An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements

Wei-Neng Chen, *Student Member, IEEE*, and Jun Zhang, *Senior Member, IEEE*

***Abstract*—Grid computing is increasingly considered as a promising next-generation computational platform that supports wide-area parallel and distributed computing. In grid environments, applications are always regarded as workflows. The problem of scheduling workflows in terms of certain quality of service (QoS) requirements is challenging and it significantly influences the performance of grids. By now, there have been some algorithms for grid workflow scheduling, but most of them can only tackle the problems with a single QoS parameter or with small-scale workflows. In this frame, this paper aims at proposing an ant colony optimization (ACO) algorithm to schedule large-scale workflows with various QoS parameters. This algorithm enables users to specify their QoS preferences as well as define the minimum QoS thresholds for a certain application. The objective of this algorithm is to find a solution that meets all QoS constraints and optimizes the user-preferred QoS parameter. Based on the characteristics of workflow scheduling, we design seven new heuristics for the ACO approach and propose an adaptive scheme that allows artificial ants to select heuristics based on pheromone values. Experiments are done in ten workflow applications with at most 120 tasks, and the results demonstrate the effectiveness of the proposed algorithm.**

***Index Terms*—Ant colony optimization (ACO), grid computing, workflow scheduling.**

I. Introduction

GRID computing has been increasingly considered as a promising next-generation computing platform that supports wide-area parallel and distributed computing since its advent in the mid-1990s [1], [2]. Analogous to the pervasive electrical power grid [3], computational grids enable the sharing, selection, and coordinated use of diverse computational resources from geographically distributed components, such as personal computers, workstations, clusters, and scientific instruments. Grid technologies satisfy the need of a large number of science and business computing applications, and make feasible the solution of these computation-intensive problems.

When processing a computing application in grids, we often regard the application as a workflow. Workflow is a wide concept in technology. In grid environments, we can define workflow as a collection of atomic tasks that are processed in a specific order to accomplish a complicated goal [4]. The workflow model based on loosely coupled coordination of atomic tasks has become one of the most attractive paradigms for grid computing applications [5].

One of the most challenging problems in grid computing is to schedule the workflow to achieve high performance. Usually, a workflow is given by a directed acyclic graph (DAG) [6] in which the nodes represent individual application tasks and the directed arcs stand for precedence relations between the tasks. The scheduler has to assign the tasks to heterogeneously distributed computing sites to process with the objective to achieve the customers' quality of service (QoS) requirements as well as to optimize the performance. As scheduling in a DAG is NP-complete [7] in general, the workflow scheduling problem is complicated and highly critical to the performance of a computational grid.

Traditional researches into scheduling workflows for grid applications have been focused on the problems with a single QoS parameter—makespan. There have been plenty of algorithms for makespan optimization in grid workflow applications. The most popular approaches are the static or dynamic list scheduling algorithms based on different heuristics, such as opportunistic load balancing (OLB), minimum execution time (MET), minimum completion time (MCT) [8], min–min [8]–[12], max–min [8]–[12], duplex [8], sufferage [10], [11], random [12], and heterogeneous earliest finish time (HEFT) [13]. Their basic idea is to determine an order of tasks based on heuristics and schedule the tasks according to the order. Besides, there are also some metaheuristic approaches, such as genetic algorithms (GAs) [14], [15], simulated annealing (SA) [16], and genetic simulated annealing (GSA) [17]. In general, metaheuristic approaches manage to obtain much better performance, but take a longer execution time [8], [13].

Recently, grid computing technologies have been reinforced by the open grid services architecture (OGSA) [18]. OGSA introduces Web services into the grid interoperability model and soon becomes the predominant technology for grids [19]. Under the new architecture, traditional scheduling algorithms given before may fail to work. First, the key trend of OGSA is the emergence of different grid service providers (GSPs) in grid market [18], [20] to provide computational resources. Instead of being bound to a certain amount of resource and time consumption, a task can be implemented by the service instances provided by different GSPs in the new architecture, and thus, the scheduler has to map each task to a suitable service instance. Second, makespan is no longer the only QoS parameter concerned by users. Instead, the QoS parameters such as makespan,

Manuscript received May 23, 2007; revised November 18, 2007. First published October 31, 2008; current version published December 22, 2008. This work was supported in part by the National Natural Science Foundation of China under Grant 60573066, in part by Guangdong Natural Science Foundation Research under Grant 5003346, and in part by the Science Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, China. This paper was recommended by Associate Editor B. Chaib-draa.

W.-N. Chen is with the Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, China.

J. Zhang (corresponding author) is with the Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, China (e-mail: junzhang@ieee.org).

Digital Object Identifier 10.1109/TSMCC.2008.2001722

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

cost, reliability, and security are found to be crucial for the performance of grids [5], [20], [21]. The scheduler has to consider the tradeoffs between different QoS parameters in order to satisfy the QoS requirements from both grid environments and users and optimize the performance of the whole application.

To tackle this much more difficult scheduling problem, several algorithms have been proposed. In Foster *et al.* [18] and Abramson *et al.* [22], some scheduling methods are proposed for the Nimrod/G model under the consideration of both time and cost. Nimrod/G [23] is an economy-driven architecture for a resource management and scheduling system in a global computational grid. The scheduling methods for Nimrod/G support user-defined deadline and budget and use the services of grid architecture for computational economy (GRACE) [22]. A dynamic grid scheduling algorithm is presented in [24] to consider the tradeoff between cost and time. But the algorithm does not support user-defined QoS constraints such as deadline and budget. Buyya and Tham [25] propose a workflow scheduling algorithm for economy-driven or market-driven grids to minimize cost under the deadline constraint. In their approach, tasks of the workflow are partitioned into branches and every branch is assigned a subdeadline. The algorithm works by applying the Markov decision procedure (MDP) to find a schedule with minimum cost for each branch that satisfies the subdeadline constraints. The computational grid is modeled in [19] and [26] as a queuing network, and a mixed-integer nonlinear programming (MINLP) algorithm is applied to optimize the cost of the application while ensuring the QoS requirements.

Besides makespan and cost, some recent researches also take security and reliability into account. Literature [27] presents an iterative scheduling algorithm for a set of independent tasks in computational grids. In their model, three dimensions of QoS parameters are considered: cost, deadline, and reliability. Each independent task has its own utility function and the algorithm is designed to maximize the utility function of the whole system. Kyriazis *et al.* [4] also take the previous three QoS parameters into account. It allows users to set preferences on different QoS parameters. An embedded scheduling scheme is proposed to find an optimal schedule that satisfies the user-defined thresholds and preferences of QoS parameters. The algorithm is tested in small-scale workflows with three tasks.

With the development of computational grids, more and more GSPs emerge in gird environments and the scale of workflow applications becomes larger and larger. Moreover, users may set up a set of QoS constraints for the workflow application and have their own preferences. A good scheduler should find a solution that satisfies the user-defined constraints and optimizes the user-preferred QoS parameters as well. In this case, traditional workflow scheduling strategies fail to work. First, most of the existing algorithms can only deal with one QoS parameter and neglect all the others. Second, in large-scale applications, the performance of traditional deterministic algorithms for workflow scheduling decreases quickly. In order to solve this intractable scheduling problem, this paper applies an ant colony optimization (ACO) approach. ACO is a metaheuristic proposed by Dorigo [28]–[32] in the light of the foraging behavior of ants. It works by simulating the pheromone-depositing and pheromone-following behavior of ants, and has been successfully applied to various intractable combinatorial optimization problems. The primary advantage of ACO for the workflow scheduling problem is that ACO manages to make full use of the instance-based heuristic information, which is found to be important for scheduling problems [6]. The great performance of ACO algorithms for scheduling problems has been demonstrated in [33]. Therefore, this paper takes advantage of the ant colony system (ACS) algorithm [29], which is one of the best ACO algorithms so far, for the grid workflow scheduling problem.

This paper focuses on large-scale workflow scheduling problems in a global grid environment with various QoS parameters. In our model, three of the basic QoS parameters are addressed: reliability, time, and cost. These parameters are important for a grid application and their characteristics are quite different. A user may submit a workflow application and specify the basic QoS demands: lower bound of reliability, deadline, and budget. Additionally, the user may prefer to optimize one of the QoS parameters. So the objective of the proposed ACO algorithm is to find a feasible schedule that satisfies all user-defined QoS parameters as well as optimizes the user-preferred QoS parameter. To fulfill this objective, seven instance-based heuristics are designed to guide the search behavior of ants. We also propose an adaptive scheme to manage these heuristics. The scheme allows artificial ants to use pheromone to choose heuristics when they are constructing solution schedules. The algorithm is tested on ten workflow applications of different scales. Experimental results reveal that the algorithm is effective.

The paper is organized as follows. Section II describes the background of workflow application, workflow management, and workflow scheduling in grids. Section III formulates the workflow scheduling problem considered in this paper. Section IV proposes the ACS algorithm for workflow scheduling. Section V gives the experimental results and analysis. The conclusion finally comes in Section VI.

II. Architecture for Workflow Management in Grids

Many computation-intensive applications in science and business can be described as workflows. According to [34], we can image the process of workflow applications as a four-level model (Fig. 1). A user may first submit a workflow application in abstract language in the *abstract specification level*. Then, the grid system has to select and configure the application components of the application to form an *abstract workflow*. The abstract workflow specifies the execution order of tasks. As already mentioned, under OSGA [3], tasks in a workflow are implemented by Web service instances provided by GSPs. The implementation of a task may be supported by various service instances provided by different GSPs with different QoS parameters, but only a single service instance is chosen for the execution of the task by the scheduler. Therefore, the third level is on a mission to map the tasks in the abstract workflow to service instances so that a *concrete workflow* is generated. For example, in Fig. 1, every task T_i in the abstract workflow is corresponding to a set of service instances $S_i = \{s_i^1, s_i^2, \ldots, s_i^{m_i}\}$. $s_i^1, s_i^2, \ldots, s_i^{m_i}$ are marked by circles in rectangle S_i and m_i is the total number of service instances for T_i. The service instances marked by gray circles are finally selected to map to corresponding tasks to form a concrete workflow in this example. At last, in the *implementation level*, different GSPs

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

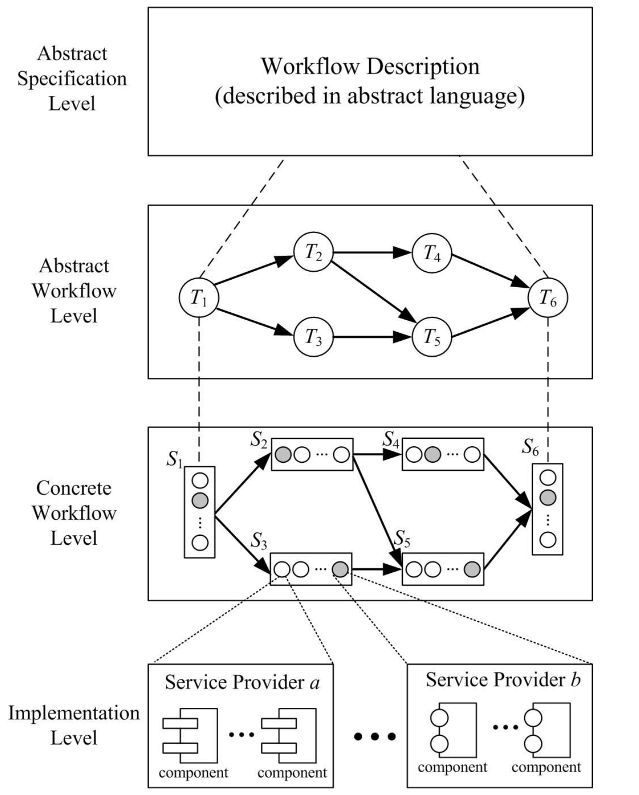


Fig. 1. Process of executing a workflow application in grids under OGSA: a four-level model.

may use different structures, which are transparent to users, to execute the same task with different QoS parameters.

In this four-level model, mapping the abstract workflow into the concrete workflow with QoS considerations is the most important step in grid applications [34]. In general, this task is completed by a workflow management system (WfMS). According to the market-driven structure of global computational girds [20], [25], the architecture of WfMS can be illustrated by Fig. 2. We can view the workflow management cycle in WfMS as the interplay of the following nine steps.

Step 1) The service instances provided by GSPs are registered to grid market directory (GMD). In a grid market, GMD [35] is used to manage the information, including the type, the provider, and the QoS parameters, of all service instances. Once an organization tends to promote services to the grid market, it first registers itself to GMD as a GSP. Then the information of new service instances is also registered.

Step 2) Users define and submit the abstract workflow to WfMS. QoS requirements of the workflow application are also submitted in this stage.

Step 3) As soon as WfMS accepts a workflow application, it asks GMD for an information list of the corresponding service instances. Typically, it collects the type, provider, access directory, and QoS parameters of each related service instance.

Step 4) WfMS enquires of each related GSP whether the service instances will be available. Only available services instances will be adopted.

Step 5) WfMS executes a scheduling algorithm to map the abstract workflow into a concrete workflow. Every task in the abstract workflow is mapped to an available service instance by the scheduler. The goal of scheduling is to achieve the users' QoS requirements and preferences. In Fig. 2, we can see that the scheduler is composed of two modules. The scheduling module maintains a scheduling algorithm to generate optimal schedules, while the QoS parameter recorder module records the run-time performance (such as reliability and availability of each service instance and GSP). These historical records will be used to predict and adjust the QoS parameters of service instances for future scheduling tasks.

Step 6) WfMS accesses the related GSPs to make reservations for all the service instances according to the solution schedule generated by step 5).

Step 7) The concrete workflow is executed according to the solution schedule.

Step 8) The contract between users and GSPs may be violated at times due to many reasons such as the failure of GSPs' resources. In this case, a service instance may be delayed or even become unavailable, which makes the following service instances fail to follow the schedule. Therefore, a rescheduling mechanism is required for violation management.

Step 9) After the completion of each service instances, the actual QoS is fed back to the QoS parameter recorder module so that historical QoS statistics can be updated.

In this paper, we focus on the scheduling module to propose an effective scheduling algorithm to generate concrete workflows for large-scale workflow applications.

III. Problem Formulation

Intuitionally, the function of the workflow scheduling algorithm for computational grids is to map all tasks of the abstract workflow to service instances to generate an optimal concrete workflow that satisfies the user-defined QoS constraints and maximizes the performance in terms of the users' preferences. In our study, we address three of the most important QoS parameters: time, cost, and reliability. They have quite different characteristics. Reliability of an application is defined by the minimum reliability of all selected service instances in the concrete workflow. Cost of an application is defined by summating the cost of all selected service instances. Makespan of the workflow is in nonlinear relation with each individual service instance. The scheduling model enables users to define thresholds for QoS parameters, i.e., to set deadline, budget, and the lower bound of reliability for the workflow application. Furthermore, user-defined QoS preferences are also supported. The objective of the scheduling algorithm is to find a schedule that optimizes the user-preferred QoS parameter and satisfies all QoS restrictions.

We model the abstract workflow as a DAG $G = (V, A)$. Let n be the number of tasks in the workflow. The set of nodes $V = \{T_1, T_2, \ldots, T_n\}$ corresponds to the tasks of the workflow. The set of arcs A represents precedence relations between

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

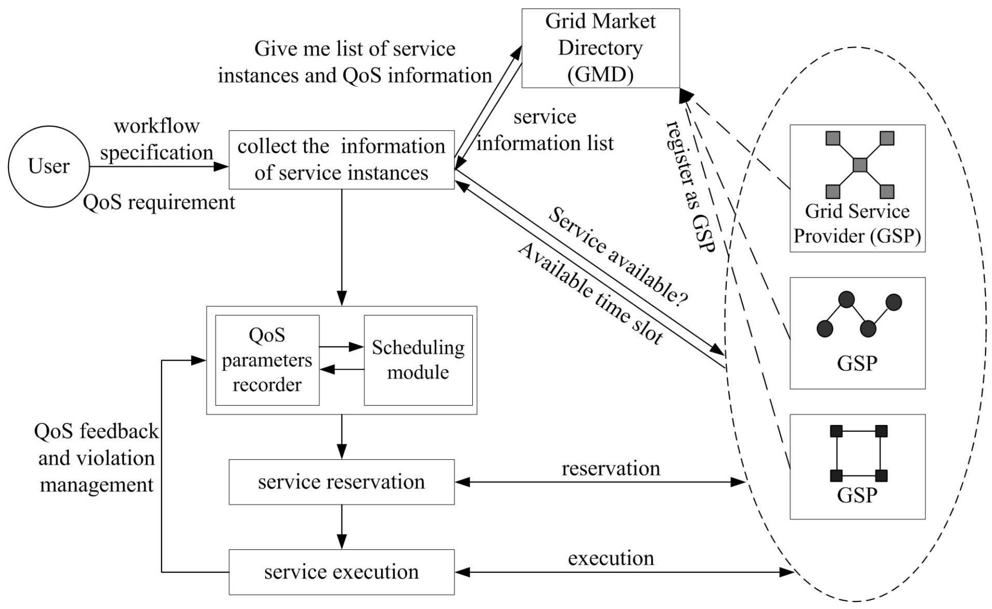


Fig. 2. Architecture for grid workflow applications and management.

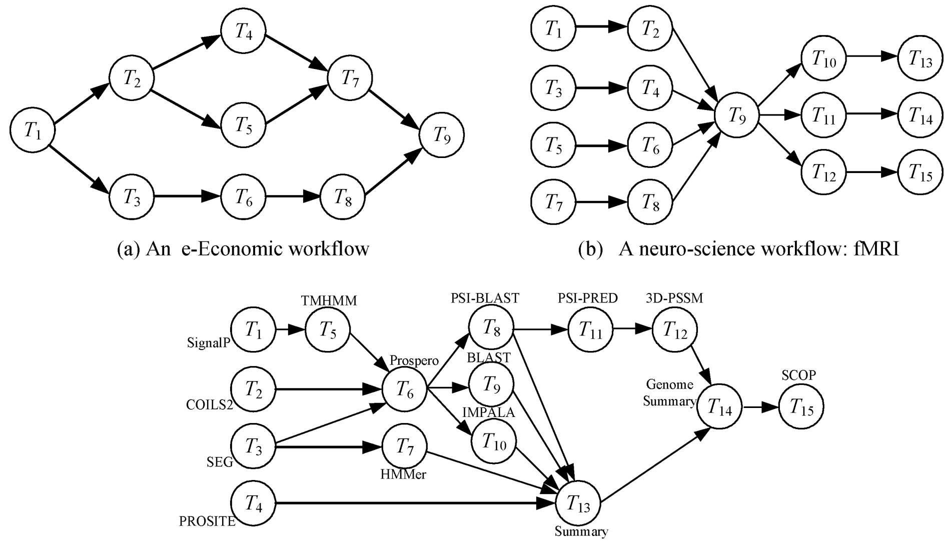


Fig. 3. Workflow applications used in our experiment. (a) Simple application of workflow with nine tasks in e-economic. (b) Neuroscience workflow with 15 tasks named fMRI [36]. (c) E-protein workflow with 15 tasks [37].

tasks. An arc is in the form of (T_i, T_j), where T_i is called the parent task of T_j and T_j is the child task of T_i. Normally, a child task can only be executed until all of its parent tasks have been completed. The set of parent tasks of T_i is denoted by Pred(T_i), and the set of child tasks by Succ(T_i). Examples of a workflow described by DAG are given in Fig. 3. For the sake of convenience, we usually add two dummy nodes T_{start} and T_{end} to stand for the beginning and the end of the workflow.

Each task T_i $(1 \le i \le n)$ has an implementation domain $S_i = \{s_i^1, s_i^2, \ldots, s_i^{m_i}\}$, where s_i^j $(1 \le j \le m_i)$ represents a service instance provided by a GSP and m_i is the total number of available service instances for T_i. The properties of a service instance can be denoted as a group of four variables ($s_i^j.g$, $s_i^j.r$, $s_i^j.t$, $s_i^j.c$). $s_i^j.g$ means that the GSP of s_i^j is $s_i^j.g$. $s_i^j.r$, $s_i^j.t$, and $s_i^j.c$ stand for reliability, execution time, and cost of s_i^j, respectively.

Precedence constraints are embedded in the model, which is given by the DAG. Precedence constraints specify the execution order of tasks that a feasible solution must satisfy.

In addition, there are also user-defined QoS constraints in the workflow scheduling model. As three QoS parameters are considered in this paper, there are correspondingly three kinds of QoS constraints.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

1) *Reliability constraints:* The reliability of the generated concrete workflow must be not smaller than a user-defined variable *ReliabilityConstraint*. In other words, given a schedule $K(K_1, \ldots, K_n)$, which means T_i is executed by $s_i^{K_i}$, if K*.reliability* is the reliability of schedule K, then K satisfies the reliability constrains only if

$$K.reliability = \min_{1 \le i \le n} S_i^{K_i}.r \ge ReliabilityConstraint. \quad (1)$$

2) *Makespan constraints:* Total execution time of the workflow must not be larger than a user-defined variable *Deadline*. In other words, if K*.makespan* is the total execution time of K, then K should satisfy

$$K.makespan \le Deadline. \quad (2)$$

3) *Cost constraints:* Given a schedule $K(K_1, \ldots, K_n)$, the total cost of K (K*.cost*) must not be larger than a user-defined variable *budget*, i.e.,

$$K.cost = \sum_{i=1}^{n} s_i^{K_i}.c \le Budget. \quad (3)$$

Users may also define QoS preferences based on which the objective function of the optimization problem is built.

4) *Reliability optimization:* The user prefers to optimize the reliability of the application. In this case, the user often sets makespan and cost constraints [(2) and (3)]. The objective of the scheduling algorithm is to find a schedule K that satisfies all constraints and maximizes the value of K*.reliability*.
5) *Makespan optimization:* The user prefers to optimize the makespan of the application. In this case, the user often sets reliability and cost constraints [(1) and (3)]. The objective of the scheduling algorithm is to find a schedule K that satisfies all constraints and minimizes the value of K*.makespan*.
6) *Cost optimization:* The user prefers to optimize the cost of the application. In this case, the user often sets reliability and makespan constraints [(1) and (2)]. The objective of the scheduling algorithm is to find a schedule K that satisfies all constraints and minimizes the value of K*.cost*.

IV. ACS Algorithm for the Scheduling Problem

The elementary idea of ACO is to simulate the foraging behavior of ant colonies. When a group of ants sets out from the nest to search for the food source, they use a special kind of chemical to communicate with each other. The chemical is referred to as pheromone. Once the ants discover a path to food, they deposit pheromone on the path. By sensing pheromone on the ground, an ant can follow the trails of the other ants to the food source. As this process continues, most of the ants tend to choose the shortest path as there have been a huge amount of pheromones accumulated on this path. This collective pheromone-depositing and pheromone-following behavior of ants becomes the inspiring source of ACO.

In this paper, we apply the ACS algorithm [31], which is one of the best ACO algorithms so far [32], to tackle the workflow scheduling problem in grid applications. Informally, the algorithm can be viewed as the interplay of the following procedures.

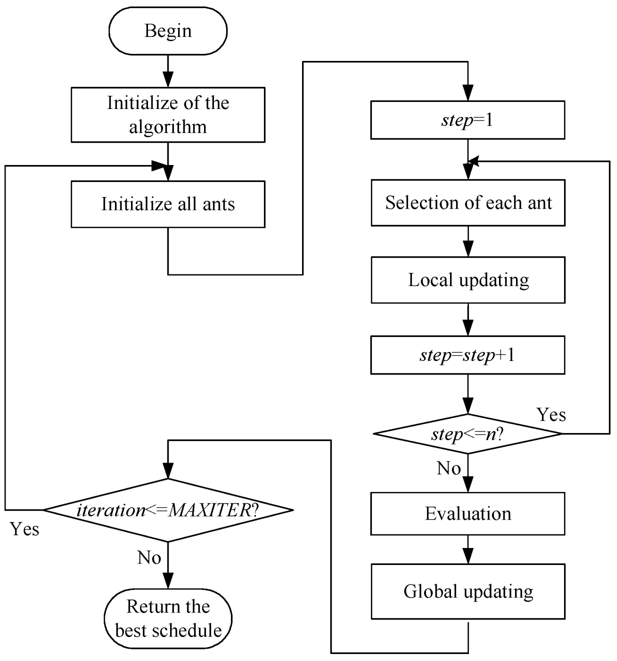


Fig. 4. Flowchart of the ACS algorithm.

1) *Initialization of algorithm:* All pheromone values and parameters are initialized at the beginning of the algorithm.
2) *Initialization of ants:* A group of M artificial ants are used in the algorithm. In each iteration, each ant randomly selects a constructive direction and builds a sequence of tasks.
3) *Solution construction: M* ants set out to build M solutions to the problem based on pheromone and heuristic values using the selection rule of the ACS algorithm.
4) *Local pheromone updating:* Soon after an ant maps a service instance s_i^j to task T_i, the corresponding pheromone value is updated by a local pheromone updating rule.
5) *Global pheromone updating:* After all ants have completed their solutions at the end of each iteration, pheromone values corresponding to the best-so-far solution are updated by a global pheromone updating rule.
6) *Terminal test:* If the test is passed, the algorithm will be ended. Otherwise, go to step 2) to begin a new iteration.

The flowchart of the algorithm is given in Fig. 4. These procedures are described in detail shortly.

A. Definition of Pheromone and Heuristic Information

Pheromone and heuristic information are the most important factors of an ACO algorithm. In general, pheromone is used to record the historical searching experiences and bias the ants' searching behavior in future. On the other hand, heuristic information is some problem-based values to guide the search direction of ants. As the scheduling problem is mainly to map all tasks in the abstract workflow to service instances to form a concrete workflow, we denote the pheromone value of mapping service instance s_i^j to task T_i as τ_{ij}, and the heuristic information value of mapping s_i^j to T_i as η_{ij}.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

At the beginning of the algorithm, we set all pheromone values to an initial value τ_0, i.e.,

$$\tau_{ij} = \tau_0, \qquad 1 \le i \le n, \quad 1 \le j \le m_i. \tag{4}$$

Moreover, as there are multiple QoS parameters with different characteristics in the considered model, we defined seven heuristics for the algorithm as follows.

1) Heuristic A: Reliability Greedy (RG): The RG heuristic biases the artificial ants to select the service instances with higher reliabilities. Suppose that an ant's heuristic type is the RG heuristic, then the heuristic value of mapping s_i^j to T_i (denoted as RG_{ij}) is set to

$$\eta_{ij} = \mathrm{RG}_{ij} = \frac{s_i^j.r - \mathit{min\_reliability}_i + 1}{\mathit{max\_reliability}_i - \mathit{min\_reliability}_i + 1} \tag{5}$$

where $\mathit{min\_reliability}_i = \min_{1\le j\le m_i}\{s_i^j.r\}$ and $\mathit{max\_reliability}_i = \max_{1\le j\le m_i}\{s_i^j.r\}$. According to (5), a service instance with higher reliability will be associated with a higher heuristic value. Equation (5) also certifies that the value of η_{ij} is normalized to the interval $(0, 1]$. The reason of adding 1 in both numerator and denominator is to prevent the case of zero divided.

2) Heuristic B: Time Greedy (TG): The TG heuristic biases the artificial ants to select the service instances with shorter execution time. Suppose that an ant's heuristic type is the TG heuristic, then the heuristic value of mapping s_i^j to T_i (denoted as TG_{ij}) is set to

$$\eta_{ij} = \mathrm{TG}_{ij} = \frac{\mathit{max\_time}_i - s_i^j.t + 1}{\mathit{max\_time}_i - \mathit{min\_time}_i + 1} \tag{6}$$

where $\mathit{min\_time}_i = \min_{1\le j\le m_i}\{s_i^j.t\}$ and $\mathit{max\_time}_i = \max_{1\le j\le m_i}\{s_i^j.t\}$. According to (6), a service instance with shorter execution time will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

3) Heuristic C: Cost Greedy (CG): The CG heuristic biases the artificial ants to select the service instances with lower cost. Suppose that an ant's heuristic type is the CG heuristic, then the heuristic value of mapping s_i^j to T_i (denoted as CG_{ij}) is set to

$$\eta_{ij} = \mathrm{CG}_{ij} = \frac{\mathit{max\_cost}_i - s_i^j.t + 1}{\mathit{max\_cost}_i - \mathit{min\_cost}_i + 1} \tag{7}$$

where $\mathit{min\_cost}_i = \min_{1\le j\le m_i}\{s_i^j.t\}$ and $\mathit{max\_cost}_i = \max_{1\le j\le m_i}\{s_i^j.t\}$. According to (7), a service instance with lower cost will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

4) Heuristic D: Suggested Deadline (SD): There are always tradeoffs between QoS parameters. For example, a service instance with shorter duration may have higher cost or lower reliability. With the consideration of such tradeoffs and the restriction of deadline, the SD heuristic biases the artificial ants to select the just-in-time service instances. To achieve this objective, we assign suggested deadlines to every task in the abstract workflow based on the user-defined deadline of the application.

To evaluate the SD heuristic for every task, we have to first evaluate the earliest start time [(2) and (3)], and the backward earliest start time of every task.

a) *Earliest start time of task T_i (EST_i):* The value of EST_i is evaluated by first mapping every task T_i to the service instance with the shortest execution time. EST_i equals to the start time of T_i under this mapping strategy. Moreover, the total makespan of the workflow under this mapping strategy regardless of resource constraints can be viewed as the estimated minimum makespan of the workflow application. We denote this estimated value as *min_Makespan*

b) *Backward earliest start time of T_i($BEST_i$):* We convert the DAG into a backward network by considering the starting node of the DAG as the ending node and vice versa, and reversing the direction of all directed arcs. For every T_i, the value of EST_i in the backward network is BEST_i.

Based on these two attributes, we can evaluate the average minimum execution time of task T_i from both forward traverse and backward traverse (denoted as $\mathit{avg\_min\_time}_i$) given by (8), as shown at the bottom of the page.

By enlarging the value of $\mathit{avg\_min\_time}_i$ on a scale of *Deadline*:*min_Makespan*, we can obtain the value of SD_i as

$$\mathrm{SD}_i = \left\lfloor \mathit{avg\_min\_time}_i \cdot \frac{\mathit{Deadline}}{\mathit{min\_Makespan}} \right\rfloor. \tag{9}$$

Suppose that an ant's heuristic type is the SD heuristic, then the heuristic value of mapping s_i^j to T_i is set to (10), as shown at the bottom of the page.

According to (10), a service instance, the execution time of which is closer to SD_i, will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

5) Heuristic E: Suggest Budget (SB): Similar to the SD heuristic, the SB heuristic biases the artificial ants to select the service instances with just-within-budget cost. To achieve this objective, for each task T_i, we assign a suggested budget SB_i based on the user-defined budget of the application.

By mapping all tasks to the service instances with the lowest cost, we can obtain the minimum cost of the whole workflow application (denoted as *min_Cost*). That is, $\mathit{min\_Cost} = \sum_{j=1}^{n} \mathit{min\_cost}_j$. The suggested budget SB_i for T_i is evaluated by enlarging the value of $\mathit{min\_cost}_i$ on a scale of

$$\mathit{avg\_min\_time}_i = \frac{(\min_{\forall T_j \in \mathrm{succ}(T_i)}\{EST_j\} - EST_i) + (\min_{\forall T_j \in \mathrm{pred}(T_i)}\{BEST_j\} - BEST_i)}{2} \tag{8}$$

$$\eta_{ij} = \frac{\max\{|\mathit{max\_time}_i - \mathrm{SD}_i|, |\mathrm{SD}_i - \mathit{min\_time}_i|\} - |s_i^j.t - \mathrm{SD}_i| + 1}{\max\{|\mathit{max\_time}_i - \mathrm{SD}_i|, |\mathrm{SD}_i - \mathit{min\_time}_i|\} + 1} \tag{10}$$

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

Budget:*min_Cost*

$$\mathrm{SB}_i = \mathit{min\_cost}_i \cdot \frac{\mathit{Budget}}{\mathit{min\_Cost}}. \tag{11}$$

Suppose an ant's heuristic type is the SB heuristic, then the heuristic value of mapping s_i^j to T_i is set to (12), as shown at the bottom of the page.

According to (12), a service instance, the cost of which is closer to SB_i, will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

6) Heuristic F: Time/Cost (TC): The TC heuristic considers the effectiveness of both time and cost of a service instance. It integrates the TG heuristic with the CG heuristic. Suppose that an ant's heuristic type is the TC heuristic, then the heuristic value of mapping s_i^j to T_i is set to

$$\eta_{ij} = \frac{1}{2}\left(\mathrm{TG}_{ij} + \mathrm{CG}_{ij}\right). \tag{13}$$

According to (13), a service instance with shorter execution time and lower cost will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

7) Heuristic G: Overall Performance (OP): The effectiveness of all QoS parameters (including reliability, time, and cost) is considered in the OP heuristic. It unites the TG, CG, and RG heuristics together. Suppose that an ant's heuristic type is the OP heuristic, then the heuristic value of mapping s_i^j to T_i is set to

$$\eta_{ij} = \frac{1}{3}\left(\mathrm{TG}_{ij} + \mathrm{CG}_{ij} + \mathrm{RG}_{ij}\right). \tag{14}$$

According to (14), a service instance with shorter execution time, lower cost, and higher reliability will be associated with a higher heuristic value and $\eta_{ij} \in (0, 1]$.

Based on different user-defined QoS preferences, the algorithm uses different heuristics. 1) If the objective of the algorithm is to optimize reliability, the algorithm will use all of these seven heuristics. Therein, the RG and OP heuristic are used to find service instances with high reliability, and the other heuristics are applied to ensure that the cost and makespan of the schedule meet the QoS constraints. 2) If the objective is to optimize makespan, only the TG, CG, SB, and TC heuristic will be used. The TG and TC heuristic are used to find the service instances with shorter execution time, and the CG, SB, and TC heuristic are used to search for the service instances that satisfy the budget constraints. In this case, reliability constraints can be achieved by simply not choosing the service instances the reliability of which is lower than the reliability constraint, so the RG and OP heuristic is not needed. As the algorithm aims at minimizing the makespan, the SD heuristic is not needed here either. 3) If the objective is to optimize the cost, only the TG, CG, SD, and TC heuristic will be used.

We adopt an adaptive scheme to manage these seven heuristics in the algorithm. These seven types of heuristic information are selected by artificial ants based on pheromone values. The pheromone value corresponding to heuristic A (the RG heuristic) is denoted as τ_A, the one corresponding to heuristic B (the TG heuristic) is denoted as $\tau_B, \ldots$, and the one corresponding to heuristic G (the OP heuristic) is denoted as τ_G. Initially, the pheromone values of all adoptive heuristics are set to τ_0, and the ones of all unused heuristics are set to 0.

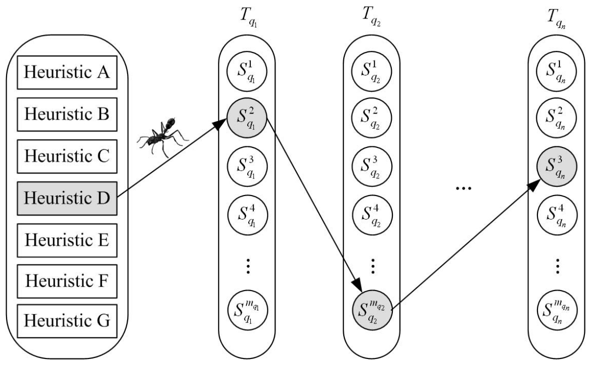

Heuristic A
Heuristic B
Heuristic C
Heuristic D
Heuristic E
Heuristic F
Heuristic G
T_{q_1}
T_{q_2}
T_{q_n}
$S_{q_1}^1$ $S_{q_1}^2$ $S_{q_1}^3$ $S_{q_1}^4$ $S_{q_1}^{m_{q_1}}$
$S_{q_2}^1$ $S_{q_2}^2$ $S_{q_2}^3$ $S_{q_2}^4$ $S_{q_2}^{m_{q_2}}$
$S_{q_n}^1$ $S_{q_n}^2$ $S_{q_n}^3$ $S_{q_n}^4$ $S_{q_n}^{m_{q_n}}$

Fig. 5. Procedure of an ant to build a solution schedule.

B. Construction of Solution Schedules

In every iteration of the algorithm, a group of M ants sets out to build solutions. The procedure of solution construction can be divided into two steps.

1) Initialization of Ants: At the beginning of each iteration, every ant randomly selects a constructive direction (forward or backward). A forward ant will normally traverse the workflow network in terms of the given precedence relations. Oppositely, a backward ant will begin the searching from the ending node of the DAG and reverse the directions of all arcs, just as what we do when calculating the value of backward earliest start time. The strategies of scheduling from both directions enable the algorithm to explore more different solutions for DAG-based scheduling problems, which have been proven in [33].

Additionally, every ant has to build a sequence of tasks that satisfies the precedence constraints given by the DAG. (Notice that the sequences generated by backward ants must satisfy the precedence constraints in the backward network.) The order of mapping tasks to service instances will be based on this sequence. For the sake of convenience, the sequence of a forward ant is marked as $(T_{\mathrm{start}}, T_{q_1}, T_{q_2}, \ldots, T_{q_n}, T_{\mathrm{end}})$, and the sequence of a backward ant is $(T_{\mathrm{end}}, T_{q_1}, T_{q_2}, \ldots, T_{q_n}, T_{\mathrm{start}})$, where $q_1, q_2, \ldots, q_n$ is a permutation of $1, 2, \ldots, n$. The sequence is built by randomly choosing a task that satisfies the precedence constraints. The reason of building a random sequence is to provide a fair environment for the mapping process of every task.

2) Solution Construction: In this step, M artificial ants build M solutions to the problem. Each ant maintains a building process and all ants construct their solutions in parallel. The building process for an ant to construct a complete solution can be illustrated by Fig. 5.

$$\eta_{ij} = \frac{\max\{|\mathit{max\_cost}_i - \mathrm{SB}_i|, |\mathrm{SB}_i - \mathit{min\_cost}_i|\} - \left|s_i^j.c - \mathrm{SB}_i\right| + 1}{\max\{|\mathit{max\_cost}_i - \mathrm{SB}_i|, |\mathrm{SB}_i - \mathit{min\_cost}_i|\}}. \tag{12}$$

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

At the beginning, out of the seven heuristics mentioned in Section IV-A, each ant adaptively chooses one kind of heuristic information based on their associated pheromone values. The roulette wheel scheme (RWS) is applied for the selection here. That is, the probability of choosing heuristic A is directly proportional to the pheromone value associated with heuristic A (τ_A), and the same rule happens to all the other heuristics.

After selecting a type of heuristic information, the ants begin to construct schedules of the problem. In every step, an ant chooses a service instance in terms of the selected heuristic information and the pheromone values to map to the first unmapped task in the sequence. The selection rule of mapping a service instance S_i^j to T_i is given by

$$S_i^j = \begin{cases} \arg\max_{1\le j\le m_i} \tau_{ij}\beta^{\eta_{ij}}, & q \le q_0 \\ \text{apply the roulette wheel scheme}, & \text{otherwise.} \end{cases} \tag{15}$$

Equation (15) shows the pseudorandom proportion selection rule in the ACS algorithm. In this selection rule, a random number $q \in [0, 1]$ is generated and compared with a parameter $q_0 \in [0, 1]$. If $q \le q_0$, the ant maps T_i to the service instance that has the largest value of $\tau_{ij}\beta^{\eta_{ij}}$. Otherwise, the RWS selection method will be used, in which the probability of selecting S_i^j is directly proportional to the value of $\tau_{ij}\beta^{\eta_{ij}}$. Here, $\beta \ge 1$ is a parameter to determine the relative influence of pheromone and heuristic information. We have mentioned in Section IV-A that all heuristic values satisfy $\eta_{ij} \in (0, 1]$, so the values of $\beta^{\eta_{ij}}$ satisfy $\beta^{\eta_{ij}} \in (1, \beta]$.

In each iteration, every ant iterates this selection scheme for N times so that N tasks are mapped to N corresponding service instances and a complete solution schedule is consequently built. In some case, if the quality of a partial solution built by an ant is already worse than the best-so-far ant, this partial solution will be deserted.

C. Pheromone Management

1) Pheromone Initialization: In the ACS algorithm, all pheromone values are initially set to τ_0. τ_0 is the minimum value of all pheromone values [31], [32]. In this paper, based on different QoS preferences, the value of τ_0 is given by the following equation:

$$\tau_0 = \begin{cases} \dfrac{\textit{min\_Reliability}}{\textit{max\_Reliability}}, & \text{reliability optimization} \\ \dfrac{\textit{min\_Makespan}}{\textit{max\_Makespan}}, & \text{makespan optimization} \\ \dfrac{\textit{min\_Cost}}{\textit{max\_Cost}}, & \text{cost optimization.} \end{cases} \tag{16}$$

In (16), *min_Reliability* is the minimum reliability of all service instances and *max_Reliability* $= 100$. *min_Makespan* is the estimated minimum makespan of the workflow application, and *max_Makespan* is the estimated maximum makespan. These two values can be evaluated by mapping every task to the service instance with the minimum (or maximum) execution time regardless of resource constraints. Similarly, *min_Cost* and *max_Cost* are the minimum and maximum cost of the workflow application, respectively. They can be evaluated by mapping every task to the service instance with the minimum (or maximum) cost.

2) Local Pheromone Updating: In the ACS algorithm, immediately after an ant has mapped a task T_i to a service instance s_i^j, the local pheromone updating rule is applied to reduce the attraction of s_i^j for the later ants. The local pheromone updating rule is given by the following equation:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\tau_0. \tag{17}$$

In (17), $\rho \in (0, 1)$ is a parameter. As τ_0 is the minimum value of all pheromone values, the function of the local updating rule is to decrease the value of τ_{ij} to enhance diversity of the algorithm.

The pheromone values of heuristics also obey the local pheromone updating rule. After an ant picks up a heuristic type (for example, heuristic A) at the beginning of each iteration, the pheromone value related to heuristic type A is immediately modified by the following equation:

$$\tau_A = (1-\rho)\tau_A + \rho\tau_0. \tag{18}$$

3) Global Pheromone Updating: Global updating takes place after all ants have built their solutions. The algorithm first compares all solutions in that iteration. The quality of a solution schedule K can be evaluated by the following equations (19)–(21). In the case of reliability optimization, the score of schedule K is given by

$$K.\textit{score} = \begin{cases} 0.5\cdot\dfrac{\textit{Budget}}{K.\textit{cost}} + 0.5\dfrac{\textit{Deadline}}{K.\textit{makespan}} + \dfrac{\textit{min\_Reliability}}{\textit{max\_Reliability}}, \\ \qquad \text{if } K.\textit{cost}>\textit{Budget} \text{ and } K.\textit{makespan} > \textit{Deadline} \\ 0.5 + 0.5\dfrac{\textit{Deadline}}{K.\textit{makespan}} + \dfrac{\textit{min\_Reliability}}{\textit{max\_Reliability}}, \\ \qquad \text{if } K.\textit{cost} \le \textit{Budget} \text{ and } K.\textit{makespan} > \textit{Deadline} \\ 0.5\cdot\dfrac{\textit{Budget}}{K.\textit{cost}} + 0.5 + \dfrac{\textit{min\_Reliability}}{\textit{max\_Reliability}}, \\ \qquad \text{if } K.\textit{cost} > \textit{Budget} \text{ and } K.\textit{makespan} \le \textit{Deadline} \\ 1 + \dfrac{K.\textit{reliability}}{\textit{max\_Reliability}}, \\ \qquad \text{if } K.\textit{cost} \le \textit{Budget} \text{ and } K.\textit{makespan} \le \textit{Deadline}. \end{cases} \tag{19}$$

The score of schedule K in terms of (19) is composed of two parts: penalties of QoS constraints and quality of the user-preferred QoS parameter. The score for each part is a value between (0,1], so the value of K.*score* is limited to the interval of (0,2]. If K satisfies all QoS constraints, its score for QoS constraints will be set to 1, and the score for the user-preferred QoS parameter will be set according to the reliability of K. Higher reliability will contribute to a higher score. On the other hand, if K fails to satisfy all QoS constraints, its score for QoS constraints will be set according to the degree of satisfaction, and the score for the user-preferred QoS parameter will be set to a minimum value. The larger the score is, the better the schedule will be.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

Similarly, in the case of makespan optimization, the score of schedule K is given by

$$K.score = \begin{cases} \dfrac{Budget}{K.cost} + \dfrac{min\_Makespan}{max\_Makespan}, & \text{if } K.cost > Budget \\ 1 + \dfrac{min\_Makespan}{K.makespan}, & \text{if } K.cost \leq Budget. \end{cases} \tag{20}$$

In the case of cost optimization, the score of schedule K is given by

$$K.score = \begin{cases} \dfrac{Deadline}{K.makespan} + \dfrac{min\_Cost}{max\_Cost}, & \text{if } K.makespan > Deadline \\ 1 + \dfrac{min\_Cost}{K.cost}, & \text{if } K.makespan \leq Deadline. \end{cases} \tag{21}$$

Global pheromone updating rule is only applied to the components on the best-so-far solution—the solution has the largest score value. If $K(K_1, \ldots, K_n)$ is the best-so-far solution, [$(K_1, \ldots, K_n)$ means task T_i is mapped to the service instance $s_i^{K_i}$], the global pheromone updating rule is given by

$$\tau_{iK_i} = (1-\rho)\tau_{iK_i} + \rho \cdot K.score, \qquad i = 1, 2, \ldots, n. \tag{22}$$

In this equation, $\rho \in (0, 1)$ is the same parameter as (17). The effect of the global pheromone updating rule is to increase the pheromone values associated with the best-so-far solution so that these good mappings will be more attractive in future iterations. In the same way, the heuristic type related to the best-so-far ant is also reinforced by the global updating rule. That is, if the heuristic type of the best-so-far solution is heuristic A, the pheromone value related to A will be modified by the following equation at the end of each iteration:

$$\tau_A = (1-\rho)\tau_A + \rho K.score, \qquad i = 1, 2, \ldots, n \tag{23}$$

where K is the best-so-far solution in this equation.

V. Experimental Analysis

A. Test Instances

We test the ACS algorithm in ten workflow applications. The basic information of these workflow applications is shown in Table I. The first three workflows, including the e-economic application with nine tasks, the neuroscience application [functional MRI (fMRI)] with 15 tasks [36], and the e-protein workflow with 15 tasks [37], are derived from real-life applications. The DAGs of these applications are shown in Fig. 3. The other seven workflows are generated based on the networks in the project scheduling problem library (PSPLIB) [38], which is a library for project scheduling problems. These networks include j301_1 with 30 tasks, j601_1 and j601_2 with 60 tasks, j901_1 and j901_2 with 90 tasks, and j1201_1 and j1201_2 with 120 tasks. In our experiment, we randomly assign six to ten service instances to each task in the workflows. The QoS parameters (reliability, execution time, and cost) of all service instances are also randomly generated, but they follow the rule that for the same task, a service instance with higher reliability or shorter execution time may cost more money and vice versa.

TABLE I
TEST INSTANCES

Instance Name	Number of Tasks	Network
e-Economic	9	Fig. 3(a)
fMRI	15	Fig. 3(b)
e-Protein	15	Fig. 3(c)
j301_1	30	j301_1 (PSPLIB)
j 601_1	60	j 601_1 (PSPLIB)
j 601_2	60	j 601_2 (PSPLIB)
j 901_1	90	j 901_1 (PSPLIB)
j 901_2	90	j 901_2 (PSPLIB)
j 1201_1	120	j 1201_1 (PSPLIB)
j 1201_2	120	j 1201_2 (PSPLIB)

B. Parameters of the Algorithm

We first test the parameters of the ACS algorithm. There are mainly three parameters in the algorithm: β and q_0 in the pseudo-random proportion selection rule [(15)] and ρ in the pheromone updating rule [(17), (18), (22), and (23)]. In our experiments, we set $\rho = 0.1$, which is the same as the suggestion given by the traditional ACS algorithm for traveling salesman problem (TSP) [31], [32]. We find that this configuration still performs well in the workflow scheduling problem.

We also investigate the configurations for β and q_0 under different levels of restrictions. On the one hand, β weights the related influence of pheromone and heuristic information. $\beta = 1$ means that the heuristic information is not used by ants at all, and a large value of β implies that heuristic information plays a more influential role in the selection scheme. In the investigation, we test $\beta = \{1, 1.02, 1.05, 1.10, 1.15, 1.2, 1.3, 1.5, 1.8, 2, 3, 5, 10, 20, 50\}$. On the other hand, q_0 determines the probability of choosing the best component directly or applying the RWS scheme to choose. A large value of q_0 results in a more aggressive behavior of ants—if $q_0 = 1$, ants will only choose the component with the largest value of $\tau_{ij}\beta^{\eta_{ij}}$, which makes the algorithm become a greedy search procedure. However, if $q_0 = 0$, the algorithm will only use the RWS selection rule, which will be more stochastic and less convergent. In the experiment, we test $q_0 = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

We test the performance of both parameters under different constraint environments. In a tight constraint environment, the problem of finding a feasible solution to meet the user-defined QoS constraints is already very difficult. On the contrary, in a loose constraint environment, it is easy to find feasible solutions so that the algorithm is able to concentrate only on optimizing the user-preferred QoS parameter. The results are shown in Figs. 6–8. In these plots, deeper colors represent better results.

Fig. 6 illustrates the results of makespan optimization in the e-protein workflow and the j601_1 workflow. For q_0, no matter in what constraint environment, $q_0 \in [0.7, 0.9]$ is always the best for the e-protein workflow and $q_0 = 0.9$ is the best for the j601_1 workflow. In the other large-scale instances with more than 30

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

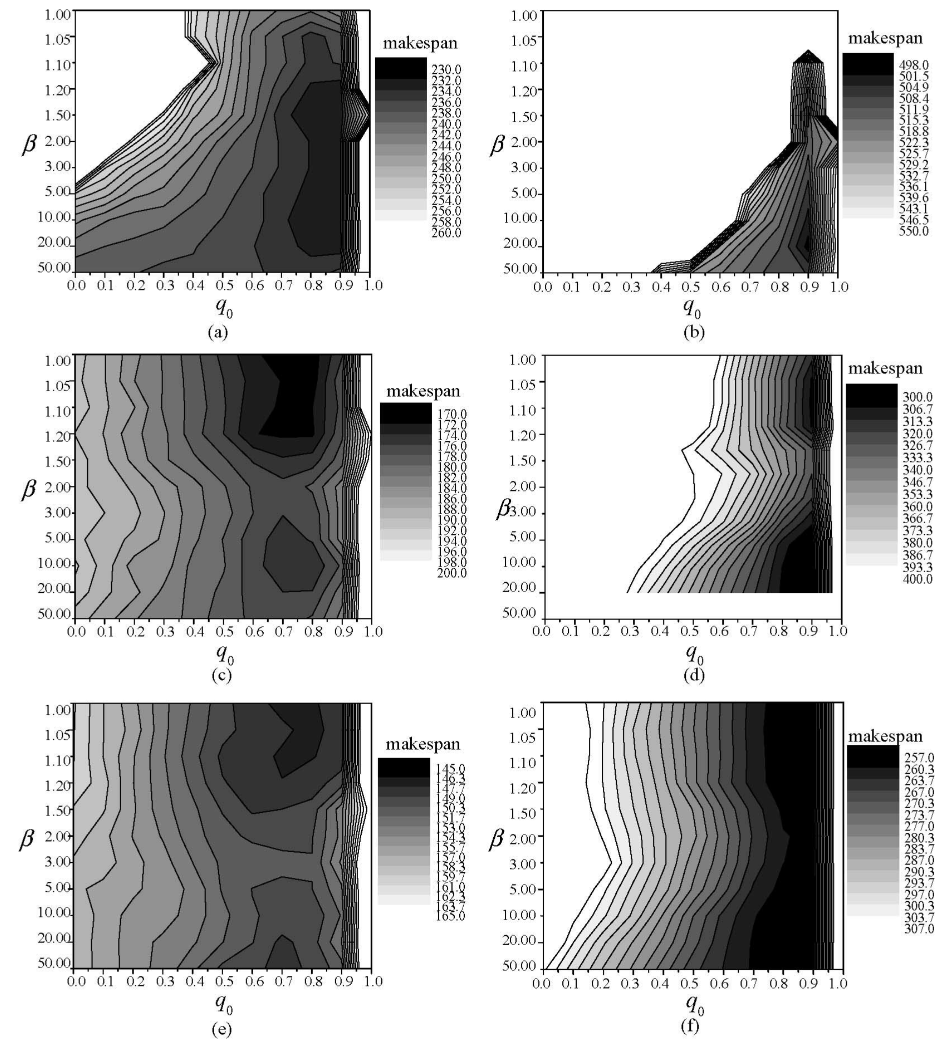


Fig. 6. Performance of different configurations for β and q_0 under different QoS constraints in the case of makespan optimization. (a) Instance e-protein with tight budget constraints. (b) Instance j601_1 with tight budget constraints. (c) Instance e-protein with medium budget constraints. (d) Instance j601_1 with medium budget constraints. (e) Instance e-protein with loose budget constraints. (f) Instance j601_1 with loose budget constraints. The results are averaged over 100 runs.

tasks, $q_0 = 0.9$ still performs the best. These results are consistent with the investigation done in the traditional ACS algorithm for TSP [31], [32]. Because the importance of the heuristic information is quite varied in different situations, the configuration for β is somewhat different under different environments. Generally, under tight constraint environment [Fig. 6(a) and (b)], acceptable results can be found when $\beta \in [1.2, 20]$, and $\beta = 20$ is the best choice. In contrast, under medium and loose constraint environment [Fig. 6(c) and (f)], the value of β does not significantly influence the quality of results. The results of cost optimization are shown in Fig. 7. Similarly, $q_0 = 0.9$ is the best configuration for both workflow applications, and the influence of β is not significant. The best results always occurred when $\beta \in [1.1, 1.3]$. Fig. 8 plots the results in the case of reliability optimization. In these two plots, $q_0 = 0.9$ is still the best choice, and β is also important in this case. Good results can be found when $\beta \in [3, 5]$ in the e-protein instance [Fig. 8(a)] and $\beta = 1.2$ in the j601_1 instance [Fig. 8(b)].

Based on previous analysis, we set $q_0 = 0.9$ and $\beta = 1.2$ in our experiment. $q_0 = 0.9$ gives the best results almost in all cases. Although the best configuration for β is inconsistent in different situations, $\beta = 1.2$ generally manages to find acceptable results in all situations.

C. Heuristics of the Algorithm

We propose seven different heuristics in the algorithm, including RG, TG, CG, SD, SB, TC, and OP. Moreover, we propose an adaptive scheme for the selection of heuristics: heuristics are also selected based on pheromone values. We execute experiments to compare the performance of these heuristic schemes and the results are shown in Figs. 9 and 10, and Table II.

In the case of makespan optimization, as has been mentioned before, the TG, CG, SB, and TC heuristics are applied. Additionally, the scheme that ants every time choose one of these heuristics randomly is denoted as RAN. The scheme adopted in this paper that ants select the heuristics based on pheromone values is denoted as PHE. Experimental results are shown in Fig. 9. Under tight budget constraints [Fig. 9(a) and (b)], because the CG and SB heuristics aim at finding low-cost service instances that meet the budget constraints, they manage to find feasible solutions in the very early iterations. However, the SB

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

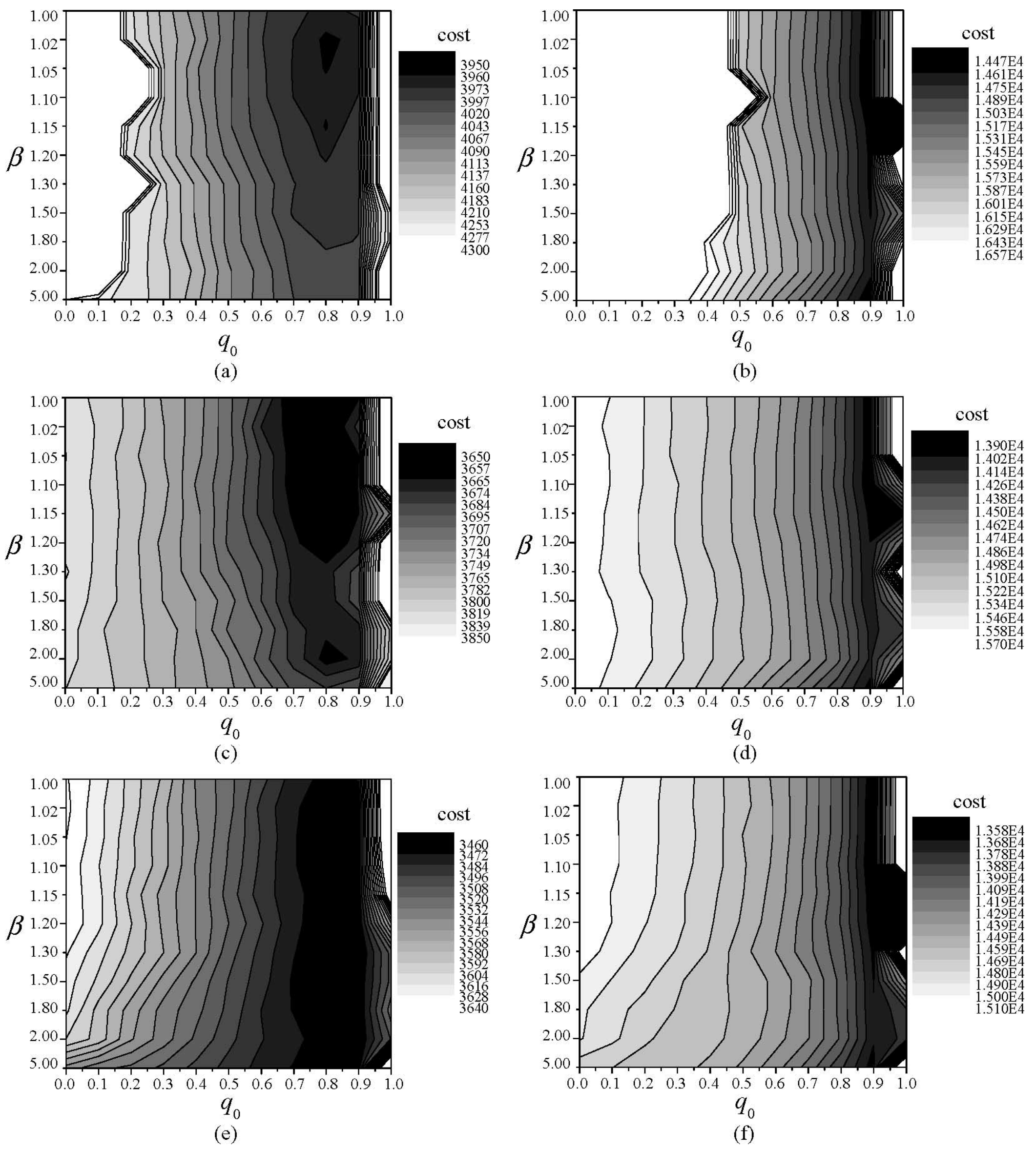


Fig. 7. Performance of different configurations for β and q_0 under different QoS constraints in the case of cost optimization. (a) Instance e-protein with tight deadline constraints. (b) Instance j601_1 with tight deadline constraints. (c) Instance e-protein with medium deadline constraints. (d) Instance j601_1 with medium deadline constraints. (e) Instance e-protein with loose deadline constraints. (f) Instance j601_1 with loose deadline constraints. The results are averaged over 100 runs.

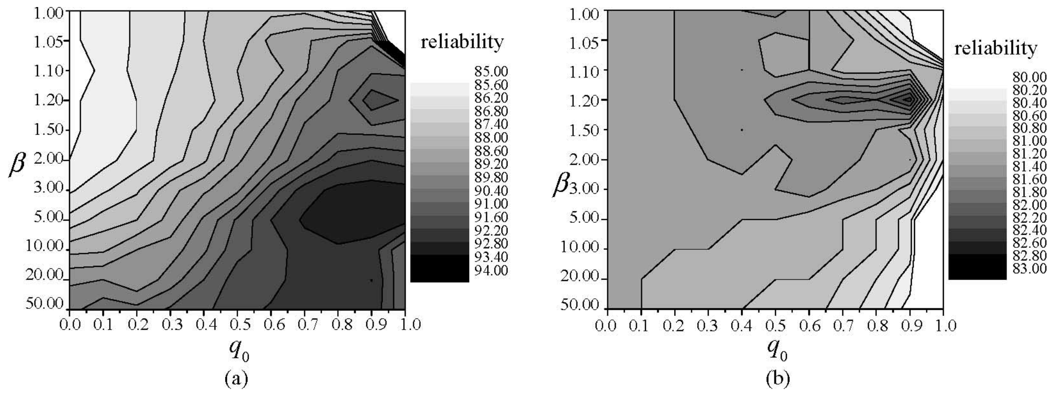


Fig. 8. Performance of different configurations for β and q_0 in the case of reliability optimization. (a) Instance e-protein. (b) Instance j601_1. The results are averaged over 100 runs.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

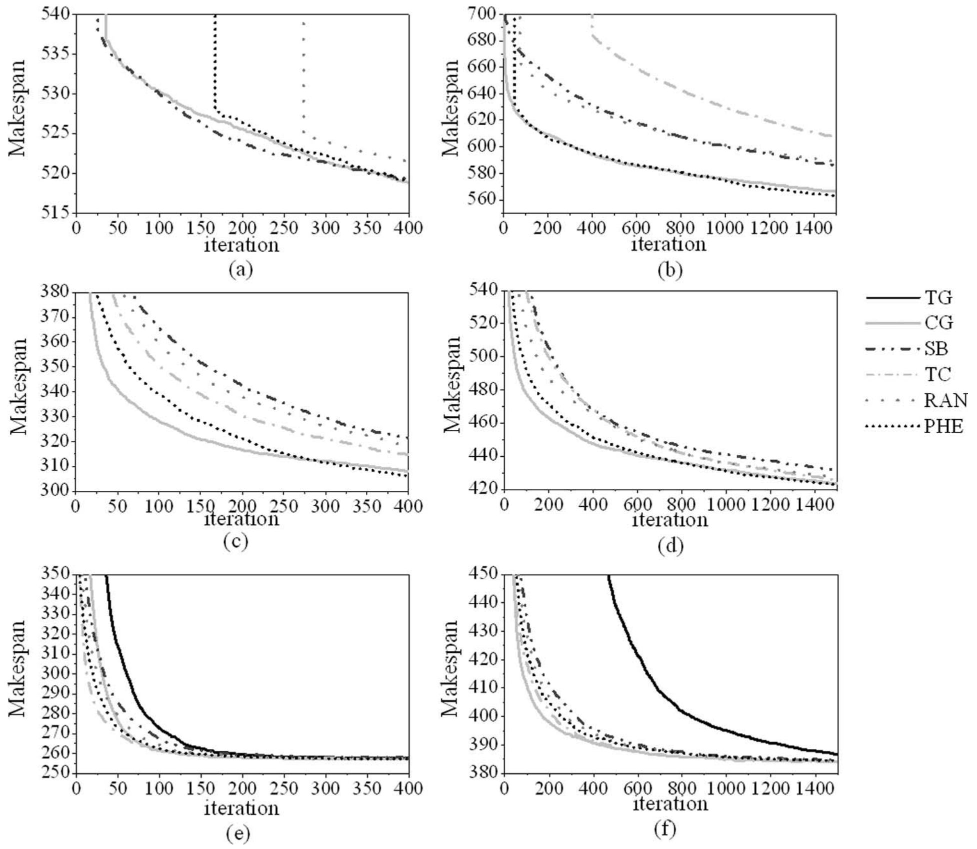


Fig. 9. Performance of different heuristic schemes in the case of makespan optimization. (a) Instance j601_1 with tight budget constraints. (b) Instance j1201_1 with tight budget constraints. (c) Instance j601_1 with medium budget constraints. (d) Instance j1201_1 with medium budget constraints. (e) Instance j601_1 with loose budget constraints. (f) Instance j1201_1 with loose budget constraints. The results are averaged over 100 runs.

heuristic fails to find better solutions in later iterations, and the proposed PHE scheme manages to find slightly better solutions than the CG heuristic finally. Under medium budget constraints [Fig. 9(c) and (d)], similarly, the CG heuristic does well in early phase but the PHE scheme gives the best solution eventually. We also notice that the TG heuristic fails to find feasible solutions in all tests, for it fails to meet the budget constraints every time. Under loose constraints [Fig. 9(e) and (f)], as finding feasible solutions is a simple task, all heuristics are able to find the best solution at last, and the difference between their performance is small, except for the TG heuristic.

In the case of cost optimization, the TG, CG, SD, and TC heuristics are applied. The comparison between all these heuristics and the RAN and PHE scheme is illustrated in Fig. 10. Under tight deadline constraints [Fig. 10(a) and (b)], no feasible solutions can be found by the CG heuristic. The SD heuristic is able to obtain feasible solutions at the very beginning, but the PHE scheme is the best in the end. Under medium [Fig. 10(c) and (d)] or loose [Fig. 10(e) and (f)] constraints, the CG heuristic is able to find feasible solutions, and once a feasible solution is found, its quality is very good. Differently, the performance of the PHE scheme is much steadier, and the ultimate solutions found by the PHE scheme are also slightly better than the ones found by the CG heuristic in most cases.

The performance of different heuristic schemes in the case of reliability optimization is shown in Table II. The results also reveal that the PHE scheme used in this paper outperforms the other heuristic schemes.

Note that both the RAN and the PHE schemes combine all seven heuristics together. However, while the RAN scheme selects heuristics randomly, the PHE scheme is able to make use of the pheromone value of ants to increase the probabilities of selecting better-fit heuristics. Fig. 11 reveals the most attractive heuristic types in different stage of the algorithm. For example, in the case of cost optimization under tight deadline constraints [Fig. 11(a)], at the beginning stage of the algorithm, heuristic D (the SD heuristic) is able to find feasible solutions that satisfy the deadline constraints quickly. So these two heuristics become more attractive to ants. At later stages, the TC heuristic manages to guide the ants to better solutions with lower cost, so that more ants will use the TC heuristic. Similar phenomena occur in the other cases. The experimental results in Figs. 9 and 10 demonstrate that the performance of the PHE scheme is significantly better than the RAN scheme in all cases.

D. Results and Comparisons

As has been mentioned before, few methods are able to tackle the large-scale workflow scheduling problem with different user-defined QoS constraints and QoS preferences. We compare our ACS approach with the deadline-MDP algorithm

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

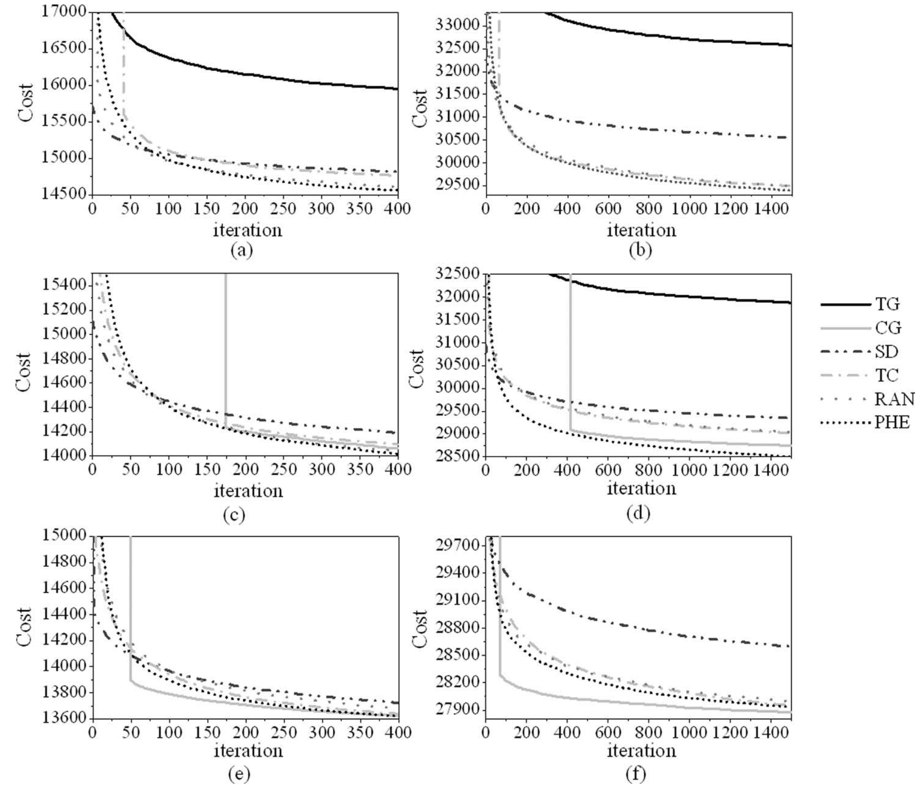


Fig. 10. Performance of different heuristic schemes in the case of cost optimization. (a) Instance j601_1 with tight deadline constraints. (b) Instance j1201_1 with tight deadline constraints. (c) Instance j601_1 with medium deadline constraints. (d) Instance j1201_1 with medium deadline constraints. (e) Instance j601_1 with loose deadline constraints. (f) Instance j1201_1 with loose deadline constraints. The results are averaged over 100 runs.

TABLE II
PERFORMANCE OF HEURISTIC SCHEMES IN RELIABILITY OPTIMIZATION

	j601_1	j1201_1
RG	82.0	N/A
TG	N/A	N/A
CG	80.0	80.0
SD	80.1	80.1
SB	81.1	80.0
TC	80.0	80.0
RTC	80.9	80.1
RAN	82.4	80.1
PHE	82.4	80.1

proposed in [25]. This algorithm works by dividing the DAG into several partitions and distributing the subdeadline to each partition. Moreover, an MDP is applied to find the best solutions for pipeline partition branches in deadline-MDP. The algorithm can only tackle the cost optimization task with deadline constraints.

Experiments are conducted on the ten instances and different levels of QoS constraints are set. Table III shows the results of the experiments. Deadline-MDP is a deterministic algorithm so it only provides one solution. Differently, as the proposed ACS is a random-guided search algorithm, we give the average value (the first line), minimum value (the second line), and maximum value (the third line) over 100 independent runs in the table. The results in the table demonstrate that the average performance of

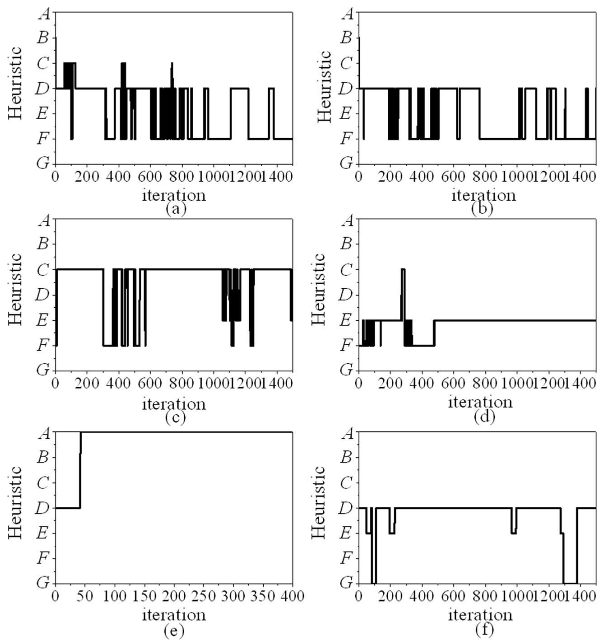


Fig. 11. Heuristic type adopted by most ants. (a) Instance j1201_1 with tight deadline constraints in the case of cost optimization. (b) Instance j1201_1 with loose deadline constraints in the case of cost optimization. (c) Instance j1201_1 with tight budget constraints in the case of makespan optimization. (d) Instance j1201_1 with loose budget constraints in the case of makespan optimization. (e) Instance j601_1 in the case of reliability optimization. (f) Instance j1201_1 in the case of reliability optimization. These results are evaluated based on 100 independent runs.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

TABLE III
COMPARISON BETWEEN THE ACS ALGORITHM AND THE DEADLINE-MDP ALGORITHM

		ACS	MDP	ACS	MDP	ACS	MDP	ACS	MDP	ACS	MDP
e-Economic	constraint	95		115		135		155		175	
	result	2269.1		2122.9		1951.7		1845.8		1739.1	
		2233	2296	2079	2152	1933	2081	1803	1986	1704	1864
		2355		2188		2045		1895		1806	
fMRI	constraint	140		160		180		200		220	
	result	3964.6		3778.8		3628.1		3456.2		3318.3	
		3922	3966	3705	3907	3537	3787	3409	3506	3287	3340
		4031		3916		3711		3683		3403	
e-Protein	constraint	160		180		200		220		240	
	result	3983.6		3809.5		3678.8		3522.2		3460.9	
		3909	4275	3727	4129	3605	3973	3514	3769	3445	3714
		4119		3909		3841		3616		3514	
j301_1	constraint	245		275		305		335		365	
	result	8394.9		7872.5		7499.5		7291.2		7124.22	
		8244	9348	7748	8953	7387	8742	7190	8425	7061	8219
		8606		7983		7727		7426		7203	
j601_1	constraint	280		310		340		370		400	
	result	14561.1		14341.0		14019.3		13817.9		13620.0	
		14341	16531	14173	16134	13843	15849	13642	15621	13466	15805
		14781		14573		14216		14092		13778	
j601_2	constraint	300		340		380		420		460	
	result	15189.2		14521.5		14010.8		13636.9		13338.1	
		14929	16694	14273	15904	13837	15415	13471	15165	13193	14963
		15473		14814		14332		13822		13542	
j901_1	constraint	230		270		310		350		390	
	result	23705.8		22211.1		21132.4		20198.3		19619.1	
		23343	24731	21791	23764	20886	22842	19948	21905	19401	20804
		24465		22260		21491		20519		19903	
j901_2	constraint	365		415		465		515		565	
	result	23183.4		22442.7		21882.9		21432.5		20915.4	
		22926	25629	22163	24614	21599	23763	21186	23098	20699	22389
		23558		22792		22198		21613		21209	
j1201_1	constraint	410		455		500		545		590	
	result	29396.6		28888.1		28499.6		28204.9		27931.6	
		29055	34228	28636	33303	28167	32529	27890	31778	27675	31019
		29756		29168		28769		28473		28268	
j1201_2	constraint	410		470		530		590		650	
	result	30787.4		29625.9		28809.2		28125.8		27696.8	
		30266	34366	29350	33169	28343	31949	27825	31035	27472	29962
		31319		29980		29273		28455		27941	

In the result of the ACS algorithm, the three numbers are the average value (the first line), the minimum value (the second line), and the maximum value (the third line).

the ACS algorithm is better than the deadline-MDP algorithm in all cases. Moreover, in the medium- or large-scale instances (with more than 30 tasks), even the worst result found by ACS is still better than the one generated by deadline-MDP. The ACS algorithm manages to decrease the cost by 10–20% compared with the Deadline-MDP approach. In other words, these results prove the effectiveness of the ACS algorithm.

VI. CONCLUSION

An ACS algorithm for a large-scale workflow scheduling problem in computational grids has been proposed. The scheduling algorithm is designed for workflow applications in market-driven or economy-driven grids under the service-oriented architecture. In the algorithm, different QoS parameters are considered, including reliability, time, and cost. Users are allowed to define QoS constraints to guarantee the quality of the schedule. Moreover, the optimizing objective of the algorithm is based on the user-defined QoS preferences. We propose seven new heuristics for this problem. An adaptive scheme is also designed to enable artificial ants to select heuristics based on pheromone values. We test the algorithm in ten workflow applications with at most 120 tasks. Experimental results demonstrate the effectiveness of the proposed algorithm.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENT

The authors would like to thank the Associate Editor and anonymous reviewers for their constructive suggestions and insightful comments that have helped enrich the content and improve the presentation of this paper.

REFERENCES

[1] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service oriented grid computing," presented at the 10th Heterogeneous Comput. Workshop (HCW' 2001), San Francisco, CA, Apr.

[2] F. Neubauer, A. Hoheisel, and J. Geiler, "Workflow-based grid applications," *Future Gen. Comput. Syst.*, vol. 22, pp. 6–15, 2006.

[3] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*. San Mateo, CA: Morgan Kaufmann, 1999.

[4] D. Kyriazis *et al.*, "An innovative workflow mapping mechanism for grids in the frame of quality of service," *Future Gen. Comput. Syst.*, to be published.

[5] R. Prodan and T. Fahringer, "Overhead analysis of scientific workflows in grid environments," *IEEE Trans. Parallel Distrib. Syst.*, to be published.

[6] Z. Shi and J. J. Dongarra, "Scheduling workflow applications on processors with different capabilities," *Future Gen. Comput. Syst.*, vol. 22, pp. 665–675, 2006.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[8] T. D. Braun *et al.*, "A comparison eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, pp. 810–837, 2001.

[9] H. XiaoShan and S. XiaoHe, "QoS guided min-min heuristic for grid task scheduling," *J. Comput. Sci. Technol.*, vol. 18, no. 4, pp. 442–451, 2003.

[10] A. Mandal *et al.*, "Scheduling strategies for mapping application workflows onto the grid," in *Proc. 14th IEEE Int. Symp. High Perform. Distrib. Comput. (HPDC-14)*, 2005, pp. 125–134.

[11] M. Maheswaran *et al.*, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 59, pp. 107–131, 1999.

[12] M. M. López, E. Heymann, and M. A. Senar, "Analysis of dynamic heuristics for workflow scheduling on grid systems," in *Proc. 5th Int. Symp. Parallel Distrib. Comput. (ISPDC'06), IEEE*, Jul., pp. 199–207.

[13] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[14] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel Distrib. Comput.*, vol. 47, pp. 8–22, 1997.

[15] V. D. Martino and M. Mililotti, "Scheduling in a grid computing environment using genetic algorithms," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS'02), IEEE*, pp. 235–239.

[16] J.-K. Kim, *et al.*, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *J. Parallel Distrib. Comput.*, vol. 67, pp. 154–169, 2007.

[17] S. Zheng, W. Shu, and L. Gao, "Task scheduling using parallel genetic simulated annealing algorithm," in *Proc. IEEE Int. Conf. Service Operations Logist. (SOLI'06)*, pp. 46–50.

[18] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid—An open grid services architecture for distributed systems integration," Open Grid Service Infrastructure WG, Global Grid Forum, Lemont, IL Tech. Rep., Jun. 2002.

[19] A. Afzal, A. S. McGough, and J. Darlington, "Capacity planning and scheduling in grid computing environments," *Future Gen. Comput. Syst.*, to be published.

[20] R. Buyya, D. Abramson, and S. Venugopal, "The grid economic," *Proc. IEEE*, vol. 93, no. 3, pp. 698–714, Mar. 2005.

[21] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *J. Web Semantics*, vol. 1, pp. 281–308, 2004.

[22] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," *Future Gen. Comput. Syst.*, vol. 18, pp. 1061–1074, 2002.

[23] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid," in *Proc. 4th Int. Conf. High-Perform. Comput. Asia-Pacific Region*, 2000, vol. 1, pp. 283–289.

[24] Y. Yuan, X. Li, and Q. Wang, "Time-cost trade-off dynamic scheduling algorithm for workflows in grids," in *Proc. 10th Int. Conf. Comput. Supported Coop. Work Des.*, Nanjing, China, May 2006, pp. 1–6.

[25] R. Buyya and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proc. 1st Int. Conf. e-Sci. Grid Comput. (e-Sci.' 05)*, pp. 140–147.

[26] A. Afzal, J. Darlington, and A. S. McGough, "QoS-constrained stochastic workflow scheduling in enterprise and scientific grids," *Proc. 7th IEEE/ACM Int. Conf. Grid Comput.*, 2006, pp. 1–8.

[27] L. Chunlin and L. Layuan, "QoS based resource scheduling by computational economy in computational grid," *Inf. Process. Lett.*, vol. 98, pp. 119–126, 2006.

[28] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Proc. ECAL91—Eur. Conf. Artif. Life*, New York: Elsevier, 1991, pp. 134–142.

[29] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[30] T. Stützle and H. Hoos, "MAX-MIN ant system and local search for the traveling salesman problem," in *Proc. 1997 IEEE Int. Conf. Evol. Comput. (ICEC'97)*, Piscataway, NJ: IEEE Press, pp. 309–314.

[31] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to TSP," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[32] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[33] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.

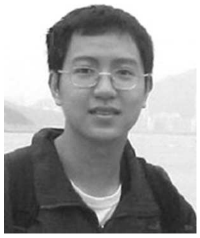
[34] E. Deelman, *et al.*, "Mapping abstract complex workflows onto grid environments," *J. Grid Comput.*, vol. 1, pp. 25–39, 2003.

[35] J. Yu, S. Venugopal, and R. Buyya, "Grid market directory: A web services based grid service publication directory," Grid Comput. Distrib. Syst. Lab. Univ. Melbourne Vic., Australia Tech. Rep., 2003.

[36] Y. Zhao *et al.*, "Grid middleware services for virtual data discovery Composition, and integration," in *Proc. 2nd Workshop Middleware Grid Comput.*, Toronto, ON, Canada, Oct. 18, 2004, pp. 57–62.

[37] A. O'Brien, S. Newhouse, and J. Darlington, "Mapping of scientific workflow within the e-protein project to distributed resources," in *UK e-Sci. All Hands Meet.*, Nottingham, U.K., Sep.2004, pp. 404–409.

[38] R. Kolisch and A. Sprecher, "PSPLIB—A project scheduling problem library: OR Software—ORSEP operations research software exchange program," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1997.



Wei-Neng Chen (S'07) received the Bachelor's degree in network engineering in 2006 from Sun Yat-Sen University, Guangdong, China, where he is currently working toward the Ph.D. degree in the Department of Computer Science.

His current research interests include ant colony optimization, genetic algorithms, and other computational intelligence techniques, and also the applications of project scheduling and financial management.

Jun Zhang (M'02–SM'08) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Kowloon, Hong Kong, in 2002.

From 2003 to 2004, he was a Brain Korean 21 Postdoctoral Fellow in the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology. Since 2004, he has been with Sun Yat-sen University, Guangzhou, China, where he is currently a Professor in the Department of Computer Science. He is the author or coauthor of two books, four research book chapters, and over 60 refereed technical papers. His current research interests include evolutionary algorithms, ant colony system, particle swarm optimization fuzzy logic, neural network, cash flow optimization, workflow optimization, and nonlinear time series analysis and prediction.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on January 6, 2009 at 20:06 from IEEE Xplore. Restrictions apply.