg A. Load balancing based task scheduling with ACO in cloud computing. In: 2017 international conference on computer and aplications (ICCA). Doha, Qatar: IEEE; 2017.
[18] Samal AK, et al. A novel fault-tolerant scheduling of real-time tasks on multiprocessor using discrete-elitist multi-ACO. In: International conference on communications and signal processing, (ICCSP). India: IEEE; 2015.
[19] Zhou Z, et al. An improved genetic algorithms using greedy strategy toward task scheduling optimization in cloud environments. Neural Comput Appl 2019;32: 1531–41.
[20] Al-maamari A, Fatma OE. Task scheduling using PSO algorithm in cloud computing environments. International Journal of Grid and Distributed Computing 2015;8(2): 245–56.
[21] Prongnuch S, et al. A heuristic approach for scheduling in heterogeneous distributed embedded systems. International Journal of Intelligent Engineering and Systems 2019;13(1):135–45.
[22] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by A colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 1996;26(1):29–41.
[23] Vassiliadis V, Dounias G. Nature–inspired intelligence: a review of selected methods and applications. Int J Artif Intell Tool 2009;18(4):487–516.
[24] Bank M, Honig U, Schiffmann W. An ACO-based approach for scheduling task graphs with communication costs. In: 2005 international conference on parallel processing (ICPP'05). Poland: IEEE; 2005.
[25] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1997;1(1):53–66.
[26] Jaiswal U, Aggarwal S. Ant colony optimization. Int J Sci Eng Res 2011;2(7):1–7.