Li, X.; Qian, L.; Ruiz García, R. (2018). Cloud Workflow Scheduling with Deadlines and Time Slot Availability. IEEE Transactions on Services Computing. 11(2):329-340. https://doi.org/10.1109/TSC.2016.2518187

Cloud workflow scheduling with deadlines and time slot availability

Xiaoping Li, *Senior Member, IEEE,* Lihua Qian, Rubén Ruiz, and Qian Wang

Abstract—Allocating service capacities in cloud computing is based on the assumption that they are unlimited and can be used at any time. However, available service capacities change with workload and cannot satisfy users' requests at any time from the cloud provider's perspective because cloud services can be shared by multiple tasks. Cloud service providers provide available time slots for new user's requests based on available capacities. In this paper, we consider workflow scheduling with deadline and time slot availability in cloud computing. An iterated heuristic framework is presented for the problem under study which mainly consists of initial solution construction, improvement, and perturbation. Three initial solution construction strategies, two greedy- and fair-based improvement strategies and a perturbation strategy are proposed. Different strategies in the three phases result in several heuristics. Experimental results show that different initial solution and improvement strategies have different effects on solution qualities.

Index Terms—Workflow, Scheduling, Time slots, Cloud computing.

1 Introduction

Nowadays much attention has been paid on workflow scheduling in service computing environments (cloud computing, grid computing, Web services, etc). Resources are generally provided in the form of services, especially in cloud computing. There are two common ways for service delivery: (i) An entire application as a service, which can be directly used with no change. (ii) Basic services are combined to build complex applications, e.g., Xignite and StrikeIron offer Web services hosted on a cloud on a pay-per-use basis [1]. Among a large number of services in cloud computing, there are many services which have same functions and supplied by different cloud service providers (CSPs). However, these services have different non-functional properties. Basic services are rented by users for their complex applications with various resource requirements which are usually modeled as workflows. Better services imply higher costs. Services are consumed based on Service-Level Agreements, which define parameters of Quality of Service in terms of the pay-per-use policy.

Though there are many parameters or constraints involved in practical workflow scheduling settings, deadline and time slot are two crucial ones in cloud computing, a new market-oriented business model, which offers high quality and low cost information services [2]. However, the two constraints have been considered separately in existing researches. It is necessary to consider both of the constraints jointly because: (i) Deadlines of the workflow applications needs to be met. (ii) Unreserved time slots is crucial for resource utilization from the perspective of service providers. (iii) Utilization of time slots in reserved intervals should be improved to avoid renting new resources (saving money). In this paper, we consider the workflow scheduling problem with deadlines and time slot availability (WSDT for short) in cloud computing. To the best of our knowledge, the considered problem has not been studied yet.

Service capacities are usually regarded to be unlimited in cloud computing, which can be used at any time. However, from the CSP's perspective, service capacities are not unlimited. Available service capacities change with workloads, i.e, they cannot satisfy user's requests at any time when a cloud service is shared by multiple tasks. Only some available time slots are provided for new coming users by CSPs in terms of their remaining capacities. For example, each activity in Figure 1 has different candidate services with various execution times, costs and available time slots. For activity 4, there are two candidate services with different workloads. If service 0 is selected for activity 4, the execution time is 4 with the price 6 and available time slots $[0,4)\bigcup[9,14)$. Time slot $[4,9)$ is unavailable because there is no remaining capacity.

The considered WSDT problem is similar to the the Discrete Time/Cost Trade-off Problem (DTCTP) [3] to some extent. We can modify existing algorithms for the latter to the problem under study with less than 200 activities and no more than 20 candidate services in the service pool, spending thousands of seconds. However, the number of activities is usually far more than 200 in practical workflow applications which makes the modified versions are not suitable for the problem under study.

Generally, longer execution time implies cheaper cost in cloud computing for the DTCTP. However, this is not true for the WSDT. In other words, the fastest schedule of the WSDT does not mean the highest total cost in a pricing model where

- *Xiaoping Li, Lihua Qian and Qian Wang work at the School of Computer Science and Engineering, Southeast University, Nanjing, China, 211189; and also at the Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, 211189, Nanjing, China. Tel: 86-25-52090916; Fax: 86-25- 52090916. E-mail:xpli@seu.edu.cn*
- *Rubén Ruiz works at the Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain. E-mail:rruiz@eio.upv.es*

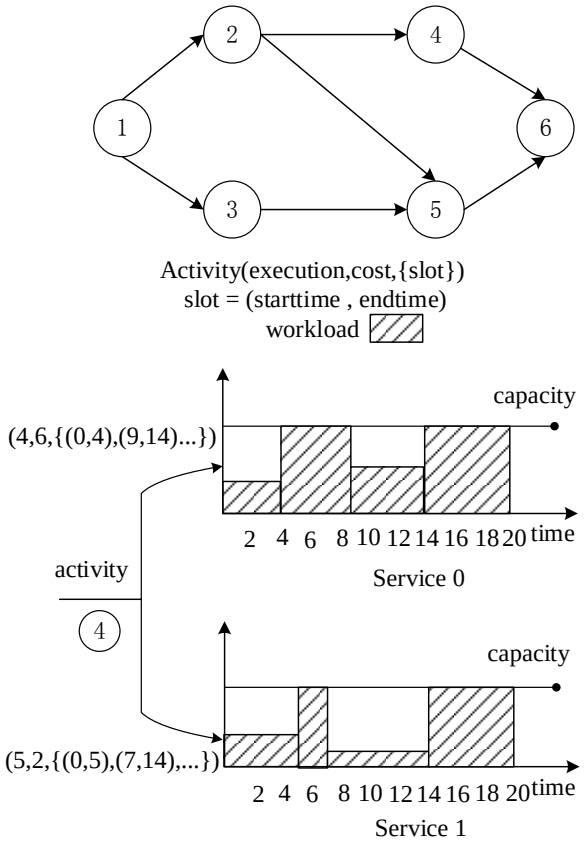

Fig. 1. A workflow example with time slots constraints

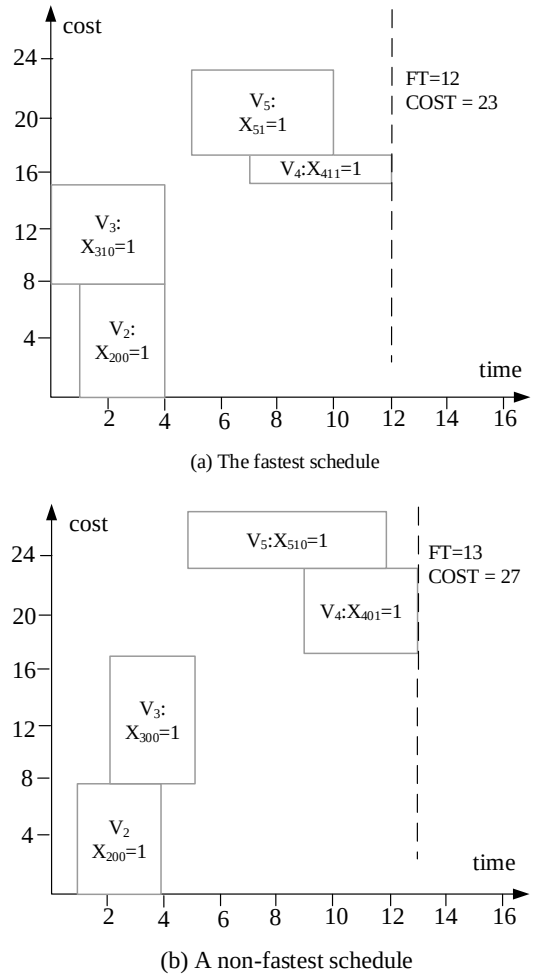

Fig. 2. A fastest schedule against a non-fastest one

the cost is in the inverse proportion to the execution time. For the example depicted in Figure 1 and Table 1, Figure 2 shows the fastest schedule and a non-fastest one where x_{ijk}=1 means the k^{th} available time slot of M_i^j is selected for v_i (M_i^j is the j^{th} available service list of v_i). In the fastest schedule π_1=$\{x_{100}$=1, x_{200}=1, x_{310}=1, x_{411}=1, x_{510}=1, x_{600}=1$\}$, each activity v_i chooses the service to finish as early as possible. The finish time is f_6=12 and the total cost is 21. The non-fastest schedule is π_2=$\{x_{100}$=1, x_{200}=1, x_{300}=1, x_{401}=1, x_{510}=1, x_{600}=1$\}$ with the finish time f_6=13 and the total cost 27. Figure 2 shows that the total cost of the fastest schedule is less than that of the non-fastest one. Therefore, existing methods for the DTCTP cannot be directly adapted to the WSDT. It is necessary to propose new algorithms for the problem under study.

TABLE 1
Activity-Service Pool for the example

Activities	Services (Time,Cost,{Time Slot List})
v_1	(0,0,{[0,+∞)})
v_2	(3,8,{[1,6),[8,10)}),(4,5,{[4,8),[9,12)})
v_3	(3,9,{[2,6),[10,13)}),(4,7,{[0,5),[7,12)})
v_4	(4,6,{[0,4),[9,14)}),(5,2,{[0,5),[7,14)})
v_5	(5,6,{[2,9),[12,15)}),(7,4,{[3,16),[20,28)})
v_6	(0,0,{[0,+∞)})

The main contributions of this paper are summarized below:

- Using mixed integer programming, we mathematically model the cloud workflow scheduling problem with deadlines and time slot availability to minimize total costs from the CSPs perspective.
- An iterated heuristic framework is presented for the problem under study which includes three phases: initial solution construction, improvement, and perturbation.
- Concerning for the different characteristics of the considered WSDT problem from the classical DTCTP, effective and efficient rules for the three phases are presented, based on which several heuristics are constructed.

The rest of the paper is organized as follows. The state of the art is reviewed in Section 2. Section 3 describes the WSDT problem and preparations in detail. Section 4 presents the proposed heuristic framework and developed heuristics. Experimental results are given in Section 5, followed by conclusions and future researches in Section 6.

2 Related Work

Cloud computing is a new market-oriented business model which provides elastic computing and storage resource on demand. Though this kind of computing paradigm becomes increasingly important for computation-intensive tasks in big

data [4]–[7], much attention has been paid on scientific workflows.

Deadline constrained workflow scheduling is one of the most popular scheduling problems in cloud computing [8]–[11]. Appropriate services are selected for all the activities in the involved workflow to maximize objectives. Minimizing time delay, costs and time-cost trade-off are common objectives. Service selection for workflow scheduling considering both time and cost was modeled as the Discrete Time/Cost Trade-off Problem (DTCTP) [3]. Each workflow or project activity can be fulfilled by more than one service, which consists of a service pool. Generally, workflows are represented by Direct Acyclic Graphes (DAGs). Deadlines (or budget limitations) are usual constraints for workflow scheduling. The DTCTP with deadline or budget limitation constraints was called DTCTP-D [12]. The DTCTP-D is harder than the DTCTP and the DTCTP was proven to be NP-hard [13]. It follows that the DTCTP-D is NP-hard as well.

Many exact, heuristic and meta-heuristic algorithms have been developed for the DTCTP-D during the past decades. Exact algorithms, such as dynamic programming [14], the branch and bound [3], benders decomposition [12], were proposed. However, exact algorithms are usually heavily time-consuming for complex workflow instances when the number of activities is more than 200. Based on deadline division, several heuristics were developed for the DTCTP-D in distributed computing environments. Yu et al. [15] proposed the Deadline Markov Decision Process (DMDP) method for utility workflow scheduling. DMDP partitioned the DAG into subgraphs and assigned each partition a sub-deadline according to the minimum execution time of activities and the whole workflow deadline. Yuan et al. [16] presented the Deadline Early Tree (DET) method where a single critical path was constructed to distinguish critical and non-critical activities. Critical activities were scheduled first and the non-critical ones were scheduled according to their priorities. The priorities were determined by floats or slacks (the amount of time that a task could be delayed without causing a delay of both subsequent tasks and the project due date). Abrishami et al. [17] proposed the Partial Critical Paths (PCP) method. All activities were partitioned into different partial critical paths. Each partial critical path was scheduled according to priorities of its activities. They demonstrated that PCP outperforms DMDP and DET. Liu et al. [18] proposed the Path Balanced based Cost Optimization (PBCO) algorithm which adjusted the length of each path in a workflow. A deadline was set for each task. The remaining time was distributed proportionally to tasks according to the number of tasks at each level (the depth of a path node from the source or sink node [15], [16]) using the bottom level strategy. PBCO demonstrated better performance than DET and DBL. Though it seems that there is no existing work on the DTCTP-D, a few metaheuristics were proposed for workflow scheduling in Grid computing ([19], [20]) or only for the DTCTP [21].

Most existing methods for workflow scheduling in cloud computing consider only task constraints (e.g., deadlines) from the perspective of users. Services are rented with an interval-based pricing model. Rented intervals are exclusively reserved and owned by users, i.e., cloud resources (services) are assumed to be unlimited during these intervals. Cai et al. [22] presented service scheduling with start time constraints in distributed collaborative manufacturing systems. They modeled this problem as a Discrete Time-Cost Trade-off Problem with Start Time Constraints (DTCTP-STC) and proved it to be NP-hard. A dynamic programming algorithm was presented for small instances. Cai et al. [23] considered the traditional service selection problem with time/cost trade-off under the workflow deadline constraint. Based on the proposed Critical Path based Iterative heuristic (CPI) in [23], they considered shareable service provisioning for workflows in public clouds in [24]. The List-based Heuristic considering Cost minimization and Match degree maximization heuristic (LHCM) was proposed for maximizing utilization, which was based on several priority rules for assigning tasks to rented services. LHCM guided users to choose proper type and number of shareable services for batch or Message Passing Interface (MPI) tasks. A special type of shareable service provisioning problem was considered in [25], where one task could be executed on only one Virtual Machine (VM) instance. Reserved intervals resulted in time slots for the cloud services from the perspective of resource providers. In addition, there would be some time slots in reserved intervals because the required amount of resources is less than that of the rented resources.

3 Problem formulation and preparation

The WSDT and the DTCTP have different available time slots. In the DTCTP, each service can be used at any time, namely the available time slot for each service is $[0,+\infty)$. To illustrate the different time slots between the WSDT and the DTCTP, a set of candidate services for the activities in Figure 1 are shown in Table 1. v_1 and v_6 are dummy activities. Both of them have only one candidate service with the execution time 0, the cost 0, and the available time slot $[0,+\infty)$. Activity v_2 has two available services M_2^0 and M_2^1. The execution time and cost of M_2^0 are 3 and 8 and those of M_2^1 are 4 and 5. The available time slot of each candidate service of v_2 is $[0,+\infty)$ in the DTCTP while available time slot lists of M_2^0 and M_2^1 are $\{[1,6),[8,10)\}$ and $\{[4,8),[9,12)\}$ respectively in the WSDT. Within the same deadline (e.g., 12), optimal schedules with total cost minimization for the DTCTP and the WSDT are different, as depicted in Figure 3. The optimal schedule's cost of the WSDT is greater than that of the DTCTP because of different available time slots. In addition, the DTCTP can be seen as a special case of the WSDT when the available time slot is $[0,+\infty)$ for each service. The DTCTP was proven to be NP-hard [13]. It is natural that the WSDT is NP-hard.

A workflow application can be represented by a directed acyclic graph $G(V,E)$. $V=\{v_1,v_2,...,v_n\}$ is the set of activities and $E=\{(v_i,v_j)|v_i,v_j\in V\}$ is the set of arcs (precedences between activities). v_1 and v_n are two dummy activities. Each arc $(v_i,v_j)\in E$ $(i\neq j)$ indicates that v_j cannot start until v_i finishes. The service pool $\mathbb{M}_i=\{M_i^0,M_i^1,...,M_i^{\mathcal{N}_i^c-1}\}$ of $v_i\in V$ is composed of $\mathcal{N}_i^c$ candidate services. The services are

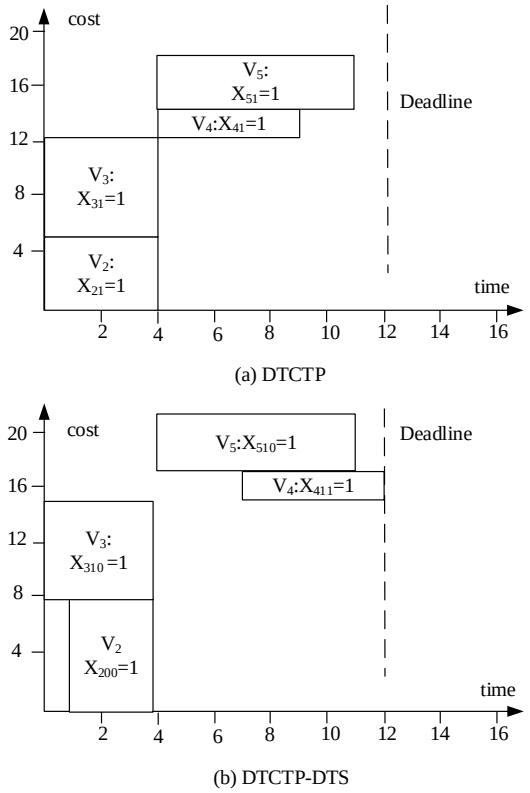

Fig. 3. Optimal schedules of the DTCTP and the WSDT constrained by the deadline of 12 units.

provided by different CSPs. M_i^j is the j^{th} service of v_i. The available time slot list of M_i^j with length $\mathcal{N}_{ij}^s$ is represented as $\mathbb{S}_{ij}$=(S_{ij0},S_{ij1},...,$S_{ij(\mathcal{N}_{ij}^s-1)}$). The k^{th} available time slot S_{ijk} in M_i^j is denoted as $[B_{ijk},F_{ijk})$, where B_{ijk} and F_{ijk} represent the start and end times of S_{ijk} respectively. M_i^j with the execution time e_{ij} and cost c_{ij} is represented by $(\mathbb{S}_{ij},e_{ij},c_{ij})$. f_i denotes the finish time of v_i. D is the given deadline.

Since the problem under study is similar to those in [15]–[18], [23], [24], the mathematical model is constructed using the mixed integer programming. The following binary decision variables are needed:

$$x_{ijk}=\begin{cases}1 & \text{time slot } S_{ijk} \text{ is chosen for activity } v_i\\ 0 & \text{Otherwise}\end{cases}$$

The following assumptions are given for workflow scheduling in cloud computing: (i) Only one service is chosen for each activity from its service pool. Only one available time slot is selected from the corresponding service time slot list. (ii) Each activity cannot be interrupted during execution. (iii) The cost is in inverse proportion to the execution time. (iv) The execution time includes computing and data transmission times. (v) No service is shared by any two activities, i.e., services for different activities are independent. (vi) Service pools of all the activities in workflow instances are constant. The objective is to minimize total costs workflow applications under given deadlines. The problem under study is modeled as follows.

$$\min \sum_{v_i\in V}\sum_{j=1}^{\mathcal{N}_i^c}\sum_{k=1}^{\mathcal{N}_{ij}^s} c_{ij}x_{ijk} \tag{1}$$

s.t.

$$\sum_{j=1}^{\mathcal{N}_i^c}\sum_{k=1}^{\mathcal{N}_{ij}^s} x_{ijk}=1,\ \ v_i\in V \tag{2}$$

$$\sum_{j=1}^{\mathcal{N}_i^c}\sum_{k=1}^{\mathcal{N}_{ij}^s} (B_{ijk}+e_{ij})\cdot x_{ijk}\le f_i,\ \ v_i\in V \tag{3}$$

$$f_i\le \sum_{j=1}^{\mathcal{N}_i^c}\sum_{k=1}^{\mathcal{N}_{ij}^s} (F_{ijk}\cdot x_{ijk}),\ \ v_i\in V \tag{4}$$

$$f_i-f_p\ge \sum_{j=1}^{\mathcal{N}_i^c}\sum_{k=1}^{\mathcal{N}_{ij}^s} (e_{ij}\cdot x_{ijk}),\ \ (v_p,v_i)\in E \tag{5}$$

$$f_i\le D,\ \ i\in V \tag{6}$$

$$x_{ijk}\in\{0,1\},\ \ v_i\in V,\ 0\le j<\mathcal{N}_i^c,\ 0\le k<\mathcal{N}_{ij}^s \tag{7}$$

Exactly one service is assigned to each activity according to constraint (2). The actual start time of each activity cannot be earlier than the start time of the selected time slot according to constraint (3). Constraint (4) guarantees that the actual finish time of each activity cannot later than the finish time of the selected time slot. Constraint (5) controls precedence constraints. The deadline is met according to constraint (6). Constraint (7) defines binary decision variables.

In a workflow application, a schedule is an assignment of services to activities, i.e., each activity v_i selects the k^{th} time slot S_{ijk} in the service list M_i^j or x_{ijk}=1. Because of the time slot constraint for each service, the activity v_i may not start immediately after all of its predecessors complete, i.e., it has to wait for being allocated to an available time slot. We let $\mathcal{P}_i$ and $\mathcal{Q}_i$ represent sets of immediate predecessors and immediate successors of v_i respectively. For each activity v_i, the following temporal parameters are calculated recursively using dynamic programming.

(i) Let $S_{ijk_t^e}$ be the earliest available time slot for v_i in the service list M_i^j at time t. $S_{ijk_t^e}$ is the time slot with $F_{ijk}\ge t$ and $\max\{t,B_{ijk}\}+e_{ij}\le F_{ijk}$, which can be obtained by traversing M_i^j from the head to the tail with time complexity $O(\mathcal{N}_{ij}^s)$. $B_{ij}^A(t)$ denotes the actual available start time if $S_{ijk_t^e}$ is selected, i.e., $B_{ij}^A(t)=\max\{t,B_{ijk_t^e}\}$. $E_{st}(i)$ and $E_{ft}(i)$ are the earliest start and finish times of v_i respectively. Obviously, $E_{st}(1)$=0 and $E_{ft}(1)$=0. For each remaining activity v_i, the two parameters are calculated by the following way with time complexity $O(\mathcal{N}_{ij}^s\times\mathcal{N}_i^c+|\mathcal{P}_i|)$.

$$E_{st}(i)=\min_{j=0,\dots,\mathcal{N}_i^c-1}\left\{B_{ij}^A(\max_{p\in\mathcal{P}_i}\{E_{ft}(p)\})\right\} \tag{8}$$

$$E_{ft}(i)=\min_{j=0,\dots,\mathcal{N}_i^c-1}\left\{B_{ij}^A(\max_{p\in\mathcal{P}_i}\{E_{ft}(p)\})+e_{ij}\right\} \tag{9}$$

(ii) Let $S_{ijk_t^l}$ be the latest available time slot for v_i in the service list M_i^j at time t. $S_{ijk_t^l}$ is the time slot

satisfying $B_{ijk} \leq t$ and $\min\{t, F_{ijk}\} - e_{ij} \geq B_{ijk}$, which can be obtained by traversing M_i^j from the tail to the head with time complexity $O(\mathcal{N}_{ij}^s)$. $F_{ij}^A(t)$ denotes the actual latest finish time if $S_{ijk_t^l}$ is selected, i.e., $F_{ij}^A(t) = \min\{t, F_{ijk_t^l}\}$. $L_{st}(i)$ and $L_{ft}(i)$ are the latest start and finish times of v_i. Obviously, $L_{st}(n) = D$ and $L_{ft}(n) = D$. For each remaining activity v_i, the two parameters can be calculated by the following way with time complexity $O(\mathcal{N}_{ij}^s \times \mathcal{N}_i^c + |\mathcal{P}_i|)$.

$$L_{ft}(i) = \max_{j=0,\ldots,\mathcal{N}_i^c - 1} \left\{ F_{ij}^A(\min_{s \in \mathcal{Q}_i} \{L_{st}(s)\}) \right\} \quad (10)$$

$$L_{st}(i) = \max_{j=0,\ldots,\mathcal{N}_i^c - 1} \left\{ F_{ij}^A(\min_{s \in \mathcal{Q}_i} \{L_{st}(s)\}) - e_{ij} \right\} \quad (11)$$

For simplicity, a solution is denoted as $\pi = \{(i, M_i^j) | x_{ijk^*} = 1\}$ in this paper, in which $x_{ijk^*} = 1$ means the earliest available time slot S_{ijk^*} in M_i^j. A schedule is feasible if and only if $E_{ft}(i) \leq f_i \leq L_{ft}(i)$ for each activity v_i. Because services are dynamically allocated in cloud computing, an extreme case is that the problem under study is unsolvable when no assignment of services to activities would result in $E_{ft}(n) \leq D$, i.e., any assignment leads to $E_{ft}(n) \geq D$.

4 Proposed Algorithms

The service assignment for each activity in the WSDT depends on both finish times of all predecessors and available time slots of the service. In this paper, an Iterated Local Adjusting Heuristic framework (ILAH) is proposed for the problem under study. ILAH consists of four components: Time Slot Filtering, Initial Solution Construction, Solution Improvement and Perturbation. ILAH starts from an initial solution π. Improving and perturbing operations are performed on π iteratively until the termination criterion is satisfied. The termination criterion is set as α, the number of consecutive iterations without improvement. Let $C(\pi)$ be the total cost of π. The high level procedure of ILAH is described in Algorithm 1.

Algorithm 1: ILAH

```
1  begin
2      Time Slot Filtering;
3      Generate the initial solution π by an initial solution
       construction strategy;
4      π_best ← π, C(π_best) ← C(π);
5      while (termination criterion not met) do
6          π ← Improve(π);
7          if (C(π_best) > C(π)) then
8              π_best ← π, C(π_best) ← C(π);
9          Perturbation(π);
10     return π_best.
```

For the last three components, we present three initial solution construction strategies, two improvement methods and one perturbation strategy.

4.1 Time Slot Filtering

Though there are many available time slots, not all of them meet requirements of activities of workflow instances. Some available time slots might not be available for an activity v_i even before the service assignment. For example, $E_{ft}(n) > D$ in the fastest schedule, or the duration of a time slot is less than the execution time of the activity, or the start or finish time is beyond the earliest start or the latest finish time of the activity. By filtering out all impossible time slots, remaining time slots are eligible for activities of the instance, which make workflow scheduling much more efficient. The Time Slot Filtering procedure is given in Algorithm 2.

Algorithm 2: Time Slot Filtering

```
1  begin
2      for (each v_i ∈ V) do
3          Calculate E_st(i), E_ft(i), L_ft(i), L_st(i) using
           equations (8), (9), (10), (11);
4          if (E_ft(n) > D) then
5              return NULL;
               /* infeasible problem          */
6      for (each v_i ∈ V) do
7          for (each service M_i^j ∈ M_i) do
8              for k = 0 to N_ij^s − 1 do
9                  if F_ijk − B_{i,j,k} < e_ik or B_{i,j,k} > D or
                   B_ijk > L_ft(i) or F_ijk < E_st(i) then
10                     Remove s_ijk from S_ij;
11             if (N_ij^s = 0) then
12                 Remove M_i^j from M_i;
13         for (each v_i ∈ V) do
14             Generate the service pool M_i by sorting all
               candidate services in non-increasing order of
               costs;
15     return {M_i}.
```

The time complexity of Steps 2∼5 is $O(n \times (\mathcal{N}_{ij}^s \times \mathcal{N}_i^c + |\mathcal{P}_i|))$, that of Steps 6∼12 is $O(n \times \mathcal{N}_{ij}^s \times \mathcal{N}_i^c)$, and that of Steps 13∼14 is $O(n \times \mathcal{N}_i^c \times \ln \mathcal{N}_i^c)$. Therefore, the time complexity of the Time Slot Filtering procedure is $O(n \times ((\mathcal{N}_{ij}^s + \ln \mathcal{N}_i^c) \times \mathcal{N}_i^c + |\mathcal{P}_i|))$.

4.2 Initial solution construction strategies

Start and finish time parameters along with available time slots are dynamically changed once an activity is assigned to a service. Different service assignments to activities result in different initial solutions. In this paper, we present three priority rules, based on which three initial solution construction strategies are developed. The Minimum Average Cost First (MACF) method focuses on the cost of an activity and those of its immediate successors, i.e., costs among activities. The Maximum Cost Ascending Ratio First (MCARF) strategy considers the two services with the lowest costs for of same activity. The Earliest finish time first (EFTF) rule

takes into account the earliest finish times of all activities. We let b_{ij} and f_{ij} be the start and finish times of activity v_i by selecting service M_i^j. Service M_i^j is available only when $b_{ij}\in[E_{st}(i),L_{st}(i)]$ and $f_{ij}\in[E_{ft}(i),L_{ft}(i)]$. Otherwise, service M_i^j is unavailable.

4.2.1 Minimum Average Cost First (MACF)

MACF generates initial solutions according to the minimum average cost on an activity and its immediate successors. A smaller average cost implies a higher priority of activity choosing a service. After an available service M_i^j is selected for v_i, $f_{ij}=B_{ij}^A(E_{st}(i))+e_{ij}$ and the earliest start times of all immediate successors of v_i are updated by $E_{st}(q)=\max\{f_{ij},E_{st}(q)\}$ $(\forall q\in\mathcal{Q}_i)$. Let $M_q^{s_j}$ be the cheapest available service for activity v_q when M_i^j is selected for v_i. $\overline{W}(i,j)$ denotes the average cost on activity v_i and all of its successors and $\overline{W}_i^*$ represents the Minimum Average Cost $\overline{W}(i,j)$ among all available services of v_i. $\overline{W}(i,j)$ and $\overline{W}_i^*$ are calculated by

$$\overline{W}(i,j)=\Big(c_{ij}+\sum_{q\in\mathcal{Q}_i}c_{qs_j}\Big)/(|\mathcal{Q}_i|+1) \tag{12}$$

$$\overline{W}_i^*=\min_{j=0,\ldots,\mathcal{N}_i^c-1}\Big\{\overline{W}(i,j)\Big\} \tag{13}$$

Let S and U be sets of scheduled and unscheduled activities, which are initialized as $S=\{1,n\}$ and $U=\{2,...,n-1\}$ respectively. Services M_1^0 and M_n^0 are assigned to the two dummy activities v_1 and v_n respectively. $E_{st}(1)=0$ and $E_{ft}(1)=0$. The four temporal parameters $E_{st}(i)$, $E_{ft}(i)$, $L_{st}(i)$ and $L_{ft}(i)$ $(\forall v_i\in V)$ calculated in the Time Slot Filtering are used for further computation.

For each $i\in U$, $\overline{W}_i^*$ is calculated by Equations (12) and (13). Since smaller $\overline{W}_i^*$ implied higher priority of v_i, the list $\mathcal{L}^U$ is constructed by sorting all values of $\overline{W}_i^*$ $(i\in U)$ in non-decreasing order, denotes as $\mathcal{L}^U=\Big(\overline{W}_{[1]}^*,\ \overline{W}_{[2]}^*,\ \ldots,\ \overline{W}_{[|U|]}^*\Big)$, i.e., $\overline{W}_{[1]}^*\le\overline{W}_{[2]}^*\le\ldots\le\overline{W}_{[|U|]}^*$. The highest priority activity $v_{[1]}$ in $\mathcal{L}^U$ is removed from U and appended to S. $v_{[1]}$ is assigned to the available service $M_{[1]}^{j^*}$, in which $j^*=\arg\{\overline{W}_{[1]}^*\}$. We check the position of the successor q_j^* of $v_{[1]}$ with the biggest Minimum Average Cost (i.e., $\overline{W}_{q_j^*}^*=\max_{q\in\mathcal{Q}_{[1]}}\{\overline{W}_q^*\}$). If the position of q_j^* is greater than $\alpha\times|U|$ $(\alpha\in[0,1])$, q_j^* is removed from U and appended to S. q_j^* is assigned to its cheapest available service. After an activity is appended to S or assigned a service, the four temporal parameters of each activity v_i in the workflow instance are recalculated as follows: (i) Calculate $E_{st}(i)$ and $E_{ft}(i)$ by assigning the available time slots with the earliest finish times of the selected services to scheduled activities, assigning the earliest available services to unscheduled activities. (ii) Calculate $L_{st}(i)$ and $L_{ft}(i)$ by assigning the available time slots with the latest finish times of the selected services to scheduled activities, assigning the latest available services to unscheduled activities.

In terms of newly obtained parameters, $\mathcal{L}^U$ is updated. The process is repeated until all activities are scheduled. MACF is formally described in Algorithm 3. We obtain that the time complexity of MACF is $O(n\mathcal{N}_i^c+n^2\ln n)$, which is mainly related to that of calculating $\overline{W}_i^*$ using Equation (13) and that of the sorting all $\overline{W}_i^*$ $(i\in U)$.

Algorithm 3: Minimum Average Cost First (MACF)

Input: Temporal parameter sets E_{st}, E_{ft}, L_{st}, and L_{ft} for all activities of the considered workflow application calculated by the Time Slot Filtering.

begin
  $S\leftarrow\{1,n\}$, $U\leftarrow\{2,...,n-1\}$, $M\leftarrow\varnothing$, $\mathcal{L}^U\leftarrow\varnothing$;
  $\pi\leftarrow\{(1,M_1^0),(n,M_n^0)\}$;
  while *(*$|U|>0$*)* **do**
    for *each* $i\in U$ **do**
      Calculate $\overline{W}_i^*$ according to Equation (13);
      Record the service M_i^j corresponding to $\overline{W}_i^*$;
      $M\leftarrow M\cup\{M_i^j\}$;
    Sort all $\overline{W}_i^*$ $(i\in U)$ in non-decreasing order, i.e., $\overline{W}_{[1]}^*\le\overline{W}_{[2]}^*\le\ldots\le\overline{W}_{[|U|]}^*$;
    $\mathcal{L}^U\leftarrow\Big(\overline{W}_{[1]}^*,\ \overline{W}_{[2]}^*,\ \ldots,\ \overline{W}_{[|U|]}^*\Big)$;
    $s\leftarrow\arg\max_{q\in\mathcal{Q}_{[1]}}\Big\{\overline{W}_q^*\Big\}$; `/* The immediate successor of` $v_{[1]}$ `with the biggest Minimum Average Cost */`
    Select the available service $M_{[1]}^{j^*}$ corresponding to $v_{[1]}$ from M;
    $\pi\leftarrow\pi\cup\{(v_{[1]},M_{[1]}^{j^*})\}$;
    $U\leftarrow U/\arg(v_{[1]})$, $S\leftarrow S\cup\{\arg(v_{[1]})\}$;
    $k\leftarrow\arg_{v_j=s}(\overline{W}_{[j]}^*)$;
    `/*` k `the position of` s `in` $\mathcal{L}^U$. `*/`
    if *(*$k\ge|U|/2$*)* **then**
      Select the available service $M_{[k]}^{j'}$ corresponding to the activity s from M;
      $\pi\leftarrow\pi\cup\{(s,\ M_{[k]}^{j'})\}$;
      $U\leftarrow U/\arg(v_{[k]})$, $S\leftarrow S\cup\{\arg(v_{[k]})\}$;
    for *each* $i\in U$ **do**
      Update $E_{ft}(i)$, $E_{st}(i)$, $L_{ft}(i)$, and $L_{st}(i)$;
  return π.

4.2.2 Maximum Cost Ascending Ratio First (MCARF)

Let M_i^j and M_i^k be the cheapest and the second cheapest available services for v_i, $T_{\min}$ is the total cost of all activities selecting the cheapest services without considering availability of time slots. M_i^k might not exist (i.e., $c_{ik}=\infty$), for which a binary variable is defined as

$$y_i=\begin{cases}1 & \text{if } M_i^k \text{ exists or } c_{ik}<\infty\\ 0 & \text{otherwise}\end{cases}$$

The cost ascending ratio R_i^I of a feasible service for activity v_i is a measure calculated as

$$R_i^I=y_i\frac{(c_{ik}-c_{ij})c_{ij}}{c_{ik}}+(1-y_i)\frac{c_{ij}(n-2)}{T_{min}} \tag{14}$$

A bigger R_i^I implies a higher additional cost if activity v_i is not assigned to the cheapest available service M_i^j. The Maximum Cost Ascending Ratio First (MCARF) strategy

chooses the service with the highest cost ascending ratio R_i^I for v_i, i.e., MCARF first assigns an appropriate service to the activity v_i which has the highest cost ascending ratio R_i^I. A faster cost increase of an activity means a higher priority for choosing its service. MCARF is similar to MACF, which is formally described in Algorithm 4. Similar to the analyzing process of Algorithm 3, the time complexity of Algorithm 4 is $O(n^2 \ln n)$.

Algorithm 4: Maximum Cost Ascending Ratio First (MCARF)

Input: Temporal parameter sets E_{st}, E_{ft}, L_{st}, and L_{ft} for all activities of the considered workflow application calculated by the Time Slot Filtering.

begin
  $S \leftarrow \{1,n\}$, $U \leftarrow \{2,...,n-1\}$, $M \leftarrow \varnothing$;
  $\pi \leftarrow \{(1,M_1^0),(n,M_n^0)\}$;
  while $(|U|>0)$ **do**
    for $i=0$ **to** $|U|-1$ **do**
      Calculate R_i^I of activity v_i according to Equation (14);
      Record the service M_i^j corresponding to R_i^I;
      $M \leftarrow M \cup \{(i,M_i^j)\}$;
    $k \leftarrow \text{argmax}_{i \in U}\{R_i^I\}$;
    $\pi \leftarrow \pi \cup \{(k,M_k^{j'})\}$;
    /* The element $(k,M_k^{j'}) \in M$ */
    $U \leftarrow U/\{k\}$, $S \leftarrow S \cup \{k\}$;
    for *each* $i \in U$ **do**
      Update $E_{ft}(i)$, $E_{st}(i)$ and $L_{ft}(i)$, $L_{st}(i)$;
  return π.

4.2.3 Earliest finish time first (EFTF)

The Earliest finish time first (EFTF) rule constructs initial solutions according to priorities of activities. A smaller $E_{ft}(i)$ means a higher priority for v_i to be assigned a service, i.e., choosing the service time slot with the earliest finish time can make activity v_i finish as early as possible. Obviously, if the scheduling problem is solvable, π must be feasible.

4.3 Improvement strategies

Let $e_{st}(i)$, $l_{st}(i)$ and $l_{ft}(i)$ be the earliest start time, the latest start time, and the latest finish time of activity v_i respectively in the current solution π. M_i^j is the service assigned to v_i in π. There are some adjustable activities (non-critical activities) in π, i.e., the earliest start time and the latest start time of each activity are not the same when there are at least two available services for each adjustable activity. In other words, v_i is adjustable if and only if $e_{st}(i)<l_{st}(i)$ and there is at least another available service $M_i^{j'}$ which is different from M_i^j. The total cost of the workflow instance can be further optimized by assigning different services to some adjustable activities while keeping the assignment of their predecessors unchanged. In this paper, we develop two improvement heuristics to exhibit these ideas. Both of them are backwards adjusting heuristics, i.e., they search adjustable activities from the last activity (sink node) to the first activity (source node) in a topological order.

4.3.1 Greedy Improvement Heuristic (GIH)

After an adjustable activity v_i is found, we compute the Cost Decreasing Ratio $R_{ijj'}^D$ between M_i^j and each cheaper service $M_i^{j'}$ by

$$R_{ijj'}^D = (c_{ij} - c_{ij'})/(e_{ij'} - e_{ij}) \tag{15}$$

The available service $M_i^{j^*}$ with the maximum $R_{ijj^*}^D$ in $[e_{st}(i),\ l_{ft}(i)]$ is selected to substitute M_i^j, i.e., $R_{ijj^*}^D = \max\{R_{ijj'}^D\}$. Algorithm 5 gives the detailed description. Let the degree of activity v_i be d_i. All successors and predecessors of the n activities are visited just once when calculating the latest finish times and the earliest start times, i.e., there are $\sum_{i=1}^n d_i$ activities being visited. Therefore, the time complexity of Algorithm 5 is $O(\sum_{i=1}^n d_i)$.

Algorithm 5: Greedy Improvement Heuristic (GIH)

Input: Initial solution π, the earliest start time of each activity v_i according to π, $est=\{e_{st}(i)|i=1,...,n\}$.

begin
  $l_{ft}(n) \leftarrow D$, $l_{st}(n) \leftarrow D$, $cost \leftarrow 0$;
  for $i=n-1$ **to** 1 **do**
    Calculate the latest finish time of v_i by $l_{ft}(i) \leftarrow \min\limits_{q \in \mathcal{Q}_i}\left\{l_{st}(q)\right\}$;
    Calculate the earliest start time of v_i by $e_{st}(i) \leftarrow \max\limits_{p_i \in \mathcal{P}_i}\left\{e_{ft}(p_i)\right\}$;
    Compute $R_{ijj'}^D$ for all candidate available services in $[e_{st}(i),l_{ft}(i)]$ using Equation (15);
    Select the available service $M_i^{j^*}$ with $R_{ijj^*}^D = \max\{R_{ijj'}^D\}$ to replace the current service M_i^j;
    Update the element (i,M_i^j) of π with $(i,M_i^{j^*})$;
    Update $l_{st}(i)$ according to $M_i^{j^*}$;
  return π.

4.3.2 Fair Improvement Heuristic (FIH)

The GIH tries to decrease the cost by assigning a cheaper service that has the maximum Cost Decrease Ratio for each v_i. Though the GIH can reduce the total cost, it might reduce the number of cheaper available services of its predecessors and resulted in inferior solutions. For this reason, we presented the Fair Improvement Heuristic (FIH) rule.

Instead of substituting M_i^j of adjustable activity v_i with the cheapest available service with the maximum Cost Decrease Ratio, the FIH simply selects the second cheapest available service in $[e_{st}(i),l_{ft}(i)]$ for the substitution. The process is iterated until there is no substitution in an iteration. The FIH is formally described in Algorithm 6. Similar to the analysis process of the GIH, the time complexity of Algorithm 6 is also $O(\sum_{i=1}^n d_i)$.

Algorithm 6: Fair Improvement Heuristic (FIH)

Input: Initial solution π, the earliest start time of each activity v_i according to π, est=$\{e_{st}(i)|i$=1,...,$n\}$.

begin
  $l_{ft}(n)\leftarrow D$, $l_{st}(n)\leftarrow D$, $cost\leftarrow 0$, $flag\leftarrow TRUE$;
  while $flag$ **do**
    $flag\leftarrow FALSE$;
    for i=n−1 **to** 1 **do**
      Calculate the latest finish time of v_i by $l_{ft}(i)\leftarrow \min_{q\in\mathcal{Q}_i}\left\{l_{st}(q)\right\}$;
      Calculate the earliest start time of v_i by $e_{st}(i)\leftarrow \max_{p_i\in\mathcal{P}_i}\left\{e_{ft}(p_i)\right\}$;
      Substitute M_i^j with the second cheapest available service $M_i^{j'}$ in [$e_{st}(i)$,$l_{ft}(i)$] for v_i;
      Update the element (i,M_i^j) of π with $(i,M_i^{j'})$;
      Update $l_{st}(i)$ according to $M_i^{j'}$;
      $flag\leftarrow TRUE$;
  return π.

4.4 Perturbation

The ILAH is likely to get trapped into local optima after some iterations. To enhance the diversification of the search process, a Perturbation method is introduced. Starting from a solution π, $\lfloor n\beta\rfloor$ ($0<\beta<1$) activities are randomly selected and their current service assignment are changed by more expensive services, which can result in bigger adjustment intervals. The new solution is then improved by the FIH or the GIH with the hope of finding a better solution than before.

Substituting the current service M_i^j of v_i with a more expensive available service, $M_i^{j'}$ might lead to changes in the start and finish times of predecessors and successors. Let ℓ_p and ℓ_s be the increased slack or float time of immediate predecessors and successors of v_i respectively with the substitution. ℓ_p=$st_{ij'}-st_{ij}$, where $st_{ij'}$ is the start time of v_i with service $M_i^{j'}$ and st_{ij} is the start time of v_i with the current service M_i^j. ℓ_s=$ft_{ij'}-ft_{ij}$, where $ft_{ij'}$ is the finish time of v_i with service $M_i^{j'}$ and ft_{ij} is the finish time of v_i with the current service M_i^j. If the service substitution delays the start time of v_i ($\ell_p>0$), more float time will be available for all of its immediate predecessors. Otherwise, no more float (ℓ_p=0) is available for its immediate predecessors. If the finish time $ft_{ij'}$ with service $M_i^{j'}$ becomes smaller than that ft_{ij} with the current service M_i^j, i.e., $\ell_s>0$, more float time will be available for all of its immediate successors. Otherwise, no more float (ℓ_s=0) is available for its immediate successors. The cost increase ratio per float unit $R_{ijj'}^F$ of substituting M_i^j with $M_i^{j'}$ for v_i is defined by

$$R_{ijj'}^F=\begin{cases}+\infty & \ell_p=0 \text{ and } \ell_s=0 \text{ or no such } M_i^{j'}\\ \frac{c_{ij'}-c_{ij}}{|\mathcal{P}_i|\times\ell_p+|\mathcal{Q}_i|\times\ell_s} & \text{Otherwise}\end{cases} \quad (16)$$

A smaller R_{ij}^F=$\min\{R_{ijj'}^F\}$ means a less cost increasing ratio per float unit and v_i has a higher priority to replace its current service with a more expensive one. To avoid repeating substitution on the same activity with the same available service, we perform replacement on the activity with the m^{th} smallest R_{ij}^F if there is no improvement in consecutive m iterations ($m\leq\alpha$). Perturbation results in the change of floats of other activities. Therefore, all R_{ij}^F are updated after each substitution. Perturbation repeats the substitution until $\lfloor n\beta\rfloor$ activities are replaced. The process is shown in Algorithm 7.

Algorithm 7: Perturbation

Input: Initial solution π, m, $M\leftarrow\varnothing$.

begin
  $count\leftarrow 0$, $U\leftarrow\{1,...,n\}$;
  while ($count<\lfloor n\beta\rfloor$) **do**
    for *each* $i\in U$ **do**
      Calculate R_{ij}^F=$\min\{R_{ijj'}^F\}$ of activity v_i according to Equation (16);
      Record the service $M_i^{j^*}$ corresponding to R_{ij}^F to M, i.e., $M\leftarrow M\cup\{(i,M_i^{j^*})\}$;
    Sort all activities in non-decreasing order of R_{ij}^F;
    Select the activity v_i with the m^{th} smallest R_{ij}^F and replace M_i^j with the corresponding available service;
    Update the element related to v_i in π;
    $U\leftarrow U/i$;
    $count\leftarrow count+1$;
  return π.

5 Experimental Analysis

There are several variants for each component or parameter of the proposed algorithm framework. We calibrate components and parameters first, based on which three heuristics are compared. All involved algorithms are coded in Java and run on Intel(R) Core(TM) i7-4770 CPU @3.40GHz with 8GB RAM on Windows Server 2008 R2 standard. The termination criterion is set as the maximum computation time $\frac{n^2}{t}$ ($t\in\{5,10,20\}$) milliseconds.

5.1 Parameter and Component Calibration

The termination condition parameter t=5, i.e., the computation time is limited to $\frac{n^2}{5}$ milliseconds for all combinations.

WSDT calibration instances are randomly generated. Seven instance factors result in different workflows. These are: N, M, OS, CF, CP, $Loadnum$ and DF. $N\in\{200,400,600,800,1000\}$ is the number of activities in a workflow. M is the size of service pool, which takes a uniformly random value from {[2,10], [11,20], [21,30]}. The complexity of network structures is measured by $OS\in\{0.1,0.2,0.3\}$ according to [26]. In terms of [12], the cost function $CF\in\{$*convex*,*concave*,*hybrid*$\}$. We let $MinMakespan$ be the minimal makespan without time slots constraints (all activities select the shortest durations). $CP\in\{3,4,5\}$ is the

longest time horizon factor of services which determines the longest time horizon of service TH=$CP\times MinMakespan$. $LoadNum\in\{0,1,2\}$ is the total length of unavailable time slot to $TH\times 10$. $DF\in\{0.2,0.4,0.6\}$ is the deadline factor. D_{min} is the makespan of fastest schedule. The deadline $D=D_{min}+(TH-D_{min})\times DF$. Therefore, there are $5\times3\times3\times3\times3\times3\times3$ = 3645 instance combinations. One calibration instance is randomly generated for each combination, i.e., 3645 instances are test to calibrate the components and parameters.

Though there are three initial solution generation strategies (EFTF, MACF and MCARF), there is a parameter α in MACF. In this paper, we test all values of $\alpha\in\{0.25,0.5,0.75,1\}$, i.e., four MACFs are tested. By replacing Equation (14) of MCARF with $R_i^I=y_i(\frac{c_{ij}}{c_{ik}})^2+(1-y_i)\frac{c_{ij}(n-2)}{T_{min}}$, a new rule MCARF_0 is obtained. In addition, the RANDOM strategy is involved, which generates initial solutions randomly. In other words, we calibrate eight variants of the initial solution construction component. Three variants of the improvement component are evaluated: GIH, FIH and NONE (no improvement strategy). β takes values from {0, 0.2, 0.4, 0.6, 0.8, 1}, i.e., 6 candidates for β are considered. There are two operations for time slots: filtering and non-filtering. There are $8\times3\times6\times2$=288 combinations of components and parameters. For each combination, five replicates are obtained. Totally, $3645\times288\times5$=5,248,800 results are obtained in the calibration.

Experimental results are analyzed by the multi-factor analysis of variance (ANOVA) technique [27]. A number of hypotheses have to be ideally met by the experimental data. The main three hypotheses (in order of importance) are the independence of the residuals, homoscedasticity or homogeneity of the factor's level variance and normality in the residuals of the model. Apart from a slight non-normality in the residuals, all the hypotheses are easily accepted. The response variable in the experiments is RPD (Relative Percentage Deviation) for each algorithm in every instance. Let $V_i(H)$ be the solution of instance i obtained by algorithm H and V_i^* be the optimum solution for i. The RPD is defined as

$$RPD=\frac{V_i(H)-V_i^*}{V_i^*}\times100\% \tag{17}$$

Figure 4 shows the means plot of the perturbation parameter β with different values and 95.0% Tukey HSD intervals. Figure 4 means that differences between any pair of values are not statistically significant, especially when $\beta>0$. The RPD is the smallest when β=0.4 and it is the largest when β=0. There is a statistically significant difference between the two cases. However, the difference is not large. β takes 0.4 in the following experiments.

Figure 5 shows the means plot of improvement strategies and 95.0% Tukey HSD intervals. Figure 5 demonstrates that there is no statistically significant difference between the FIH and the GIH. The NONE has the largest RPD which implies that the FIH and the GIH are effective improvement strategies. We choose both FIH and GIH as improvement strategies in the following experiments.

Figure 6 shows the means plot of initial solution generation strategies and 95.0% Tukey HSD intervals. Figure 6 implies that differences between any pair of the four types of initial solution generation strategies (EFTF, MACF, MCARF and RANDOM) are significant. RPDs of the MACF with different parameters have no statistically significant differences. MCARF and MCARF_0 have the lowest RPD. RPDs of MCARF and MCARF_0 are slightly less than those of MACFs but much smaller than those of the EFTF and the RANDOM, which indicates that MCARF and MCARF_0 are effective for the problem under study. However, it is strange that the EFTF has the largest RPD, even worse than RANDOM. The main reason lies in that the EFTF chooses service time slots with the earliest finish times which always results in poor initial solutions. Both the MCARF and the EFTF are selected as initial solution generation strategies for the proposed algorithm in the following comparing experiments.

Figure 7 shows interactions between the eight initial solution construction strategies and the three improvement strategies with 95.0% Tukey HSD intervals. From Figure 7, we can observe that combinations between MACFs and the three improvement strategies and those between MCARFs and the three improvement strategies have similar RPDs. It is strange that the combinations between two improvement strategies (FIH and GIG) and RANDOM have nearly the best RPDs which demonstrate that FIH and GIG are effective improvement strategies except for the EFTF.

5.2 Algorithm Comparison

To the best of our knowledge, there is no existing algorithm for the problem studied in this paper. Except the Time Slot Filtering component of ILAH (the statistical difference is not significant between ILAHs with and without the Time Slot Filtering component in the above calibration), there are two variants of the Initial Solution Construction (MCARF and EFTF), two candidates of the Solution Improvement (FIH and GIH) and one Perturbation operator (β=0.4). Therefore, we compare four ILAH-based algorithms: MCFIH, MCGIH, EFIH and EGIH. The MCARF are selected by the MCFIH and the MCGIH while the EFTF are chosen by the EFIH and the EGIH to generate initial solutions.

The termination criterion is set as the maximum computation time $\frac{n^2}{20}$ (i.e., t=20) milliseconds. To avoid bias in the result, we do not take results from the previous calibration experiments but rather we run all algorithms again. For each of the 3645 instance combinations, 10 instances are randomly generated, i.e., totally $3645\times10\times4$=145800 tests are conducted for the four algorithm comparisons. For each algorithm, the average RPD on 10 instances of every combination is used to measure the algorithm's effectiveness.

To demonstrate the performance of the compared algorithms in a sound and statistical way, we first analyze the experimental results using the same ANOVA tool as before. The means plot of compared algorithms and 95.0% Tukey HSD intervals is shown in Figure 8.

Figure 8 illustrates that the EFIH has the smallest RPD and the EGIH has the largest, the difference is statistically significant. The RPD difference between the MCFIH and the MCGIH is not statistically significant. However, RPDs of

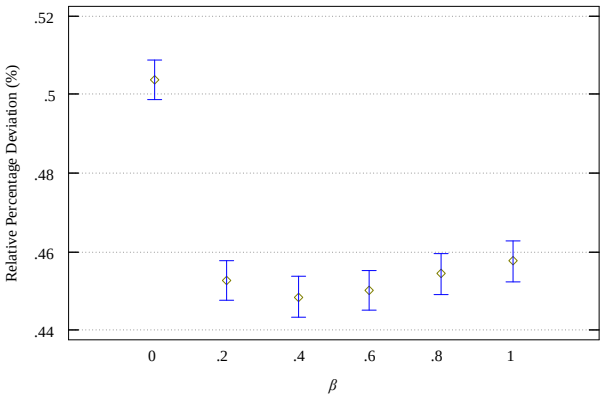

Fig. 4. Means plot of β and 95.0% Tukey HSD intervals.

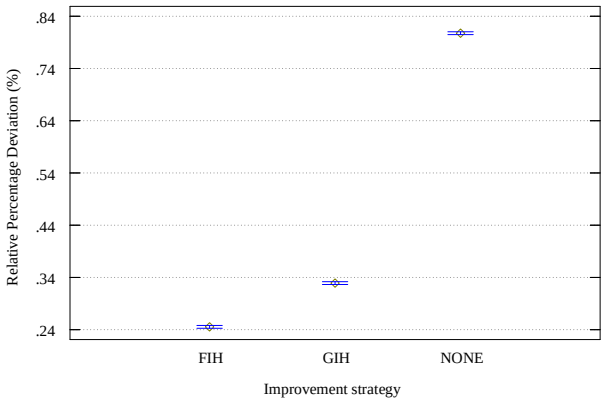

Fig. 5. Means plot of improvement strategies and 95.0% Tukey HSD intervals.

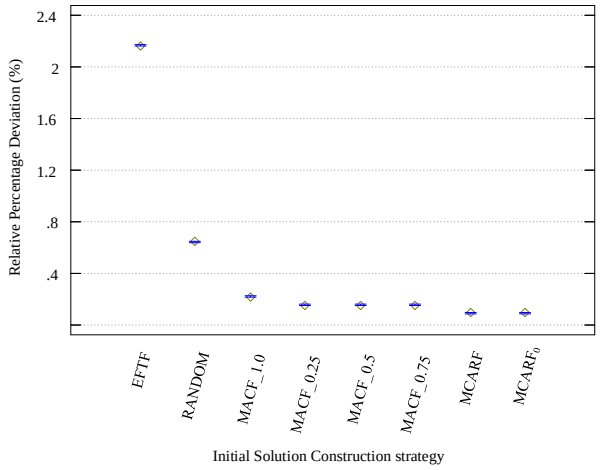

Fig. 6. Means plot of initial solution construction strategies and 95.0% Tukey HSD intervals.

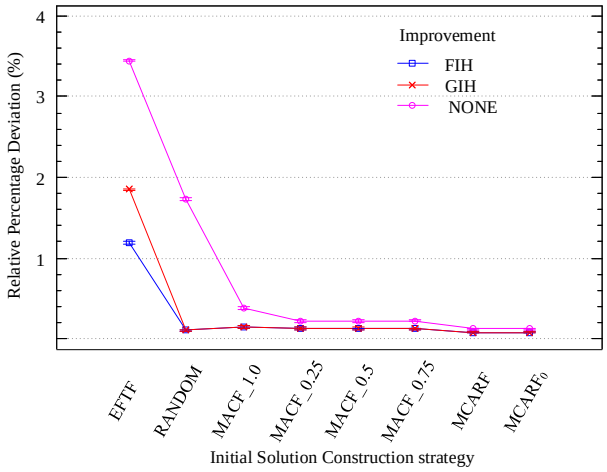

Fig. 7. Interactions between initial solution construction strategies and improvement strategies with 95.0% Tukey HSD intervals.

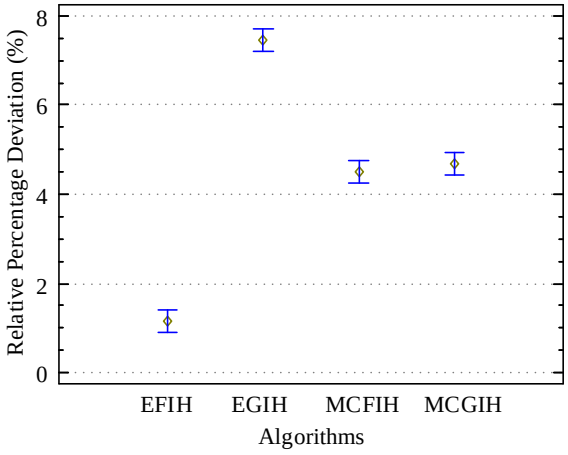

Fig. 8. Means plot of compared algorithms and 95.0% Tukey HSD intervals.

the MCFIH and the MCGIH are significantly different from those of the EFIH and EGIH. It is surprising that the EFIH outperforms the other three algorithms though the performance of EFTF is even worse than that of RANDOM to generate initial solutions. This implies that the FIH is much more effective than the GIH for improving poor solutions.

To further demonstrate the performance of the four compared algorithms, ARPDs (Average Relative Percentage Deviation) on all instance combinations are shown in Table 2.

From Table 2, we observe that the EFIH has the smallest ARPDs on all parameter combinations. The EGIH has the largest ARPD on each parameter combinations. The MCFIH and the MCGIH have similar ARPDs on all the instances. The average ARPDs of the four algorithms supports the results shown in Figure 8. The average ARPD of EFIH is only 1.17% which is much better than that of EGIH with 7.44%. The average ARPDs of MCFIH and MCGIH are 4.52% and 4.68%, which are worse than that of EFIH.

5.3 Influences of instance parameters on the compared algorithms' performance

Table 2 also shows that ARPDs of the compared algorithms are different when an instance parameter takes different values.

TABLE 2
ARPDs of the compared algorithms

Parameter	Value	EFIH	EGIH	MCFIH	MCGIH
DF	0.2	**1.24**	5.06	2.13	2.26
	0.4	**1.17**	7.28	4.55	4.68
	0.6	**1.09**	9.99	6.87	7.08
N	200	**1.68**	6.45	4.15	4.35
	400	**1.43**	7.01	4.28	4.37
	600	**1.08**	7.55	4.37	4.59
	800	**0.96**	8.07	5.02	5.21
	1000	**0.70**	8.14	4.75	4.85
OS	0.1	**1.11**	6.86	4.49	4.62
	0.2	**1.19**	7.64	4.17	4.34
	0.3	**1.20**	7.83	4.90	5.06
LoadNum	0	**1.14**	5.91	4.85	5.12
	1	**0.98**	7.89	4.85	4.93
	2	**1.38**	8.53	3.84	3.98
CF	convex	**0.87**	0.93	9.47	9.54
	concave	**1.30**	15.82	1.49	2.06
	hybrid	**1.32**	5.59	2.59	2.42
M	U[2,10]	**2.71**	15.99	6.92	7.43
	U[11,20]	**0.47**	4.91	4.60	4.60
	U[21,30]	**0.31**	1.43	2.03	2.00
CP	1	**1.37**	6.43	2.66	2.83
	2	**1.07**	7.78	4.34	4.55
	3	**1.06**	8.13	6.55	6.64
Average		**1.17**	7.44	4.52	4.68

We analyze influences of six instance parameters (M, OS, CF, CP, $Loadnum$ and DF) on the compared algorithms' performance using the ANOVA tool as before. Interactions between each parameter and the compared algorithms with 95.0% Tukey HSD intervals are depicted in Figure 9.

Figure 9 illustrates that the size of service pool M exerts great influences on the performance of the compared algorithms. The RPDs are the smallest when M takes U[21,30]. All algorithms obtain similar RPDs in the U[21,30] case. The reason lies in that a bigger service pool gives more service candidates which leads to smaller total costs. RPD differences are statistically significant for each algorithm in the three cases except that the RPD difference of the EFIH is not statistically significant in the U[11,20] and U[21,30] cases. OS exerts a little influence on the compared algorithms. RPD differences are not statistically significant for each algorithm in all cases. The cost function CF has a great influence on the EGIH. However, RPD differences are not statistically significant for the EFIH with any cost functions. The MCFIH and the MCGIH obtain much poorer performance when we adopt the convex cost function. RPD differences of EFIH and EGIH for different CP values are not statistically significant while those of MCFIH and MCGIH are statistically significant. With the increase of CP, the RPD of EFIH decreases but those of the other three increase. $Loadnum$ gives impact on the performance of the EGIH. RPD differences of the EGIH are statistically significant for different $Loadnum$ values. However, RPD differences of the other three algorithms are not statistically significant for different $Loadnum$ values, i.e., $Loadnum$ exerts a little influence on EFIH, MCFIH and MCGIH. RPD differences of the EFIH are not statistically significant whereas those of the other three are statistically significant for different DF values. In addition, RPDs increase with the increase of DF for all the compared algorithms except the EFIH.

Table 2 and Figure 9 demonstrate that the proposed EFIH is the most robust algorithm among the proposals for all instance parameters (M, OS, CF, CP, $Loadnum$ and DF). In other words, EFIH is suitable for scheduling deadline constrained realistic workflow applications (e.g., Montage, CyberShake, Epigenomics, LIGO, and SIPHT) on rented resources (e.g., EC2 virtual machines from Amazon). In addition, EFIH is applicable to workflow scheduling in cloud manufacturing where virtualized manufacturing resources are rented for worklfow applications with deadlines.

6 Conclusions and Future Work

We have considered workflow scheduling with deadline and time slots constraints in cloud computing to minimize total costs. The problem was modeled as the WSDT which is more practical than the DTCTP. We proved that the WSDT had different properties from the DTCTP. The ILAH (iterated local adjusting heuristic) framework was proposed for the NP-hard WSDT. Three initial solution construction strategies were developed among which the MCARF and the MACF showed more effective than the EFTF on initial solution construction. Two improvement strategies, the FIH and the GIH, were introduced which had similar influences on the solution improvement. The FIH was very effective for improving poor solutions. By integrating the worst and best initial solution construction strategies (EFTF and MCARF) with the two improvement strategies, four ILAH-based algorithms were developed. Though the EFTF was the worst initial solution construction strategy, it was strange that the EFIG showed the best performance. However, the EGIH obtained the worst performance. In addition, the EFTF was not sensitive to instance parameters while the EGIH was affected by most of the parameters.

For future research, the impact of different pricing interval lengths on workflow scheduling is worth studying. Instance-intensive workflows is also a desirable area of study for future work.

Acknowledgment

This work has been supported by the National Natural Science Foundation of China (Nos. 61572127, 61272377) and the Key Research & Development Program in Jiangsu Province (No. BE2015728). Rubén Ruiz is partially supported by the Spanish Ministry of Economy and Competitiveness, under the project "RESULT - Realistic Extended Scheduling Using Light Techniques" (No. DPI2012-36243-C02-01) partially financed with FEDER funds.

References

[1] C. Weinhardt, A. Anandasivam, B. Blau, and J. Stößer, "Business models in the service world." *IT professional*, vol. 11, no. 2, pp. 28–33, 2009.

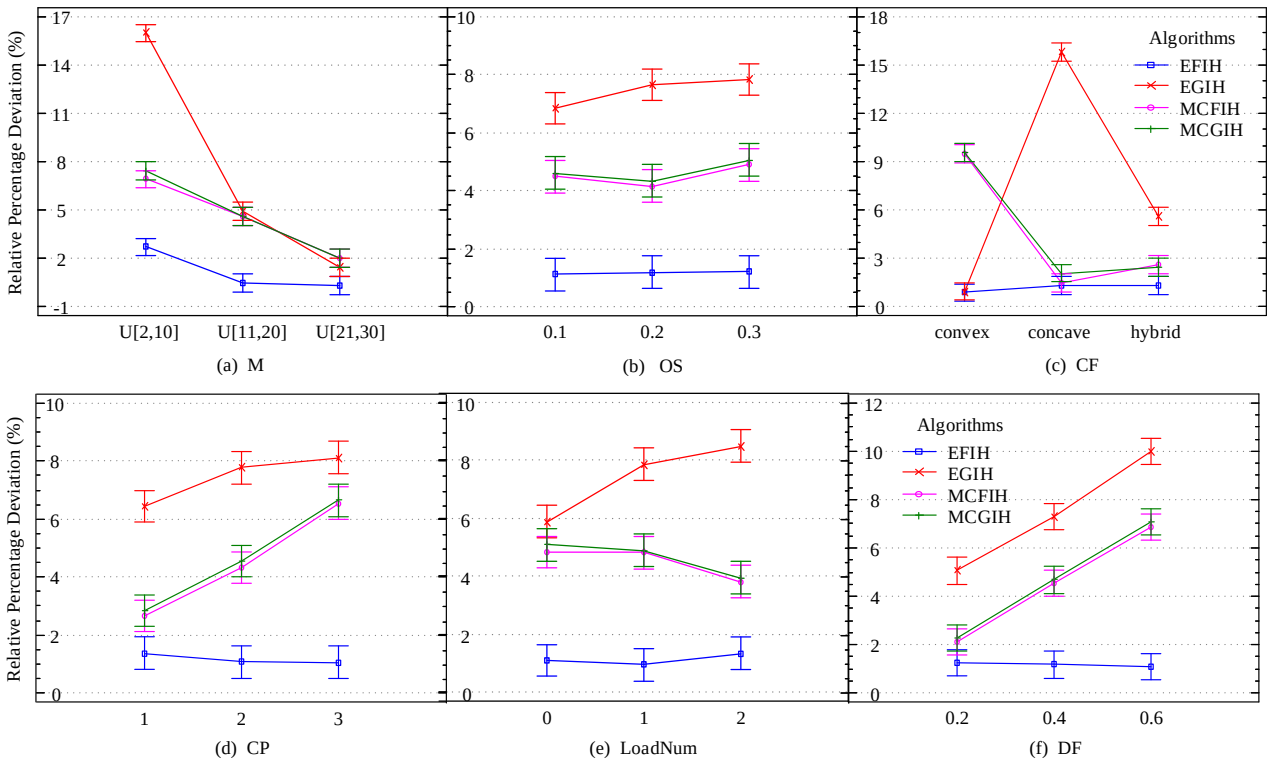

Fig. 9. Interactions between instance parameters and the compared algorithms with 95.0% Tukey HSD intervals.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. IEEE, 2008, pp. 5–13.

[3] E. L. Demeulemeester, W. S. Herroelen, and S. E. Elmaghraby, "Optimal procedures for the discrete time/cost trade-off problem in project networks," *European Journal of Operational Research*, vol. 88, no. 1, pp. 50–68, 1996.

[4] X. Zhang, L. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 363–373, 2014.

[5] M. Menzel, R. Ranjan, L. Wang, S. Khan, and J. Chen, "Cloudgenius: A hybrid decision support method for automating the migration of web application clusters to public clouds," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1336–1348, 2015.

[6] W. Dou, X. Zhang, J. Liu, and J. Chen, "Hiresome-ii: Towards privacy-aware cross-cloud service composition for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 455–466, 2015.

[7] C. Liu, R. Ranjan, C. Yang, X. Zhang, L. Wang, and J. Chen, "Mur-dpa: Top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2609–2622, 2015.

[8] A. Verma and S. Kaushal, "Deadline constraint heuristic–based genetic algorithm for workflow scheduling in cloud," *International Journal of Grid and Utility Computing*, vol. 5, no. 2, pp. 96–106, 2014.

[9] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[10] S. Abrishami and M. Naghibzadeh, "Deadline-constrained workflow scheduling in software as a service cloud," *Scientia Iranica*, vol. 19, no. 3, pp. 680–689, 2012.

[11] A. G. Delavar and Y. Aryan, "Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems," *Cluster computing*, vol. 17, no. 1, pp. 129–137, 2014.

[12] C. Akkan, A. Drexl, and A. Kimms, "Network decomposition-based benchmark results for the discrete time–cost tradeoff problem," *European Journal of Operational Research*, vol. 165, no. 2, pp. 339–358, 2005.

[13] P. De, E. J. Dunne, J. B. Ghosh, and C. E. Wells, "Complexity of the discrete time-cost tradeoff problem for project networks," *Operations Research*, vol. 45, no. 2, pp. 302–306, 1997.

[14] T. J. Hindelang and J. F. Muth, "A dynamic programming algorithm for decision CPM networks," *Operations Research*, vol. 27, no. 2, pp. 225–241, 1979.

[15] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *e-Science and Grid Computing, 2005. First International Conference on*. IEEE, 2005, pp. 8–pp.

[16] Y. Yuan, X. Li, Q. Wang, and X. Zhu, "Deadline division-based heuristic for cost optimization in workflow scheduling," *Information Sciences*, vol. 179, no. 15, pp. 2562–2575, 2009.

[17] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[18] C.-C. Liu, W.-M. Zhang, and Z.-G. Luo, "Path balance based heuristics for cost optimization in workflow scheduling," *Ruan Jian Xue Bao/Journal of Software*, vol. 24, no. 6, pp. 1207–1221, 2013, (in Chinese).

[19] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3, pp. 217–230, 2006.

[20] W.-N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 1, pp. 29–43, 2009.

[21] H. Mokhtari, R. Baradaran Kazemzadeh, and A. Salmasnia, "Time-cost tradeoff analysis in project management: An ant system approach," *IEEE Transactions on Engineering Management*, vol. 58, no. 1, pp. 36–43, 2011.

[22] Z. Cai, X. Li, and L. Chen, "Dynamic programming for services scheduling with start time constraints in distributed collaborative manufacturing systems," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, 2012, pp. 803–808.

[23] Z. Cai, X. Li, and J. N. D. Gupta, "Critical path-based iterative heuristic for workflow scheduling in utility and cloud computing," in *Service-Oriented Computing (ICSOC), 11 th International Conference on*. Springer, 2013, pp. 207–221.

[24] ——, "Heuristics for provisioning services to workflows in XaaS Clouds," *IEEE Transactions on Services Computing, Accepted, Doi: 10.1109/TSC.2014.2361320*.

[25] Z. Cai, C. L. Li, Xiaoping, and J. N. D. Gupta, "Bi-direction adjust heuristic for workflow scheduling in clouds," in *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE, 2013, pp. 94–101.

[26] E. Demeulemeester, M. Vanhoucke, and W. Herroelen, "Rangen: A random network generator for activity-on-the-node networks," *Journal of Scheduling*, vol. 6, no. 1, pp. 17–38, 2003.

[27] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, *Experimental methods for the analysis of optimization algorithms*. Springer, 2010.