Date of publication MAR-31, 2024, date of current version JUL-20, 2023.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.23.1.3436

SA-Based QoS Aware Workflow Scheduling of Collaborative Tasks in Grid Computing

MOHEB R. GIRGIS[1], TAREK M. MAHMOUD [2], HAGAR M. AZZAM[1]
[1] Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt
[2] Faculty of Computers and Artificial Intelligence, University Of Sadat City, Cairo, Egypt

Corresponding author: Moheb R. Girgis (e-mail: moheb.girgis@mu.edu.eg).

ABSTRACT Scheduling workflow tasks in grid computing is a complex process, especially if it is associated with satisfying the user's requirements to complete tasks within a specified time, with lowest possible cost. This paper presents a proposed Simulated Annealing (SA) based Grid Workflow Tasks Scheduling Approach (SA-GWTSA) that takes into account users' QoS (quality of service) constraints in terms of cost and time. For a given set of inter-dependent workflow tasks, it generates an optimal schedule, which minimizes the execution time and cost, such that the optimized time is within the time constraints (deadline) imposed by the user. In SA-GWTSA, the workflow tasks, which are modeled as a DAG, are divided into task divisions, each of which consists of a set of sequential tasks. Then, the optimal sub-schedules of all task divisions are computed applying SA algorithm, and used to obtain the execution schedule of the entire workflow. In the proposed algorithm, the sub-schedule of each branch division is represented by a vector, in which each element holds the ID of the service provider chosen from a list of service providers capable of executing the corresponding task in the branch. The algorithm uses a fitness function that is formulated as a multi-objective function of time and cost, which gives users the ability to determine their requirements of time against cost, by changing the weighting coefficients in the objective function. The paper also exhibits the experimental results of assessing the performance of SA-GWTSA with workflows samples of different sizes, compared to different scheduling algorithms: Greedy-Time, Greedy-Cost, and Modified Greedy-Cost.

KEYWORDS grid computing; workflow tasks scheduling; simulated annealing algorithm; quality of service constraints.

I. INTRODUCTION

GRID computing has emerged as an efficient approach to solve extensive problems in business, engineering and science. In Grid computing, several processing resources are integrated and connected to work together as one huge computing power to accomplish a common goal. These resources could be geographically distributed over the world, and they could have significantly different capabilities and specifications. To benefit from the grid computing capabilities, effectual scheduling algorithms are primarily essential. Grid scheduling is an activity that assigns and manages the execution of related tasks on distributed resources. The main challenge in grid scheduling is how to distribute collaborative tasks to the available resources, taking into account the quality of service (QoS), time and cost available to the user. For workflow processing systems, ***time*** denotes the overall time needed for completing the workflow execution; while ***cost*** denotes the cost linked to the workflows execution incorporating Grid resources usage charge for processing workflow tasks and the workflow systems management cost. The algorithms of scheduling collaborative tasks utilize DAGs (Directed Acyclic Graphs) to model tasks dependency.

Grid scheduling is an NP-complete problem as the computational Grid comprises resources that are heterogeneous and reside in different administrative regions, which employ distinctive management rules. SA (Simulated Annealing Algorithm) [1] is one of the efficient heuristic algorithms, which have been successfully applied to deal with NP-complete problems.

This paper presents a proposed SA-based Grid Workflow Tasks Scheduling Approach (SA-GWTSA) that takes into account users' QoS (quality of service) constraints in terms of cost and time. The input to SA-GWTSA is a set of workflow tasks, the dependencies between them, and the time limit (deadline) stated by the user for the execution of the workflow**.** The output of SA-GWTSA is an optimal schedule for all workflow tasks that minimizes the execution time and cost, such that the scheduled time is within the deadline imposed by the user. In this algorithm, a DAG is used to represent the dependency between the workflow tasks. The DAG is divided, and then the optimal sub-schedules of all task divisions are computed and used to obtain the execution schedule of the entire workflow. In SA-GWTSA, the SA technique is used to compute the optimal execution sub-schedule for each branch division that consists of a set of sequential tasks. In this technique, the sub-schedule of each branch division is represented by a vector, in which each element holds the ID of the service provider chosen to execute the corresponding task in the branch, and the fitness function is formulated as a multi-objective function of time and cost.

The next sections of this paper are as follows: The 2nd section presents related work; the 3rd section presents the problem description; the 4th section describes the proposed QoS-based grid workflow tasks scheduling algorithm, SA-GWTSA; the 5th section presents a case study to illustrate the working of SA-GWTSA; the 6th section exhibits the experimental results; and the 7th section presents the conclusion and future work.

II. RELATED WORK

Several research studies were proposed, in which heuristic and metaheuristic algorithms were used to address the problem of scheduling tasks in computational grids. This section gives a review of examples of such studies.

A. METAHEURISTICS-BASED APPROACHES

Aggarwal et al. [2] presented a scheduler based on a GA for computational grids. It minimizes make-span, the available resources idle time, and turn-around time, while satisfying the deadlines specified by users. Yu and Buyya [3] proposed a GA-based workflow scheduling approach with budget constraint. It aims to minimize execution time while satisfying a specified processing budget. Yu and Buyya [4] presented a GA-based workflow scheduling approach with two QoS constraints, deadline and budget. Zhang et al. [5] proposed an approach based on PSO (particle swarm optimization) for solving task scheduling problem in grid environment, which aims to generate an optimal schedule that minimizes the completion time of the tasks. Chen et al. [6] proposed a grid scheduling approach that combines a discrete PSO with the SA (simulated annealing) method, aimed at minimizing the grid cost, which comprises communication and computing costs. Kant et al. [7] proposed a framework for grid scheduling using dynamic information and an ant colony optimization (ACO) algorithm to minimize the maximal total tardiness time of dynamic job scheduling in grid computing, while optimizing the resource utilization. Bouali et al. [8] proposed a hybrid approach between the Heterogeneous Earliest Finish Time (HEFT) heuristic and PSO, to minimize the overall completion time of all tasks in the DAG. Jiang and Chen [9] presented TSGA genetic algorithm for task scheduling that divides the search space into random patterns to check out the search space to minimize the execution time. Gabaldon et al. [10] proposed a PSO-based approach for scheduling parallel jobs containing cooperating tasks aimed to minimize energy consumption. Gabaldon et al. [11] proposed a hybrid PSO-GA meta-heuristic approach for solving the resource matching and scheduling parallel tasks including collaborative ones in heterogeneous multi-cluster systems, which aimed to minimize both makespan and energy consumption. Younis and Yang [12] proposed two hybrid meta-heuristic schedulers. The first scheduler combines Ant Colony Optimization and Variable Neighborhood Search (VNS), while the second one merges the GA with VNS to minimize the makespan. Ghosh et al. [13] presented a hybrid GA-PSO algorithm for Grid job scheduling, which aimed to reduce the schedule makespan and flowtime. Chhabra et al. [14] proposed a multi-objective hybrid scheduling algorithm that combines Cuckoo Search and Firefly algorithm for scheduling offline workload of parallel jobs with collaborative tasks in High-Performance Computing Grid systems to optimize both energy-efficiency and QoS-aware performance expectations. Abdulal et al [15] presented a Mutation Based Simulated Annealing Algorithm (MSA), which uses simulated annealing selection, single change mutation, and a new random minimum completion time (Random-MCT). Also it maintains two solutions simultaneously.

B. HEURISTICS-BASED APPROACHES

Yu et al. [16] proposed an algorithm for QoS-based workflow scheduling, which minimizes the execution cost while satisfying the deadline. This algorithm utilizes an approach based on Markov Decision Process to schedule the execution of sequential workflow tasks. Benedict and Vasudevan [17] proposed a grid scheduling approach that uses Tabu Search method, for obtaining better computational Grid schedules, with two objectives: maximizing job completion ratio and minimizing the Grid scheduler overhead to choose the precise workflow sequence. Meddeber and Yagoubi [18] presented a dependent task allocation approach for Grids, which divides a given task graph into a set of linked components to decrease, if possible, the average execution time of submitted tasks, and to reduce communication costs, while respecting the dependency between tasks constraints. Bahnasawy et al. [19] presented an algorithm for scheduling distributed heterogeneous computing systems. The algorithm divides the given DAG into levels based on the precedency relationships, and sorts the tasks of each layer in descending order according to their computation sizes, then the tasks are selected from that layer in order. Bidgoli and Nezad [20] proposed a scheduling algorithm, GCDM, for grid computing to minimize the final cost of implementation tasks, taking into account the data transfer cost between different tasks and their inter-dependencies that are modeled as a DAG. Goel et al. [21] presented a scheduling algorithm that combines three scheduling algorithms: Shortest Job First, First Come First Serve, and Round Robin, considering the dependencies between tasks in grid environments and aiming to minimize the time required for executing all tasks. Hossam et al. [22] proposed the algorithm WS-GCDM (WorkStealing-Grid Cost Dependency Matrix), which is an enhancement of GCDM [20]. It balances task scheduling among the available grid resources, while GCDM utilizes certain number of grid resources irrespectively of the number of available resources. Rahman et al. [23] presented a dynamic and adaptive workflow scheduling algorithm based on critical path (CP) for grid computing, which dynamically and efficiently maps tasks of the workflow to grid resources via determining, at every step, the CP in the workflow graph. They, also, outlined a hybrid heuristic that merges the presented adaptive scheduling technique features with metaheuristics to obtain optimal execution time and cost while satisfying the users' requirements. Chauhan and Nitin [24] proposed a decentralized P2P algorithm for grid scheduling that schedules sub-tasks of DAG tasks, taking into account three factors: subtasks computation and communication costs, and the subtask waiting time caused by predecessors and precedence constraints. Garg and Singh [25] proposed an adaptive approach based on a rescheduling method for scheduling workflow dependent tasks on the dynamic grid resources. It initially performs static scheduling, followed by resource monitoring, and then rescheduling to minimize the execution time for workflow application.

The proposed QoS-based scheduling approach differs from the above mentioned approaches in the following points:

- The problem of scheduling workflow tasks on Grid is formulated as a problem of multi-objective optimization, where the execution time and cost are minimized, such that the optimized time is within the deadline imposed by the user.
- It employs a SA-based technique to compute the optimal execution sub-schedule for each set of sequential tasks, represented by a branch division in the workflow DAG.
- This technique uses a novel representation for the candidate solution (sub-schedule) of each branch division as a vector, in which each element holds the ID of the service provider, chosen from a list of service providers capable of executing the corresponding task in the branch; and the fitness function is formulated as a multi-objective function of time and cost.
- The optimal sub-schedules of all task divisions are used to obtain the execution schedule of the entire workflow.

III. PROBLEM DESCRIPTION

Workflow application tasks in grid computing system can be modeled as a DAG, which is represented with two sets (T, E), where T = {T_i, i = 1 . . . n} denotes a set of n tasks, while E denotes the set of directed edges between tasks, where an edge (T_i, T_k) represents the dependency of task T_k on task T_i, which means task T_i must be completed before scheduling task T_k. Task T_i is referred to as task T_k's parent and task T_k is referred to as task T_i's child. Assuming D is the user's provided deadline (time constraint) for the workflow execution, then the workflow application can be expressed as G (T, E, D). In the DAG, an *entry task* is a task that has no parent tasks and referred to as T_{entry}, and an *exit task* is a task that has no child tasks, and referred to as T_{exit}.

In a grid computing system, there is a set of service types, where diverse service providers can support each service type. Let m be the number of available services. Each task T_i has a set of services S_i^j ($1 \le i \le n$, $1 \le j \le m_i$, $m_i \le m$) that can execute this task, but only one of these services is chosen to execute the task. The processing capabilities of services vary and are provided at different prices. In general, there is an inverse proportion between the service price and the processing time [14]. The service price and time for executing task T_i on service S_i^j are denoted by c_i^j and t_i^j, respectively.

The scheduling problem is to assign every task T_i to a service S_i^j to minimize the execution time and cost, such that the execution of the workflow is completed within the user's provided deadline, while taking into account the task precedence constraints.

IV. THE PROPOSED QOS-BASED GRID WORKFLOW TASKS SCHEDULING APPROACH

In this work, the following steps are performed to solve the scheduling problem:

Step1: Detect available services, and choose available service providers for each task according to the QoS parameters of services specified by the user.

Step2: Cluster tasks of the workflow into task divisions.

Step3: Distribute the user's provided deadline (referred to as *specified deadline*) on task divisions.

Step4: Compute optimal sub-schedules for individual task divisions by using a SA-based strategy, and then use these sub-schedules to generate an optimal schedule for the entire workflow.

The following subsections provide detailed description of these steps.

A. SERVICE DETECTION AND QOS REQUEST

Providing details of QoS for every available service is important for efficient workflow tasks scheduling. As Figure 1 illustrates, the WMS (Workflow Management System) first sends a query to the GIS (Grid Information Service), which has knowledge of all registered Grid service providers, to detect services, which are suitable for processing every task of the workflow user. Each query specifies parameters of the task, estimated execution time, and workflow user. The GIS, in turn, replies with the available list of services for every task. Then, the WMS sends a QoS request to these services to obtain their processing price and time for providing the service with the required QoS level.

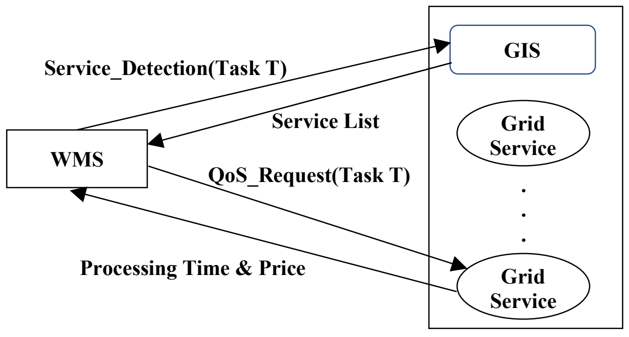


Figure 1. Service Detection and QoS Request

B. WORKFLOW TASK PARTITIONING

Workflow DAG dividing process starts by categorizing workflow tasks in G into a simple task or a synchronization task [14]. A synchronization task is a one that has many parent tasks and/or child tasks, while a simple task is a one which has at most one child and/or parent task. For example, in Figure 2(a), the 1st, 10th, and 14th tasks are synchronization tasks, while the remaining tasks are simple tasks. Then, the workflow tasks in G are divided into independent branches *B* and synchronization tasks Y, which leads to minimizing the size of G making it simpler and thus, containing less number of nodes. Let *P* be a set of nodes representing a set of task divisions P_i, $1 \le i \le nY+nB$, where nY and nB are the total numbers of synchronization tasks and workflow branches, respectively. Assume *E′* is the set of directed edges, where each edge takes the form (P_i, P_j) with P_i is a parent of P_j. Then, the divided graph can be described as *G′*(*P*, *E′*, *D*). Figure 2(b) shows the DAG of Figure 2(a) after dividing. For example, in this figure, the sequence of tasks T_2, T_3, and T_4 forms a branch division. A *simple path* in *G′* is a task division sequence that includes a directed edge from each task division in it to its successor, where the path task divisions are not repeated. The ***DAG_Dividing*** algorithm is shown in Figure 3.

Each task division P_i has 4 attributes: deadline ($dl[P_i]$), expected execution time (*eet*[P_i]), start time (start_time[P_i]), and minimum execution time (*met*[P_i]). If P_i is a branch, then its earliest start time is the earliest start time of the 1st task in it, and is calculated according to the deadlines of its parent divisions as follows:

$$start\_time[P_i] = \max_{P_j \in PP_i} dl[P_j], \quad (1)$$

where PP_i is the set of P_i's parent task divisions. The P_i's minimum execution time is calculated as follows:

$$met[P_i] = \sum_{T_x \in P_i} \min_{1 \le y \le m_x} t_x^y \,. \quad (2)$$

Expected execution time of P_i is calculated as follows:

$$eet[P_i] = dl[P_i] - start\_time[P_i]. \quad (3)$$

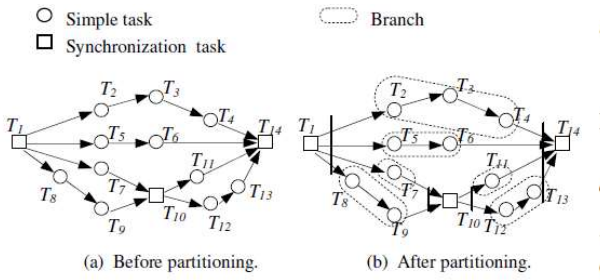


Figure 2. An example workflow DAG partitioning [7]

```
Algorithm 1: DAG_Dividing
Input: Original task graph G
Output: Divided task graph G'
Begin
    1.  Initialize:  Divisions counter DivsNo=0,
                     List of divisions  DivisionList = []
    2.  For each task t in G
            // Determine the task type, either synchronization (Y)
            or simple task (T)
    3.      If (no. of parent tasks of t = 1 && no. of child tasks
            of t = 1) ||
               (no. of parent tasks of t = 1 && no. of child tasks
               of t = 0)  ||
               (no. of parent tasks of t = 0 && no. of child tasks
               of t =1)  then
    4.              task t is T
    5.      Else
    6.              task t is Y
    7.      End If
    8.      If (t  is Y) then
    9.              DivsNo++;
    10.             DivisionList[DivsNo].add(t)
    11.     Else If (t is T) then
    12.             If (parent task A of t is Y) then
    13.                     DivsNo++;
    14.                     DivisionList[DivsNo].add(t)
    15.             Else if (parent task A of t is T) then
    16.                     Index = index of division that includes
                            A;
    17.                     DivisionList[Index].add(t)
    18.             End If
    19.     End If
    20. End For
End
```

Figure 3. DAG Dividing Algorithm

C. DEADLINE ASSIGNMENT

Having divided the workflow graph G, the specified deadline, *D*, is distributed on the G' task divisions, such that the deadline $dl[P_i]$ allocated to each task division P_i is a *sub-deadline* of *D*.

Following are the deadline distribution rules [14]:

R1: The total sub-deadline of any path from a synchronization task Y_i to another synchronization task Y_j must be the same.

R2: Any path from P_i to P_j, where $T_{entry} \in P_i$ and $T_{exit} \in P_j$, has a total sub-deadline, which is equal to D.

R3: A sub-deadline allocated to any task division P_i must be greater than or equal to ***met***(P_i) .

R4: The specified deadline, *D*, is distributed over task divisions in proportion to their ***met***.

These deadline distribution rules are implemented on the task division graph by using *BFS (Breadth-First Search)* algorithm and *DFS (Depth-First Search)* algorithm to calculate, for each task division, *start time* and *sub-deadline*. The deadline distribution algorithm is shown in Figure 4.

```
Algorithm 2: Deadline_Distributing
Input:   Divided task graph G', and Overall Deadline D
Output: Deadline of each division Pi, dl[Pi]
Begin
    1.  Get all start nodes    //breadth traversing
    2.  Get all possible paths PTH from each start node to exit
        node  //depth traversing
    3.  Get met for each division $P_j$ in G', using Eq. (2)
    4.  For each path $pth_i$ in PTH
    5.      dl($pth_i$) = D;   // Apply R2
    6.      met($pth_i$) = $\sum_{P_j \in pth_i} met(P_j)$;
    7.      For each division $P_j$ in path $pth_i$
    8.          dl($P_j$) = (met($P_j$) / met($pth_i$)) * dl($pth_i$);  //
                Apply Rule R4
    9.      End For
    10. End For
End
```

Figure 4. Deadline Distribution Algorithm

D. GENERATION OF AN OPTIMAL SCHEDULE

Once the sub-deadline of a task division is determined, an optimal sub-schedule for this task division can be obtained. If the obtained optimal sub-schedules for all task divisions ensure that the execution of these task divisions can be completed within their sub-deadlines, the entire workflow execution will be finished within the specified deadline. Also, minimizing the costs for all task divisions leads to reaching an optimal cost for the whole workflow. Thus, by combining all optimal sub-schedules, an optimized workflow schedule can be easily obtained. The scheduling solutions for the two task division types: *synchronization task* and *branch division,* as well as the overall grid workflow tasks scheduling algorithm (SA-GWTSA), are described below.

D.1. Scheduling Synchronization Task

The synchronization task scheduling (STS) is a single task scheduling problem. The optimal solution for such a problem can be simply obtained by selecting the service with the lowest cost which is able to execute the synchronization task within its allocated sub-deadline. Thus, for scheduling a synchronization task Y_i, the objective function is as follows:

$$\min c_i^k \text{, where } 1 \leq k \leq m_i \text{, and } t_i^k \leq eet(Y_i) \quad (4)$$

D.2. Branch Division Scheduling

If a branch division contains only one simple task, the solution for the branch division scheduling (BDS) is the same as STS. But, if a branch contains multiple tasks, the SA algorithm is used to get an optimal solution according to the evaluation of an objective function. Here, the optimal solution is to minimize the branch execution time and cost, with the condition that the optimized time is within its allocated sub-deadline. Thus, the objective function to be minimized to obtain an optimal sub-schedule for branch B_j can be represented as a weighted sum that combines the following two objectives:

$$\min cost(B_j) \text{ and } \min time(B_j) \text{, such that } time(B_j) \leq eet(B_j),$$

where

$$cost(B_j) = \sum_{T_i \in B_j} c_i^k, \quad (5)$$

$$time(B_j) = \sum_{T_i \in B_j} t_i^k, \quad (6)$$

and $1 \leq k \leq m_i$.

That is the objective function is formulated as follows:

$$F(B_j) = w_1\, cost(B_j) + w_2\, time(B_j), \quad (7)$$

where w_1 and w_2 are weighting coefficients, which satisfy the condition $w_1 + w_2 = 1$. This objective function F will be used by the proposed SA as a fitness function to evaluate the candidate sub-schedules.

E. THE PROPOSED SA-BASED BRANCH DIVISION TASK SCHEDULING (BDSSA) ALGORITHM

SA developed by Metropolis et al. [1] is a powerful optimization algorithm that can be used for task scheduling in the grid. By defining an appropriate objective/fitness function, neighborhood function, initial temperature and annealing schedule, we can efficiently allocate tasks to grid resources while minimizing the branch execution time and cost, with the condition that the optimized time is within its allocated sub-deadline.

The basic idea behind SA is to start with an initial solution, and iteratively improve it by making small changes to it. At each iteration, the algorithm evaluates the new solution and decides whether to accept or reject it based on a probability function. The probability function is designed to allow the algorithm to escape local optima and explore the solution space.

The main steps of the SA algorithm are as follows:

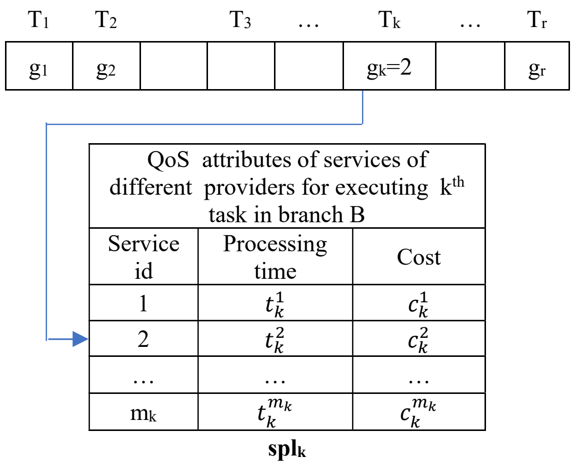
1. Initialize the temperature and the current solution.
2. While the temperature is above a minimum threshold:
   - Generate a new solution by applying the neighborhood function to the current solution.
   - Evaluate the new solution using the objective function.
   - Calculate the change in the objective function between the current solution and the new solution.
   - If the change in the objective function is negative, accept the new solution.
   - If the change in the objective function is positive, accept the new solution with a probability determined by the current temperature and the change in the objective function.
   - Update (decrease) the temperature.
3. Return the best solution found.

Solving the BDS problem by using SA requires the determination of the solution (sub-schedule) representation, the annealing schedule, the neighborhood function to generate a new solution from the current solution, and a suitable objective function. The proposed SA's components are presented below.

E.1. BDS Problem Representation and Initial Solution

In the proposed SA-based branch division scheduling (BDSSA) algorithm, for each branch in G', we build a data structure that represents a possible solution to a branch division scheduling on available service providers that satisfy the QoS constraints and specified deadline.

Each branch consists of a number of tasks, and each task has its own service providers. So, the proposed branch data structure representation is a vector, called bds_vector, consisting of a number of elements corresponding to the branch tasks, and each task T_i in the branch is accompanied by a list of service providers, spl_i, capable of executing this task. An element g_i that corresponds to a task T_i in the bds_vector holds the ID of a service provider, chosen from spl_i, to execute this task. Figure 5 shows an encoding for a branch B that consists of r tasks, $T_1, T_2, \ldots, T_k, \ldots, T_r$, where $g_k \in [1, m_k]$, and m_k is the number of services capable of executing task T_k.



T_1	T_2	T_3	…		T_k	…	T_r
g_1	g_2				$g_k=2$		g_r

QoS attributes of services of different providers for executing kth task in branch B		
Service id	Processing time	Cost
1	t_k^1	c_k^1
2	t_k^2	c_k^2
…	…	…
m_k	$t_k^{m_k}$	$c_k^{m_k}$

Figure 5. bds_vector representation of branch B

For example, the following bds_vector represents the branch that consists of tasks T_2, T_3, and T_4, in Figure 1, and indicates that these tasks will be executed on services with ids 1, 3, and 2, respectively.

T_2	**T_3**	**T_4**
1	**3**	**2**

Each vector for each branch consists of a random set of providers that are capable of executing each task in the branch. The bds_vector representing branch B must satisfy the condition:

$$\sum_{T_i \in B} t_i^k \leq eet(B) \quad (8)$$

According to the fitness evaluation for a bds_vector, a new neighborhood solution is generated, then the neighborhood's fitness is evaluated, and the algorithm decides whether to accept it or not. This process is repeated until the specified minimum threshold of the SA temperature is reached. The best schedule for each branch division in G' is kept. Then, these best schedules, with the best schedules of synchronization tasks, are used by the proposed SA-GWTSA to obtain the best schedule for the whole DAG (workflow tasks).

E.2. The Fitness Function

Based on the considered optimization objective, a fitness function is utilized to assess the quality of the solution (bds_vector). The scheduling goal here is to optimize the grid system performance in terms of cost and time for each division, as explained above. Therefore, in BDSSA, the fitness function is the multi-objective function defined by Eq. (7).

E.3. The Initial Temperature and Annealing Schedule

The initial temperature for the search is provided as a parameter and gradually decreases with the progress of the search. The annealing schedule is used to control the probability of accepting a worse solution, as it is implemented as a function of the current temperature.

E.4. The Neighborhood Function

This function is used to generate a new solution by making small changes to the current solution. In BDSSA, a mutation operator is used as a neighborhood function. Mutation is performed by randomly choosing an element in the bds_vector with certain mutation rate (M_r), then replacing the id value in it with another id value from the remaining providers' ids that can execute the corresponding task. An example illustrating the mutation operation is shown in Fig. 6.

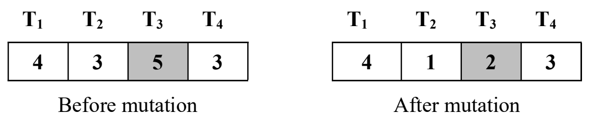


Figure 6. An example illustrating the mutation operation

```
Algorithm 3: BDSSA,  A SA-based branch division scheduling
algorithm
Input:  Branch B {$T_1, T_2, …, T_r$};
          Service Providers Lists SPL {$spl_1, spl_2, …, spl_r$}, where
          $spl_i$ is the list of service providers of  task $T_i$ in B;
          dl(B) (Deadline of B);
          $M_r$ (Mutation Rate);
          Maximum no. of  iterations Max_N;
          Initial temperature Init_Temper>0;
          The cooling rate CR;
          Weights ($w_1$ and $w_2$) of the fitness function;
Output: Best Schedule for branch B
Begin
  1.    Generate an initial solution $S_c$ at random from $spl_i$ of
        each task $T_i$ in B;
  2.    Evaluate $S_c$ (calculate $F(S_c)$ using Eq. (7))
  3.    Apply SA($S_c$)
  4.    Begin
  5.      Temper= Init_Temper
  6.      Repeat
  7.        For n = 1 To Max_N Do
  8.          Generate a new valid solution $S_n$, a random
              neighbor of $S_c$, using mutation operator;
  9.          Calculate $F(S_n)$, using Eq. (7);
              // Compare the change in objective function
  10.         Set $\Delta F = F(S_n) - F(S_c)$
              // if the new solution is better, accept it
  11.         If $\Delta F \le 0$ Then
  12.           $S_c \leftarrow S_n$    // $S_n$ replaces $S_c$
              // if the new solution is worse, accept it with a
              // probability
  13.         Else if random(0,1) < $e^{-\Delta F/Temper}$ Then
  14.           $S_c \leftarrow S_n$
  15.       End For
  16.       Temper = Temper × CR; // decrement temperature
  17.     Until stopping criterion is true;
  18.   End
  19.   Return the Best Schedule;
End
```

Figure 7. The proposed SA-based branch task division scheduling algorithm (BDSSA)

E.5. Overall BDSSA Algorithm

The BDSSA algorithm is given in Figure 7. The input to BDSSA is presented by a branch B, the list of service providers spl_i of each task T_i in B (see Figure 5), the dl(B) (deadline of branch B), initial temperature Init_Temper, the cooling rate CR, the max number of iterations Max_N, M_r (mutation rate), and the weights w_1 and w_2 of the fitness function. In steps 1-2, BDSSA generates an initial solution (bds_vector), S_c, for branch B, where the elements in S_c are populated by the service providers' IDs randomly selected from the list of service providers of the corresponding tasks, and evaluates its fitness using Eq. (7). The fitness evaluation is done by calculating the schedule (time and cost) for S_c, according to providers' IDs placed in its elements, using the procedure ***ComputeBranchSchedule()***, shown in Figure 8, then the obtained time and cost are substituted in Eq. (7). Next, steps 3-18 include the main steps of the SA algorithm. In step 5, the current temperature *Temper* is set to the initial temperature *Init_Temper*. Steps 6-17 include the outer loop of the SA algorithm, which repeatedly decreases the temperature by the cooling rate CR, until the stopping criterion is reached. For each temperature, an inner loop (Steps 7-15) is executed *Max_N* iterations. In each iteration, a neighboring solution S_n is generated by applying the mutation operator. S_n is accepted as the new current solution, if the difference $\Delta F = F(S_n) - F(S_c)$ is greater than zero, i.e., the new solution is better. If $\Delta F \le 0$, i.e., the new solution is worse, then accept it with a probability, which is a function of Tempr, $e^{-\Delta F/Temper}$. This probabilistic acceptance is achieved by generating a random number in [0, 1), and if it is less than $e^{-\Delta F/Temper}$, then replace the current solution by the new one. Finally, the best bds_vector (best schedule for branch B) is returned in step 19.

E.6. Decoding

After obtaining the best bds_vector, S_c, which represents the best schedule for the given branch B, it is decoded in order to set the start and end time for each task composing B. This decoding process is performed by applying the procedure ***ComputeBranchSchedule()***, shown in Figure 8. It uses the provider's ID placed in each element in S_c to get the time and price set by this provider for the corresponding task. Next, the procedure calculates the best execution cost and time for the whole branch by summing prices and times of all tasks in B, using Eq. (5) and Eq. (6), respectively.

```
Procedure ComputeBranchSchedule(S_c, SPL)
// Compute time schedule and cost of tasks in S_c according to
// providers ids placed in its elements using Eqs. (5) and (6)
Begin
   1.  time ← 0;
   2.  cost ← 0;
   3.  For element g_k in bds_vector S_c
   4.      Get service provider id stored in g_k (say s)
   5.      Get processing time $t_k^s$ for task T_k from its spl_k
   6.      Get processing cost $c_k^s$ for task T_k from its spl_k
   7.      time ← time + $t_k^s$
   8.      cost ← cost + $c_k^s$
   9.  End For
   10. Return (time, cost)
End
```

Figure 8. The proposed procedure for computing a branch schedule

F. OVERALL SA-BASED GRID WORKFLOW TASKS SCHEDULING ALGORITHM (SA-GWTSA)

This section describes the proposed ***SA-based Grid Workflow Tasks scheduling Algorithm*** (SA-GWTSA). It schedules the tasks of a workflow on Grid services based on users' QoS constraints. Figure 9 shows the flowchart of the proposed SA-GWTSA and Figure 10 shows its procedural details.

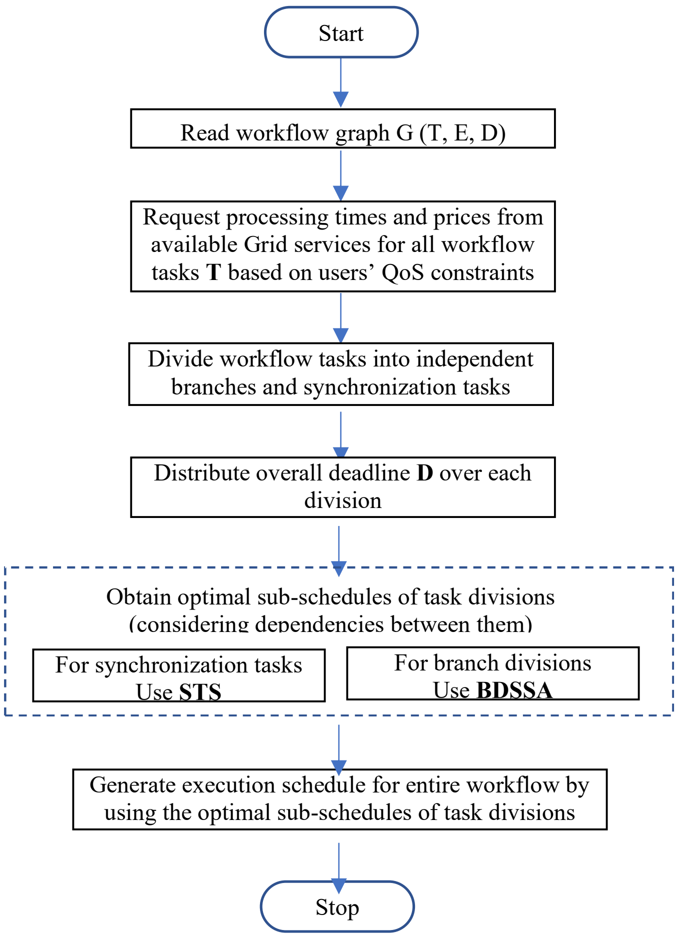

Start
Read workflow graph G (T, E, D)
Request processing times and prices from available Grid services for all workflow tasks **T** based on users' QoS constraints
Divide workflow tasks into independent branches and synchronization tasks
Distribute overall deadline **D** over each division
Obtain optimal sub-schedules of task divisions (considering dependencies between them)
For synchronization tasks Use **STS**
For branch divisions Use **BDSSA**
Generate execution schedule for entire workflow by using the optimal sub-schedules of task divisions
Stop

Figure 9. Flowchart of the proposed SA-GWTSA

As shown in Figure 10, the input to SA-GWTSA is the workflow graph G (T, E, D), where T = {T_i, i = 1 . . . n} denotes the workflow tasks set, E denotes the set of edges connecting tasks, and *D* denotes the user's provided deadline for the execution of workflow. The output of SA-GWTSA is an optimal schedule for all workflow tasks, which minimizes both time and cost of workflow execution, such that the optimized time is within the specified deadline. The algorithm starts by requesting processing times and prices from available Grid services for all workflow tasks. Then, it divides workflow tasks into independent branches (sequences of simple tasks) and synchronization tasks, by using the *DAG_Dividing* algorithm, shown in Figure 3, generating a reduced graph *G*'(*P*, *E*', *D*), where *P* denotes the divisions (branches and synchronization tasks) set, and *E'* denotes the set of directed edges between divisions in G'. Then, it uses the deadline assignment algorithm, *Deadline_Distributing*, shown in Figure 4, to distribute the overall deadline *D* over each division. Finally, SA-GWTSA generates the execution schedule for the entire workflow by using the optimal sub-schedules of task divisions. If a task division is a branch, its optimal schedule is obtained using BDSSA, as described in Sec. E., otherwise STS is used, as described in Sec. D.1. If a task division has one or more child divisions, then the procedure *HandleChildDivision()* is called to compute their schedules.

Algorithm 4: SA-GWTSA, a SA-based grid workflow tasks scheduling algorithm

```
Input : G(T, E, D) (a workflow graph)
Output: An optimal workflow schedule
Begin
   1.  From available services, get processing price and time
       $\forall Ti \in T$
   2.  Call DAG_Dividing;          // Transform G into
       reduced graph G'(P, E', D)
   3.  Call Deadline_Distributing;   // Distribute deadline
       $D \ \forall P_i \in P$
   4.  Queue Q ← []
   5.  For each $P_i \in P$ do
   6.      scheduled[P_i] ← false;
   7.  End For
   8.  EP ← all entry divisions (divisions that has no parents)
   9.  For each P_i ∈ EP do
   10.     If P_i is a synchronization task Then
   11.        Obtain an optimal sub-schedule for P_i via STS
   12.     Else // P_i is a branch division
   13.        Obtain an optimal sub-schedule for P_i via BDSSA
   14.     End IF
   15.     scheduled[P_i] ← true;
   16.     HandleChildDivision(P_i);
   17. End For
   18. While Q is not empty do
   19.     B ← remove top element of Q;
   20.     PB ← all parent divisions of B;
   21.     If (scheduled[$P_j$] = true $\forall P_j \in$ PB) Then
   22.        start_time[B] ← $\max_{P_j \in PB}$ dl[P_j];
   23.        scheduled[B] ← true;
   24.     Else
   25.        If B is not on top of Q Then add B to Q;
   26.        If B has child divisions Then
   27.           HandleChildDivision(B);
   28.        End If
   29.     End If
   30. End While
End

Procedure HandleChildDivision(B)
```

```
Begin
    1.  dl[B] ← expected completion time of B;
    2.  If B has child divisions Then CP ← child task divisions
        of B;
    3.  For each $P_j$ ∈ CP do
    4.      If $P_j$ is a branch division Then
    5.          start_time[$P_j$] ← dl[B];
    6.          Obtain an optimal sub-schedule for $P_j$ via
                BDSSA;
    7.          scheduled[$P_j$] ← true;
    8.          $P_k$ ← child task division of $P_j$;
    9.          If $P_k$ is not on top of Q Then add $P_k$ to Q;
    10.     Else
    11.         If $P_j$ is not on top of Q Then add $P_j$ to Q;
    12.     End If
    13. End For
End
```

Figure 10. The proposed SA-GWTSA

V. CASE STUDY

To illustrate the working of the proposed Grid scheduling algorithm, SA-GWTSA, it has been applied to the workflow modeled by the DAG given in Figure 2 (adapted from [13]), which consists of 14 tasks. So, 14 service types have been simulated, with a number of diverse service providers supporting each service type. Table 1 shows, for each task, the QoS attributes of providers that will provide the same service type needed for processing this task. These attributes are: provider id ID, processing time (sec), and cost ($). The required deadline (DL) is assumed to be 350 sec.

The input to SA-GWTSA includes:

- the specified deadline D;
- SA parameters: Max_N, Mr, Init_Temper, CR, w1, and w2;
- a file containing the edges of the DAG of the example workflow: 1-2, 1-5, 1-7, 1-8, 2-3, 3-4, 4-14, 5-6, 6-14, 7-10, 10-11, 11-14, 10-12, 12-13, 13-14, 8-9, 9-10; and
- a file containing the service providers' information shown in Table 1.

The output produced by SA-GWTSA consists of:

- a file containing the divisions of the example workflow DAG, with their types, as shown in Table 2;
- a file containing the scheduled start and end times for each task with its service provider id, as shown in Table 3. Note that each provider id is prefixed with the corresponding task number to differentiate between providers of different tasks that have same id, for example, 3:1 refers to provider with id 1 of task 3, and 5:1 refers to provider with id 1 of task 5;
- a file containing the scheduled start and end times for each division, as shown in Table 4;
- resultant best schedule time: **150 sec** and best cost: **$ 221**.

Table 1. QoS attributes (provider id, processing time in sec, and cost in $) of services of different providers for executing the tasks of the example workflow

Task	QoS Attributes					
T1	PID	0	1	2		
	Time	20	10	30		
	Cost	20	30	10		
T2	PID	0	1	2	3	
	Time	40	25	10	5	
	Cost	10	25	40	45	
T3	PID	0	1	2		
	Time	25	10	15		
	Cost	10	20	16		
T4	PID	0	1			
	Time	6	18			
	Cost	15	5			
T5	PID	0	1	2		
	Time	8	4	19		
	Cost	10	15	5		
T6	PID	0	1	2		
	Time	20	10	30		
	Cost	20	30	10		
T7	PID	0	1	2	3	
	Time	7	10	15	35	
	Cost	25	20	15	7	
T8	PID	0	1	2	3	4
	Time	10	20	30	40	25
	Cost	60	40	20	10	35
T9	PID	0	1			
	Time	6	18			
	Cost	15	5			
T10	PID	0	1	2		
	Time	20	10	30		
	Cost	20	30	10		
T11	PID	0	1	2	3	
	Time	40	25	10	30	
	Cost	10	25	40	20	
T12	PID	0	1	2		
	Time	8	4	19		
	Cost	10	15	5		
T13	PID	0	1	2	3	
	Time	10	5	15	17	
	Cost	10	15	5	3	
T14	PID	0	1	2		
	Time	40	25	10		
	Cost	10	25	40		

Table 2. The divisions of the example workflow DAG, with their types (Y: Synchronization, B: Branch)

Partition	Type	Tasks
P1	Y	T1
P2	B	T2, T3, T4
P3	B	T5, T6
P4	B	T7
P5	B	T8, T9
P6	Y	T10
P7	B	T11
P8	B	T12, T13
P9	Y	T14

VI. EXPERIMENTAL RESULTS

Experiments have been carried out to assess SA-GWTSA performance. Three workflows of 11, 14, and 25 tasks have been used. For each task in each workflow, a different service type with diverse service providers has been simulated.

Table 3. The scheduled start and end times for each task with its service provider ID

Task	Start	End	PID
T1	0	30	1:2
T2	30	70	2:0
T3	70	95	3:0
T4	95	101	4:0
T5	30	38	5:0
T6	38	68	6:2
T7	30	65	7:3
T8	30	40	8:0
T9	40	85	9:1
T10	65	95	10:2
T11	95	125	11:3
T12	95	103	12:0
T13	103	113	13:0
T14	125	150	14:1

Table 4. The scheduled start and end times for each division

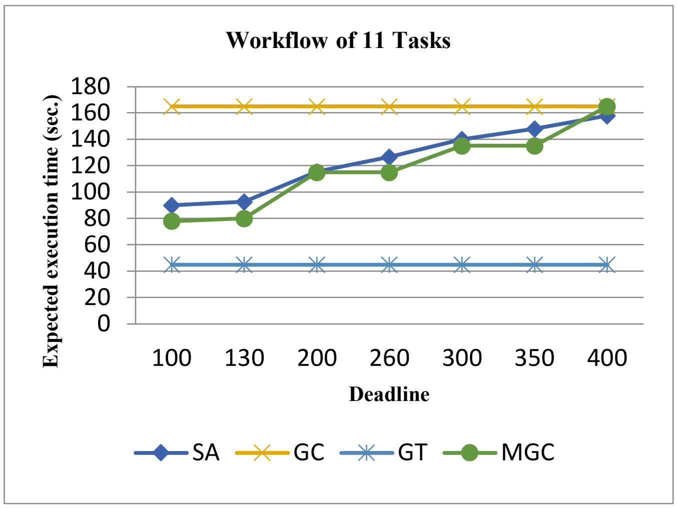
Partition	Start	End
P1	0	30
P2	30	101
P3	30	68
P4	30	65
P5	30	58
P6	65	95
P7	95	125
P8	95	113
P9	125	150

SA-GWTSA has been applied to the three workflows, with different deadlines, and the results are compared with 3 other scheduling algorithms, namely, Greedy-Time (GT) [13], Greedy-Cost (GC) [13], and Modified Greedy-Cost (MGC). For processing each task, GC chooses the lowest-cost service, whereas GT chooses the quickest service. MGC searches for the lowest-cost service for processing each task within the required deadline. The evaluation criteria were the execution cost and time constraint. The first criterion shows the workflow tasks scheduling costs on the utilized service Grid, while the second one shows whether the scheduling algorithm has generated a schedule that satisfies the specified deadline. For each deadline value, SA-GWTSA was run ten times and the average of the best time and cost values generated were calculated. The SA parameters used were: Max_N = 20, Mr = 0.05, CR=0.5, Init_Temper =0.9, and $w_1 = w_2 = 0.5$. The algorithms were implemented using C#, and run on TOSHIBA-Lap Intel(R) Core™ i5-2430M CPU, 2.4 GHz, 4 GB RAM.

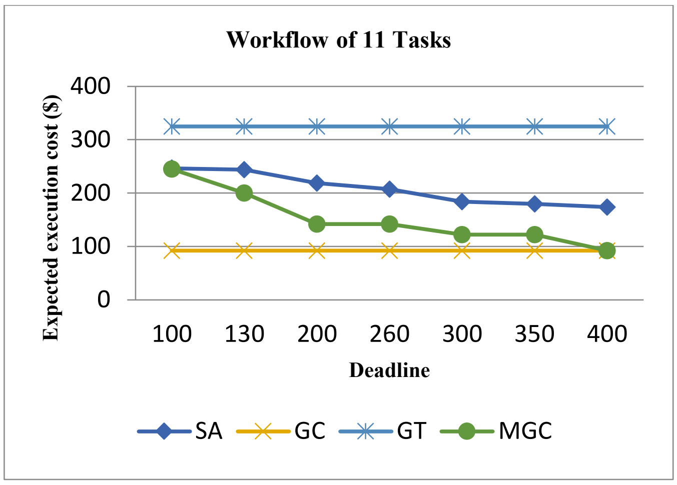
Figures 12, 13 and 14 show comparisons of the results of applying the 4 scheduling algorithms for the 3 workflows in terms of the execution time and costs. Figures 12(a), 13(a) and 14(a) show that the expected execution time for the 3 workflows using SA-GWTSA and MGC increases as the deadline is relaxed. For the 3 workflows, the expected execution time using SA-GWTSA is less than MGC with most deadlines. The workflows execution time using the GC algorithm is higher and cannot meet the required deadline when it is low. The GT provides lower execution time than the other three algorithms.

As shown in Figures 11(b), 12(b) and 13(b), for the 3 workflows, the execution cost using the GT algorithm is higher, but when using SA-GWTSA and MGC, it is reduced as the deadline is relaxed. For the 3 workflows, the execution cost using MGC is lower than SA-GWTSA. The GC provides lower execution cost than the other three algorithms.

As can be seen from these results, SA-GWTSA tries to optimize both the execution time and cost, the MGC algorithm tries to minimize the cost while keeping the execution time within the required deadline, whereas the GT and GC algorithms try to minimize the execution time and cost, respectively.

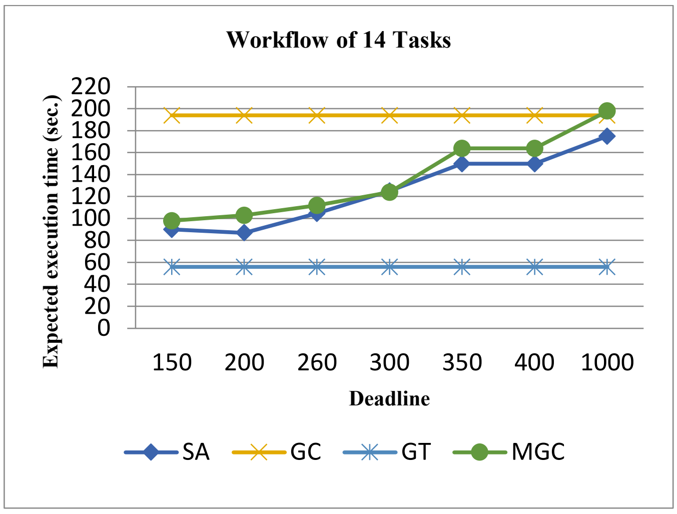


(a)

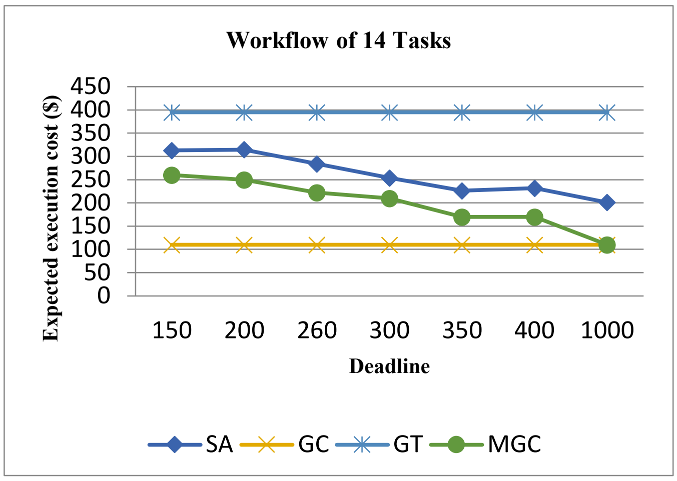


(b)

Figure 11. Expected execution time (a) and cost (b) for the workflow of 11 tasks using the four scheduling algorithms

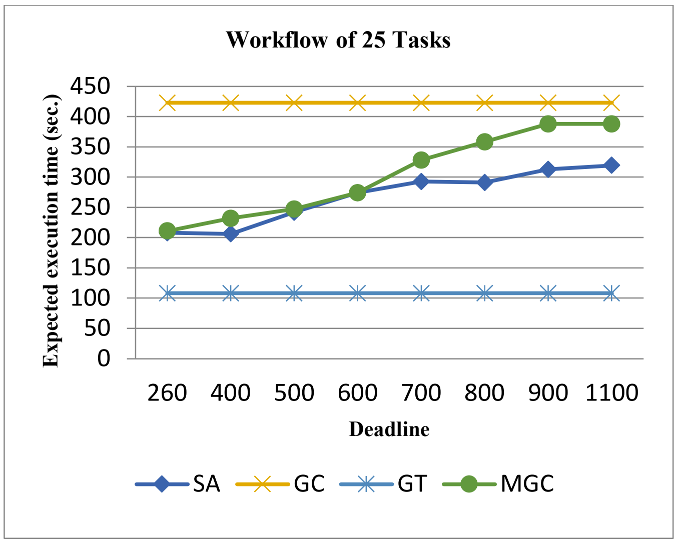


(a)

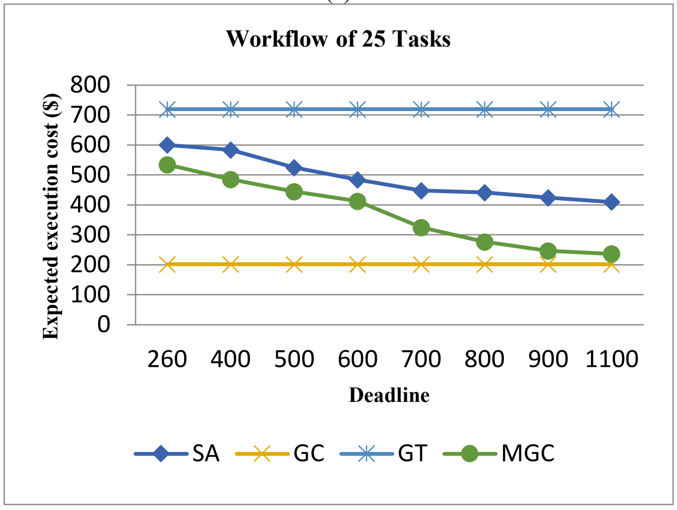


(b)

Figure 12. Expected execution time (a) and cost (b) for the workflow of 14 tasks using the four scheduling algorithms



(a)



(b)

Figure 13. Expected execution time (a) and cost (b) for the workflow of 25 tasks using the four scheduling algorithms

VII. CONCLUSION

This paper presents a proposed SA-based approach, SA-GWTSA for scheduling workflow tasks on Grid services based on users' QoS constraints in terms of time and cost. For a given collaborative set of workflow tasks, SA-GWTSA generates an optimal schedule, which minimizes the execution time and cost, such that the optimized time is within the deadline imposed by the user. In this approach, a DAG is used to represent the dependency between the workflow tasks. The DAG is divided, and then the optimal sub-schedules of task divisions are computed and utilized to generate the schedule for executing the entire workflow. SA is employed in SA-GWTSA to compute the optimal execution sub-schedule for each branch division, which consists of a set of sequential tasks.

Experiments have been carried out to assess SA-GWTSA's performance. The results are compared with three other scheduling algorithms: GC, GT and MGC. The results indicate that SA-GWTSA tries to optimize both the execution time and cost; the MGC algorithm tries to minimize the cost while keeping the execution time within the required deadline; whereas the GT and GC algorithms try to minimize the execution time and cost, respectively.

In the future work, we intend to modify the proposed workflow scheduler SA-GWTSA to consider resource dynamics, such that the schedule is adapted and updated during scheduling according to these dynamics. We also intend to augment BDSSA with a metaheuristic algorithm, such as GA, in order to improve the optimal execution schedule it produces for each branch division.

References

[1] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, issue 6, pp. 1087–1092, 1953.

[2] M. Aggarwal, R. D. Kent and A. Ngom, "Genetic algorithm based scheduler for computational grids," *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, Guelph, ON, Canada, 15-18 May 2005, pp. 209-215.

[3] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, Paris, France, 19-23 June 2006, pp. 1-10.

[4] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, pp. 217–230, 2006.

[5] L. Zhang, Y. Chen, R. Sun, S. Jing and B. Yang, "A task scheduling algorithm based on PSO for grid computing," *International Journal of Computational Intelligence Research*, vol. 4, issue 1, pp. 37–43, 2008.

[6] R. Chen, D. Shiau, S. H. Andlo, "Combined discrete particle swarm optimization and simulated annealing for grid computing scheduling problem," *Lecture Notes in Computer Science*, vol. 57, Springer, Berlin, pp. 242–251, 2009.

[7] A. Kant, A. Sharma, S. Agarwal, and S. Chandra, "An ACO approach to job scheduling in grid environment," In: B. K. Panigrahi, S. Das, P. N. Suganthan and S. S. Dash (eds), *Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, Lecture Notes in Computer Science*, vol. 6466, Springer, Berlin, Heidelberg, 2010.

[8] Y. Jiang, M. Chen, "Task scheduling for grid computing systems using a genetic algorithm," *The Journal of Supercomputing*, vol. 71, issue 4, pp. 1357–1377, 2015.

[9] L. Bouali, K. Oukfif, S. Bouzefrane, F. B. Oulebsir, "A hybrid algorithm for DAG application scheduling on computational grids," *International Conference on Mobile, Secure and Programmable Networking (MSPN'2015)*, Paris, France, June 2015, pp. 63-77.

[10] E. Gabaldon, F. Guirado, J. L. Lerida, J. Planes, "Particle swarm optimization scheduling for energy saving in cluster computing heterogeneous environments," *Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Vienna, Austria, August 2016, pp. 321–325.

[11] E. Gabaldon, S. Vila, F. Guirado, J. L. Lerida, J. Planes, "Energy efficient scheduling on heterogeneous federated clusters using a fuzzy multi-objective meta-heuristics," *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Naples, Italy, 2017, pp. 1-6.

[12] M. T. Younis, S. Yang, "Hybrid meta-heuristic algorithms for independent job scheduling in grid computing," *Applied Soft Computing*, vol. 72, pp. 498-517, 2018.

[13] T. K. Ghosh, S. Das, N. Ghoshal, "Job scheduling in computational grid using a hybrid algorithm based on genetic algorithm and particle swarm optimization," O. Castillo, D. Jana, D. Giri, A. Ahmed (eds), *Recent Advances in Intelligent Information Systems and Applied Mathematics, ICITAM, Studies in Computational Intelligence*, vol. 863, Springer, 2019.

[14] A. Chhabra, G. Singh, and K. S. Kahlon, "Performance-aware energy-efficient parallel job scheduling in HPC grid using nature-inspired hybrid meta-heuristics," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 1801–1835, 2021.

[15] W. Abdulal, O. A. Jadaan, A. Jabas, and S. Ramachandram, "Mutation based simulated annealing algorithm for minimizing makespan in grid computing systems," *Proceedings of the IEEE International Conference on Network and Computer Science (ICNCS'2011)*, Kanyakumari, India, April 2011, pp. V6-90-V6-94.

[16] J. Yu, R. Buyya and C. K. Tham, "QoS-based scheduling of workflow applications on service grids," *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science'05)*, Melbourne, Australia, December 2005, pp. 1-8.

[17] S. H. Benedict and V. Vasudevan, "Improving scheduling of scientific workflows using tabu search for computational grids," *Information Technology Journal*, vol. 7, issue 1, pp. 91–97, 2008.

[18] M. Meddeber and B. Yagoubi, "Tasks assignment for Grid computing," *International Journal of Web and Grid Services*, Inderscience Enterprises Ltd., pp. 427-443, 2011.

[19] N. A. Bahnasawy, M. A. Koutb, M. Mosa and F. Omara, "A new algorithm for static task scheduling for heterogeneous distributed computing systems," *African Journal of Mathematics and Computer Science Research*, vol. 4, issue 6, pp. 221-234, 2011.

[20] A. M Bidgoli and Z. M. Nezad, "A new scheduling algorithm design for grid computing tasks," *Proceedings of the 5th Symposium on Advances in Science and Technology*, Khavaran Higher-education Institute, Mashhad, Iran, 2011, pp. 12-14.

[21] R. Goel, D. Singh, and Minakshi, "Scheduling algorithm design for grid computing," *International Journal of Innovations in Engineering and Technology (IJIET)*, vol. 3, issue 1, October 2013.

[22] H. S. Hossam, H. Abdel-Galil, and M. Belal, "WorkStealing algorithm for load balancing in grid computing," *International Journal of Advanced Computer Science and Applications*, vol. 12, issue 7, pp. 98-104, 2021.

[23] M. Rahman, R. Hassan, R. Ranjan, and R. Buyya, "Adaptive workflow scheduling for dynamic grid and cloud computing environment," *Concurrency and Computation: Practice and Experience*, vol. 25, pp. 1816–1842, 2013.

[24] P. Chauhan and Nitin, "Decentralized scheduling algorithm for DAG based tasks on P2P grid," *Hindawi Publishing Corporation Journal of Engineering*, vol. 2014, Article ID 202843, pp. 1-14, 2014.

[25] R. Garg and A. K. Singh, "Adaptive workflow scheduling in grid computing based on dynamic resource availability," *Engineering Science and Technology, an International Journal*, vol. 18, pp. 256-269, 2015.

***MOHEB R. GIRGIS** received his B.Sc. degree from Mansoura University, Egypt, in 1974, M.Sc. degree from Assuit University, Egypt, in 1980, and Ph.D. degree from the University of Liverpool, England, in 1986. He is a professor of computer science at Minia University, Egypt. His research interests include software engineering, software testing, information retrieval, evolutionary algorithms, image processing, computer networks, and bioinformatics.*

***TAREK M. MAHMOUD** received his B.Sc. degree from Minia University, Egypt, in 1984, M.Sc. degree from Assuit University, Egypt, in 1991, and Ph.D. degree from Berman University, German, in 1997. Currently, he is a professor of computer science at faculty of computers and artificial intelligence, University of Sadat City, Egypt. His research interests include computer networks, pattern recognition, social networks, analytics, web and text mining and artificial intelligence.*

***HAGAR M. AZZAM** received her B.Sc. degree from Minia University, Egypt, in 2005, M.Sc. degree from Minia University, Egypt, in 2016, and she is a Ph.D. student in Minia University, Egypt. she is an assistant lecturer of computer science at Minia University, Egypt. Her research interests include parallel computing, grid computing, cloud computing.*