Task Scheduling Algorithm for the Communication, Ocean, and Meteorological Satellite

Soojeon Lee, Won Chan Jung, and Jae-Hoon Kim

In this paper, we propose an efficient single-resource task scheduling algorithm for the Communication, Ocean, and Meteorological Satellite. Among general satellite planning functions such as constraint check, priority check, and task scheduling, this paper focuses on the task scheduling algorithm, which resolves conflict among tasks which have an exclusion relation and the same priority. The goal of the proposed task scheduling algorithm is to maximize the number of tasks that can be scheduled. The rationale of the algorithm is that a discarded task can be scheduled instead of a previously selected one depending on the expected benefit acquired by doing so. The evaluation results show that the proposed algorithm enhances the number of tasks that can be scheduled considerably.

Keywords: Task scheduling algorithm, satellite mission planning system.

Manuscript received Apr. 27, 2007; revised Nov. 23, 2007.

This work was carried out by the Electronics and Telecommunications Research Institute (ETRI) under the contract (07MR1110) with the Ministry of Information and Communications (MIC), Rep. of Korea, for 'Development of Satellite Communications System for Communication, Ocean, and Meteorological Satellite (COMS)'.

Soojeon Lee (phone: + 82 42 860 1079, email: soojeonlee@etri.re.kr), Won Chan Jung (email: wcjung@etri.re.kr), and Jae-Hoon Kim (email: jhkim@etri.re.kr) are with Radio and Broadcasting Research Division, ETRI, Daejeon, Rep. of Korea.

I. Introduction

As a multi-mission GEO satellite, the Communication, Ocean, and Meteorological Satellite (COMS) is being developed jointly by the Korea Aerospace Research Institute, Electronics and Telecommunications Research Institute, and domestic and foreign industries. COMS is scheduled to be launched in 2009. The major missions of COMS are in the following three categories.

Satellite communications
- Next generation communication payload technology and space qualification
- Broadband satellite multimedia test service

Ocean observation
- Observation of marine ecology and environment around the Korean peninsula
- Assessment of oceanic life and generation of high quality fishery information

Meteorological observation
- Continuous observation of high resolution multi-channel meteorological images and generation of meteorological elements
- Early detection of abnormal meteorological phenomena such as typhoons, torrential rain, yellow sand, sea fog, and so on.
- Generation of long term sea surface temperature and cloud data

Several organizations, such as the Communications Test Earth Station (CTES), the Korea Ocean Satellite Center (KOSC), and the Meteorological Satellite Center (MSC), submit task requests for their tasks to be scheduled. However, if more than two task requests require the same resource at the same time, task requests

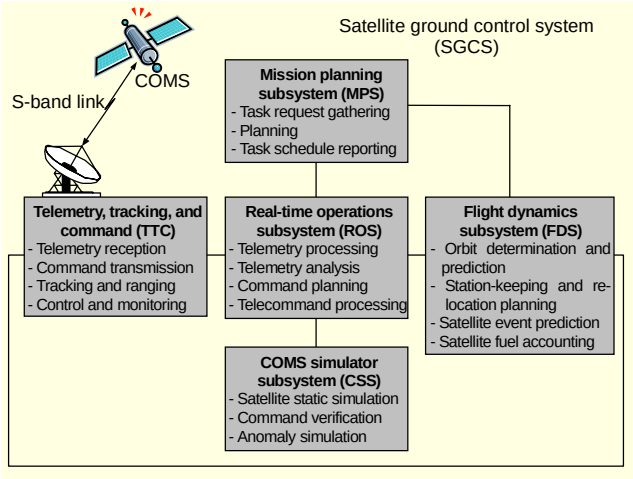

Fig. 1. Functional block diagram of SGCS.

conflict with each other. To solve this problem, a single-resource task scheduling algorithm (TSA) is required.

The goal of the proposed algorithm is to maximize the number of tasks to be scheduled. At each step of the algorithm, the task which can be finished first is meant to be selected. By selecting the task, however, another task might be discarded even though both of them have possibility to coexist. In this case, the discarded one can be scheduled instead of the formerly selected one depending on the expected benefit acquired by doing so.

II. COMS Satellite Ground Control System

1. Satellite Ground Control System

The satellite ground control system (SGCS) of COMS enables the satellite operator to perform the satellite mission. The SGCS consists of five subsystems: TTC, ROS, MPS, FDS, and CSS. Figure 1 shows a functional block diagram of SGCS. The role of each subsystem is briefly described in the figure.

In this study, the proposed TSA is implemented in MPS. Therefore, the remaining part of this section explains the MPS.

2. Mission Planning Subsystem

Figure 2 shows the functional architecture of the mission planning subsystem (MPS). Task request gathering is done through a Web interface. The task requesters, CTES, KOSC, and MSC, access the Web server of MPS and submit their task requests. Event information, such as solar eclipses, and maneuver requests, such as station-keeping, are received from the FDS. Thereafter, the MPS plans the task request in three steps.

A. Meteorological Imager and Geostationary Ocean Color Imager Algorithms

For meteorological imager (MI) requests received from the MSC, image duration calculation, scan coordinate conversion, and proportional command generation are performed by an MI algorithm. For geostationary ocean color imager (GOCI) requests received from KOSC, the displacement angle of the mirror pointing mechanism is calculated by the GOCI algorithm. Note that the requests received from CTES do not need any specific algorithm.

B. Constraint and Priority Check

By the pre-defined COMS specific relation rules (such as exclusion, inclusion, and predecessor-successor among task request, event information and maneuver requests), constraints are checked and resolved. Especially if tasks which have an exclusion relation and different priorities overlap, the task having lower priority is always discarded by the priority rules.

C. Task Scheduling Algorithm

Using its own TSA, the MPS generates a conflict-free task schedule. Note that the proposed TSA only focuses on the scheduling of tasks having an exclusion relation and the same priority.

After completing planning, the MPS transmits the conflict-free task schedule to the real-time operations subsystem (ROS) so that the schedule can be used for command planning. The task schedule is also returned to the task requesters through a Web interface so that the task requesters can confirm the scheduling result. If the number of task requesters is large, a multi-task approach [1], [2] can be used to reduce the network burden. The task execution status is sent from the ROS and is stored in MPS.

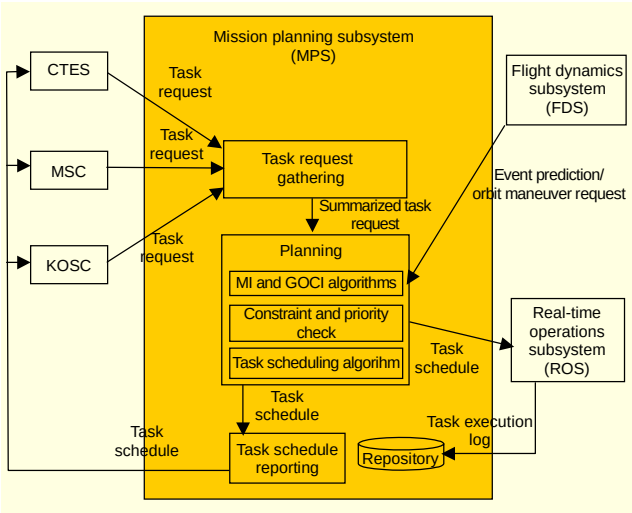

Fig. 2. Functional architecture of the MPS.

III. Related Works

There have been many studies on scheduling algorithms in

computer science and operations research. Although only a limited number of papers specifically deal with satellites, general scheduling algorithms can be adapted to the specific problems of satellite task scheduling. [3]

The task scheduling problem can be considered as a specific form of the general job-shop scheduling problem which requires the optimization of resource usage. In the job-shop scheduling problem, there is a finite set of *n* jobs, and each job consists of a chain of tasks. There is a set of *m* machines, and each machine can handle a maximum of one task at a time. Each task needs to be processed during an uninterrupted period of a given length on the machine [4]. A precedence relation between tasks exists; however, the total ordering only belongs to the same job and no precedence exists between tasks of different jobs. In [5], a performance comparison of several well-known single-resource (one machine) task scheduling algorithms is performed.

The task is to find a schedule *s* that is feasible with respect to the given constraints, and the make-span (total time) of the schedule that is minimal among all feasible schedules. The general problem is NP-hard, which means that it takes an exponential amount of computation to obtain an optimal schedule with a minimal make-span. Therefore, most approaches to the job-shop scheduling problem are based on heuristics-guided local search methods or constraint-based methods which trade-off the optimality demand on the schedule with the efficiency of the scheduling algorithm. The underlying rationale is to quickly obtain a reasonably good schedule, rather than spending too much time and producing an optimal schedule which is only slightly better than the one found by heuristics-based methods.

In [6], a very good survey is presented on local-search methods for the job-shop scheduling problem. A local search is based on the idea that a given solution (schedule) can be improved by making small (local) changes to the current solution. Essentially, a local search method requires three concepts. First, we need a "cost function" c such that $c(s)$ (> 0) is the cost of the schedule s. Second, we need a "neighborhood" $N(s)$ for a schedule s, which is a set of schedules "close" to schedule s. Third, we need a strategy for searching the neighborhood of a schedule. Thus, a local search can be seen as a search of the set of all feasible schedules, moving from one schedule to one of its neighbors. The simplest form of cost function is just the total time of the schedule. A simple example of $N(s)$ is the set of schedules which are obtained from schedule s by swapping the order of two tasks. In most cases, the means to define a neighborhood and search the neighborhood efficiently affect the search for good schedules. Numerous variant methods exist for the local search paradigm, including iterative improvement, simulated annealing (which accepts a non-improving solution probabilistically), genetic algorithms, and so on.

Besides local search methods, the constraint satisfaction approach has also been used by numerous researchers for the job-shop scheduling problem. The general idea is to treat the scheduling problem as one of finding assignments to variables (start times of tasks) such that the assignments satisfy all the relevant constraints (precedents, and so on). The basic loop in a constraint satisfaction scheme is to select a variable (task) and the value of the selected variable (start time for the chosen task), then check for consistency and escape from inconsistencies (dead-ends) until a solution is found [7].

A method to improve a feasible schedule using schedule-repair was developed in [8]. It is the most notable relevant proposal on scheduling in the space-related research field. The main procedure used in this method is called *MissionSwap*. Mission is a synonym for task in this context. The idea is as follows. First, a feasible schedule s is obtained quickly using mission priorities as selection criteria. Then, the method invokes the MissionSwap procedure, which temporarily "swaps out" some of the scheduled missions to make room for some currently un-scheduled missions. MissionSwap will then try to reinsert those just swapped out. If such a reinsertion is successful, then the updated schedule s^* is retained; otherwise, the previous schedule s is restored. In [8], Kramer and others performed some experiments on this method using the US Air Force's Air Mobility Command scheduling problem as a test problem. They also described how the MissionSwap procedure could be used to handle scheduling problems in space-related domains, such as the problem in earth orbiting satellite fleet observation scheduling and the observation scheduling problem for a satellite's solid state recorder.

IV. Single-Resource Task Scheduling Algorithm

1. Notation

A task T is composed of six parameters as follows:

$$T = \langle ID, S\_w, F\_w, D, S, F \rangle \quad (1)$$

The ID of T is $T.ID$. The lower and upper bounds of T's time window are $T.S\_w$ and $T.F\_w$. Thus, the time window is denoted by $[T.S\_w, T.F\_w]$. The duration of T is $T.D$. Start and finish times of T are $T.S$ and $T.F$, respectively, where $T.F = T.S + T.D$. The slack of T is the same as $T.F\_w – T.S\_w – T.D$.

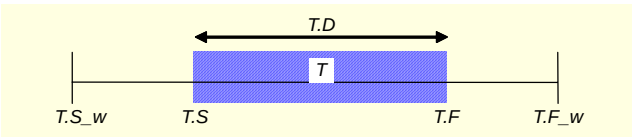

Fig. 3. Description of a task *T*.

As shown in Fig. 3, *T* can only be processed continuously within its time window.

2. First-Finished First-Scheduled Algorithm

The first-finished first-scheduled (FFFS) algorithm is similarly used in [9]. It begins with an empty schedule and adds first-finished tasks to the schedule as the algorithm proceeds. The purpose of this algorithm is to compress the current partial schedule as much as possible so that more tasks can be inserted later.

As shown in Fig. 4, the function *fffs* begins with the parameter *req*, which is a sequential list of tasks:

$$req = [T1, T2, \cdots, Tn]. \quad (2)$$

At this phase, each *T* in *req* has not determined *T.S* and *T.F*. Those two values are decided as the algorithm proceeds. *Finish_time* indicates the finish time of the task scheduled latest and is initialized by zero. Also, *Sched* is initialized as an empty schedule.

After that, while the loop is performed, the function *gen* is executed. It makes each *T* have the earliest possible start time *T.S* within each time window [*T.S_w*, *T.F_w*] and contains them within the *req_list*. If a task cannot be scheduled within its time window, it is discarded. Finally, the *req_list* is returned by *gen*.

```
def gen(req, finish_time):
    req_list=[]
    for t in req:
        if finish_time <= t.S_w:
            t.S=t.S_w
            t.F=t.S_w+t.D
            req_list.append(t)
        elif finish_time+t.D<=t.F_w:
            t.S=finish_time
            t.F=finish_time+t.D
            req_list.append(t)
    return req_list

def fffs(req):
    finish_time=0
    sched=Schedule()
    while True:
        req=gen(req, finish_time)
        if len(req)==0:
            break
        sort_by_finish_time(req)
        finish_time=req[0].F
        sched.append(req[0])
        req.remove(req[0])
    return sched
```

Fig. 4. Pseudo-code of the FFFS algorithm.

If *req* is empty, the algorithm terminates and returns *sched*. Otherwise, *req* is sorted by the finish time of each *T*. The *T*, which is finished earliest of those requested is *req*[*0*] and its finish time is stored to the variable, *finish_time*. Finally, the *T* is added to *sched* and removed from *req*.

3. Accommodating Discarded Task Algorithm

At each step of the FFFS algorithm, a task that is finished first is always selected. In this case, however, another task may be discarded even though both of the tasks have the possibility to coexist.

We propose the accommodating discarded task (ADT) algorithm to overcome the disadvantage of FFFS. If a task is meant to be discarded due to the first-finished one, we estimate the possibility of the coexistence of the two. If they have the probability to coexist, the discarded one is scheduled instead of the first-finished one.

There are two methods using the ADT algorithm: ADT-PC and ADP-PCEB methods.

A. Accommodating Discarded Task by Predicting Coexistence Method

The pseudo-code of ADT-PC is shown in Fig. 5. It uses the same *gen* function shown in Fig. 4. The difference between FFFS and accommodating discarded task by predicting coexistence (ADT-PC) is that the latter has additional lines from line 10 to 16 of the pseudo code as shown in Fig. 5.[1)] The rationale behind this change is that by selecting the first-finished task *req*[*0*], another task *T* might be discarded even though both of them may coexist. Rather than unconditionally excluding *T*, we schedule the discarded *T* and remove the former *req*[*0*].

This principle does not follow the conventional wisdom that the current partial schedule should be compressed as much as possible so that more tasks can be inserted later. However, it may help to include more tasks in the schedule as shown in Fig. 6. For example, let us denote possible instances of *T1* as *T1_a* and *T1_b*. At first, *T1_a* is meant to be scheduled because it is the first finished. If *T1_a* is scheduled, *T2* should be discarded by the function *gen* because any instances of *T2* cannot coexist with *T1_a*. In this case, *sched* contains only one task, *T1_a*.

However, it is possible for *T2_a* to coexist with *T1_b*. In this case, *T2_a* is selected instead of *T1_a* even though *T1_a* is the first-finished one. After selecting *T2_a*, *T1_b* is selected because it is now the first-finished one. As a result, both of the tasks are scheduled and *sched* includes two tasks, *T2_a* and *T1_b*.

1) The semantics of Python [10]'s "for-else" structure listed from line 10 to 18 of Fig. 5 means that if the "for" clause is completed without break, then the "else" clause is executed. Otherwise, "else" clause is not executed.

```
1   def adt_pc(req):
2       finish_time=0
3       sched=Schedule()
4       while True:
5           req=gen(req, finish_time)
6           if len(req)==0:
7               break
8           sort_by_finish_time(req)
9           finish_time=req[0].F
10          for t in req[1:]:
11              if t.F_w-t.D<=finish_time and t.F<req[0].F_w-req[0].D:
12                  finish_time=t.F
13                  sched.append(t)
14                  req.remove(t)
15                  break
16          else:
17              sched.append(req[0])
18              req.remove(req[0])
19      return sched
```

Fig. 5. Pseudo-code of ADT-PC.

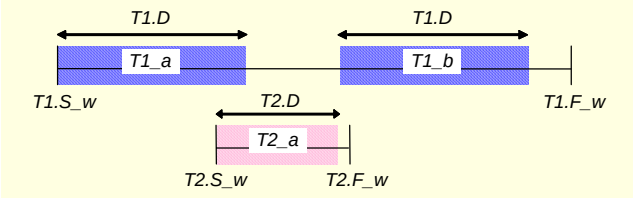

Fig. 6. Example of ADT-PC operation.

B. Accommodating Discarded Task by Predicting Coexistence and Expected Benefit Method

The accommodating discarded task by predicting coexistence and expected benefit (ADT-PCEB) method is shown in Fig. 7. It uses the same *gen* function shown in Fig. 2. The difference between ADT-PC and ADT-PCEB is that the lines from 11 to 15 of ADT-PC in Fig. 4 are changed to those of line 12 to 28 of the ADT-PCFB in Fig. 5. At each step of the algorithm, ADT-PCEB estimates the benefit of including the discarded task instead of the first-finished one *req*[*0*]. If the strategy of selecting the discarded task is expected to produce the same number of tasks with a shorter partial schedule, that is, if the condition of line 20 of Fig. 7 is met, ADT-PCEB selects the discarded task. Otherwise, if the condition of line 25 of Fig. 7 is met, ADT-PCEB selects the first-finished one *req*[*0*].

The rationale behind this heuristic is the following. The ADT-PC method always accommodates the discarded task earlier than the first-finished one if there is a probability of coexistence. In some cases, however, this method is not efficient. For example, in Fig. 8(a), ADT-PC will generate *T2_a* and *T1_b* while FFFS will generate *T1_a* and *T3_a*. Both

```
1   def adt_pceb(req):
2       finish_time=0
3       sched=Schedule()
4       while True:
5           req=gen(req, finish_time)
6           if len(req)==0:
7               break
8           sort_by_finish_time(req)
9           finish_time=req[0].F
10          for t in req[1:]:
11              if t.F_w-t.D<=finish_time and t.F<req[0].F_w-req[0].D:
12                  selected_later_F=t.F+req[0].D
13                  req2=gen(req[1:], finish_time)
14                  req2.remove(t)
15                  if len(req2)==0:
16                      finish_time=t.F
17                      sched.append(t)
18                      req.remove(t)
19                      break
20                  elif req2[0].F > selected_later_F:
21                      finish_time=t.F
22                      sched.append(t)
23                      req.remove(t)
24                      break
25                  else:
26                      sched.append(req[0])
27                      req.remove(req[0])
28                      break
29          else:
30              sched.append(req[0])
31              req.remove(req[0])
32      return sched
```

Fig. 7. Pseudo-code of ADT-PCEB.

algorithms produce two tasks, but the length of the current partial schedule of FFFS is shorter because *T3_a.F* < *T1_b.F*. For that reason, ADT-PCEB selects *T1_a* and *T3_a* instead of *T2_a* and *T1_b*. In the case shown in Fig. 8(b), ADT-PCEB also selects *T2_a* and *T1_b* instead of *T1_a* and *T3_a* because *T3_a.F* > *T1_b.F*.

V. Performance Evaluation

We compared the performance of FFFS, ADT-PC and ADT-PCEB in terms of the number of tasks included in the task schedule. Evaluations were performed using a workstation with a Pentium 4 3.0 GHz CPU, 3 GB of RAM, and MS Windows XP.

Evaluation parameters are shown in Table 1. The upper and lower bounds of a task's time window were bounded within the maximum time window, [0, 1440]. Slack and duration of the task were uniformly distributed within the corresponding interval, and *T.S_w* was also uniformly distributed within

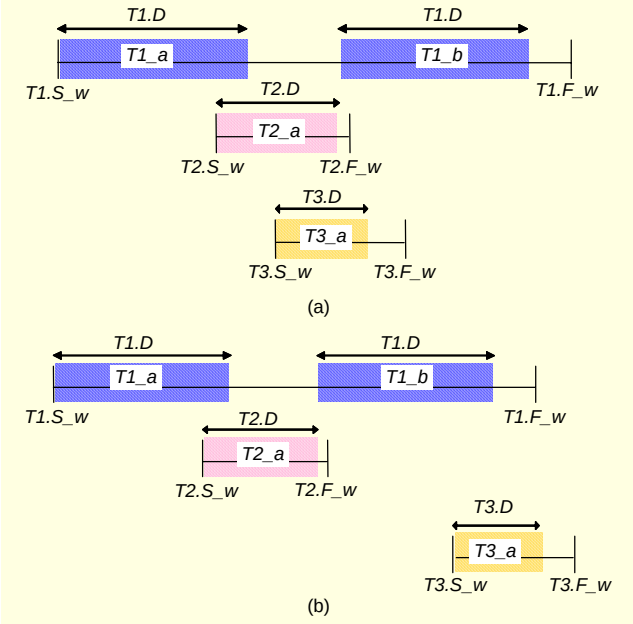

Fig. 8. Example of ADT-PCEB operation.

Table 1. Evaluation parameters.

Parameter	Value
Max time window	[0, 1440]
Slack window	[0, 50], [0, 100], [0, 150], [0, 200], [0, 250], [0,300]
Duration window	[0, 20], [0, 40], [0, 60], [0, 80], [0, 100]
Number of tasks	20, 40, 60, 80, 100

[0, 1440 – *T*'s slack – *T.D*].

Figure 9 compares the performance of FFFS, ADT-PC, and ADT-PCEB on each evaluation parameter listed in Table 1; (a), (b), (c), (d), and (e) describe the cases when the numbers of requested tasks are 20, 40, 60, 80, and 100, respectively. We repeated the evaluation procedures 100 times to acquire the averages. The FFFS algorithm is considered as a baseline measure to analyze the characteristics of the other two algorithms.

The colored areas indicate that ADT-PC shows the best performance. Values in bold in these areas indicate that both ADT-PC and ADT-PCEB show the best performance. The white areas indicate that ADT-PCEB shows the best performance. Values in bold in these areas indicate that ADT-PC shows the worst performance.

1. Effectiveness of Number of Requested Tasks and Duration

A. Toughness

There are two elements which determine the *toughness* of the experimental environment: number of requested tasks and duration. The greater the number of requested tasks is, or the longer the duration is, the tougher the environment is. It is called a *tough environment* if both elements are large. A *soft environment* is the opposite of a tough environment. Figure 9 shows that, for all algorithms, the performance deteriorates as the environment becomes tougher. Interestingly, in Fig. 9, the performance of ADT-PCEB is always superior to that of FFFS regardless of the toughness level.

B. Soft Environment

The softer the environment is, the better the ADT-PC performs compared to the other two algorithms. For example, when the number of requested tasks is small, namely, 20, ADT-PC shows the best performance, regardless of the duration. Moreover, when the duration window is small, that is, [0, 20], ADT-PC is the best regardless of the number of requested tasks.

The ADT-PCEB method also works well in a soft environment, although it is not as good as ADT-PC. The FFFS algorithm shows the worst performance overall.

C. Tough Environment

The tougher the environment is, the better the ADT-PCEB performs compared to the other two algorithms. However, ADT-PC performs even worse than FFFS in a very tough environment, including two cases shown in Fig. 9(d) and six cases shown in Fig. 9(e).

2. Effectiveness of Slack

A. Degree of Freedom

Slack can be considered a *degree of freedom* because a requested task with larger slack can be placed in a wider range of possible positions within a time window.

B. Influence of Slack to Performance

It is clear that, in most cases, performance of the three algorithms tends to improve as the slack increases, although there are some exceptions. However, the impact of slack on performance is not as strong as the impact of the number of requested tasks and the duration. Moreover, slack seems neither advantageous nor disadvantageous for a specific algorithm.

C. Performance in a Tough Environment with a Large Degree of Freedom

It is interesting to note that in a tough environment, ADT-PC performs even worse than FFFS, and these cases happen only with a large slack window. As shown in Fig. 8(a), ADT-PC

Duration window	Slack window [0,50]		[0,100]		[0,150]		[0,200]		[0,250]		[0,300]	
[0,20]	**FS**	**19.75**	**FS**	**19.75**	**FS**	**19.9**	**FS**	**20**	**FS**	**19.9**	**FS**	**19.9**
	PC	**19.85**	**PC**	**20**	**PC**	**20**	**PC**	**20**	**PC**	**20**	**PC**	**20**
	EB	**19.85**	**EB**	**20**	**EB**	**20**	**EB**	**20**	**EB**	**20**	**EB**	**20**
[0,40]	**FS**	**18.5**	**FS**	**19.05**	**FS**	**19.15**	**FS**	**19.45**	**FS**	**19.65**	**FS**	**19.9**
	PC	**19.3**	**PC**	**19.8**	**PC**	**19.75**	**PC**	**19.85**	**PC**	**19.95**	**PC**	**19.95**
	EB	**19.3**	**EB**	**19.8**	**EB**	**19.75**	**EB**	**19.85**	**EB**	**19.95**	**EB**	**19.95**
[0,60]	**FS**	**16.75**	FS	17.7	FS	18.55	**FS**	**18.5**	**FS**	**19.2**	FS	19.3
	PC	**17.3**	PC	18.75	PC	19.25	**PC**	**19.55**	**PC**	**19.65**	PC	19.9
	EB	**17.3**	EB	18.7	EB	19.15	**EB**	**19.55**	**EB**	**19.65**	EB	19.85
[0,80]	**FS**	**15.95**	FS	16.6	**FS**	**17.5**	FS	17.6	FS	18.25	FS	18.3
	PC	**16.25**	PC	17.5	**PC**	**18.55**	PC	18.7	PC	19.1	PC	19.3
	EB	**16.25**	EB	17.45	**EB**	**18.55**	EB	18.65	EB	19	EB	19.2
[0,100]	**FS**	**14.35**	**FS**	**15.9**	**FS**	**16.75**	**FS**	**16.45**	FS	17.25	FS	17.75
	PC	**14.65**	**PC**	**16.5**	**PC**	**17.5**	**PC**	**17.6**	PC	18.4	PC	18.8
	EB	**14.65**	**EB**	**16.5**	**EB**	**17.5**	**EB**	**17.6**	EB	18.3	EB	18.55

(a) Number of requested tasks = 20

Duration window	Slack window [0,50]		[0,100]		[0,150]		[0,200]		[0,250]		[0,300]	
[0,20]	**FS**	**38.2**	FS	39	**FS**	**39.3**	FS	39.35	**FS**	**39.5**	**FS**	**39.8**
	PC	**39.25**	PC	39.9	**PC**	**39.95**	PC	39.95	**PC**	**39.9**	**PC**	**40**
	EB	**39.25**	EB	39.85	**EB**	**39.95**	EB	39.9	**EB**	**39.9**	**EB**	**40**
[0,40]	**FS**	**33.95**	FS	35.15	FS	36.4	FS	37.4	FS	37.75	FS	37.6
	PC	**35.4**	PC	37.1	PC	38.5	PC	39.3	PC	39.2	PC	39.65
	EB	**35.4**	EB	36.9	EB	38.15	EB	39	EB	38.9	EB	39.1
[0,60]	FS	28.8	FS	31.8	FS	33.1	FS	33.25	FS	33.35	FS	34.2
	PC	30.35	PC	33.55	PC	35	PC	35.4	PC	35.55	PC	36.7
	EB	30.4	EB	33.4	EB	34.75	EB	35.15	EB	35.3	EB	35.95
[0,80]	**FS**	**25.7**	FS	28.05	FS	29	FS	29.55	FS	29.2	FS	30.75
	PC	**26.45**	PC	29.45	PC	31.1	PC	31.55	PC	31.4	PC	32.4
	EB	**26.45**	EB	29.4	EB	31.05	EB	31.65	EB	31.25	EB	32.2
[0,100]	**FS**	**22.55**	FS	24.95	FS	25.6	FS	26.5	FS	27.65	FS	27.6
	PC	**23.7**	PC	26.55	PC	27.1	PC	28.3	PC	29.4	PC	29.25
	EB	**23.7**	EB	26.7	EB	27.5	EB	28.8	EB	29.2	EB	29.4

(b) Number of requested tasks = 40

Duration window	Slack window [0,50]		[0,100]		[0,150]		[0,200]		[0,250]		[0,300]	
[0,20]	FS	55.8	FS	58.45	FS	58.1	FS	58.4	FS	58.75	FS	58.75
	PC	57.9	PC	59.6	PC	59.75	PC	59.75	PC	59.85	PC	59.9
	EB	57.75	EB	59.4	EB	59.6	EB	59.65	EB	59.7	EB	59.75
[0,40]	FS	46.75	FS	48.9	FS	51.05	FS	52	FS	52.3	FS	53.3
	PC	49.3	PC	52.05	PC	54.1	PC	55.45	PC	55.95	PC	56.5
	EB	49.35	EB	52	EB	53.7	EB	54.3	EB	55	EB	55.25
[0,60]	FS	38.55	FS	42.25	FS	42.25	FS	43.55	FS	43.8	FS	45.05
	PC	41.05	PC	44.75	PC	44.75	PC	46.4	PC	46.1	PC	47.25
	EB	41	EB	45.05	EB	45.05	EB	46.05	EB	45.95	EB	47.45
[0,80]	**FS**	**33.1**	FS	35.25	FS	36.65	FS	37.6	FS	39.35	FS	38.95
	PC	**34.65**	PC	37.4	PC	38.75	PC	38.5	PC	40.85	PC	40.3
	EB	**34.65**	EB	37.8	EB	39.15	EB	39.75	EB	41.65	EB	41.35
[0,100]	FS	29.65	FS	31.2	FS	32.85	FS	32.65	FS	33	FS	34.3
	PC	31.2	PC	32.55	PC	34.35	PC	33.2	PC	34.2	PC	35.0
	EB	31.35	EB	33.15	EB	35.1	EB	34.4	EB	35.6	EB	36.45

(c) Number of requested tasks = 60

Duration window	Slack window [0,50]		[0,100]		[0,150]		[0,200]		[0,250]		[0,300]	
[0,20]	FS	71.45	FS	74	FS	76.2	FS	77.1	FS	77.3	FS	77.45
	PC	75.3	PC	78	PC	79	PC	79.35	PC	79.55	PC	79.5
	EB	75.1	EB	77.65	EB	78.4	EB	78.8	EB	78.8	EB	79
[0,40]	FS	55.4	FS	59.1	FS	62.3	FS	62.15	FS	63.05	FS	63.3
	PC	58.3	PC	63.35	PC	65.75	PC	65.95	PC	66.4	PC	67.85
	EB	58.75	EB	63.25	EB	65.9	EB	65.25	EB	66.15	EB	66.4
[0,60]	FS	44.45	FS	49.1	FS	50.95	FS	51.3	FS	52.6	FS	51.9
	PC	46.9	PC	51.05	PC	53.3	PC	53.45	PC	53.9	PC	53.4
	EB	46.95	EB	51.65	EB	54.8	EB	54.35	EB	55.4	EB	54.6
[0,80]	FS	39.95	FS	41.95	FS	42.55	FS	44.7	FS	44.25	FS	45.3
	PC	42.65	PC	44.35	PC	44.1	PC	45.3	PC	44.35	PC	45.7
	EB	42.75	EB	45.3	EB	45.3	EB	47.5	EB	46.6	EB	48.05
[0,100]	FS	34.35	FS	37.1	FS	38.35	FS	39.15	**FS**	**39.9**	**FS**	**39.85**
	PC	36.35	PC	39.1	PC	39.3	PC	38.9	**PC**	**38.95**	**PC**	**39.05**
	EB	36.5	EB	39.9	EB	41.3	EB	41.35	**EB**	**42.3**	**EB**	**41.9**

(d) Number of requested tasks = 80

Duration window	Slack window [0,50]		[0,100]		[0,150]		[0,200]		[0,250]		[0,300]	
[0,20]	FS	85	FS	89.3	FS	92.6	FS	93	FS	94.65	FS	95.4
	PC	90.55	PC	95.6	PC	97.55	PC	98.3	PC	98.8	PC	98.95
	EB	90.25	EB	94.1	EB	96.05	EB	96.2	EB	97.25	EB	97.5
[0,40]	FS	63.75	FS	68.85	FS	70.95	FS	72.45	FS	71.35	FS	72.85
	PC	67.15	PC	72.55	PC	74.5	PC	75.05	PC	74.35	PC	75.75
	EB	67.45	EB	72.4	EB	74.85	EB	75.5	EB	74.85	EB	76.3
[0,60]	FS	51.35	FS	55.75	FS	57.3	FS	58.8	FS	59.75	FS	59.85
	PC	54.9	PC	57.9	PC	58.2	PC	60.9	PC	60.25	PC	60.3
	EB	54.95	EB	59.5	EB	60.65	EB	62.7	EB	63.1	EB	62.85
[0,80]	FS	43.45	FS	47.55	FS	49.8	**FS**	**49.45**	**FS**	**50.3**	**FS**	**52.1**
	PC	45.9	PC	49.75	PC	49.8	**PC**	**49**	**PC**	**49.55**	**PC**	**50.1**
	EB	46.15	EB	50.7	EB	52.6	**EB**	**52.45**	**EB**	**53.4**	**EB**	**54.95**
[0,100]	FS	40.1	FS	41.05	FS	43.6	**FS**	**44.75**	**FS**	**44.15**	**FS**	**45.95**
	PC	42.3	PC	43.25	PC	43.8	**PC**	**43.85**	**PC**	**42.65**	**PC**	**42.55**
	EB	42.35	EB	43.6	EB	46.2	**EB**	**47.25**	**EB**	**46.85**	**EB**	**48.05**

(e) Number of requested tasks = 100

Note: FFFS, ADT-PC, and ADT-PCEB are abbreviated to FS, PC, and EB, respectively.

Fig. 9. Performance of FFFS, ADT-PC, and ADT-PCEB algorithms.

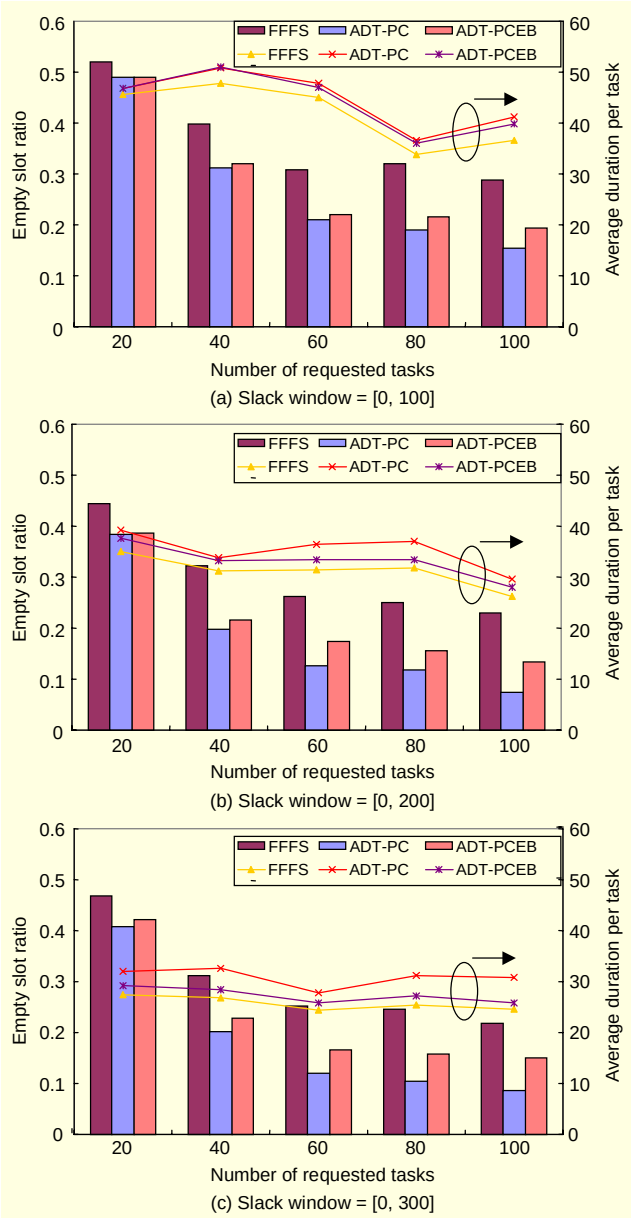

Fig. 10. Empty slot ratio and average duration per task when duration window = [0, 100].

selects *T2_a* instead of *T1_a*, and this strategy is demonstrated to be less efficient than FFFS if there is *T3*. However, as the environment gets tougher and the degree of freedom becomes larger, the probability that at least one task can be placed in the location of *T3* increases. Consequently, ADT-PC performs worse than FFFS under this condition.

3. Effectiveness of Empty Slot and Average Duration per Task

A. Empty Slot Ratio and Utilization Ratio

An *empty slot* means an interval of a schedule on which no task is allotted. For example, in Fig. 8(a), if *T1_a* and *T3_a* are selected, [*T1_a.F*, *T3_a.S*] will be empty on the schedule and this interval becomes an empty slot. The empty slot ratio (ESR) is the proportion of the length of total empty slots to the length of the maximum time window, which is 1,440 in this paper. The ESR is calculated as

$$ESR = \frac{|TotalEmptySlot|}{|MaxTimeWindow|}. \tag{3}$$

Figure 10 shows the empty slot ratios for each algorithm on various slack windows when the duration window is [0, 100]. In all cases, ADT-PC and FFFS show the smallest and largest empty slot ratio, respectively. This means that the utilization ratio (UR) of ADT-PC is the biggest while that of FFFS is the smallest regardless of the number of tasks to be scheduled. The UR is calculated as

$$UR = 1 - \frac{|TotalEmptySlot|}{|MaxTimeWindow|}. \tag{4}$$

As the number of requested tasks increases, the empty slot ratio of the all three algorithms tends to decrease. However, the empty slot ratio of ADT-PC decreases the most rapidly.

B. Average Duration per Task

For Fig. 10, the duration window is [0, 100]. The expected duration per requested task is 50. However, the evaluation results demonstrate that average duration per task (ADPT) is shorter than expected in most cases. This is because all of the algorithms are basically trying to shorten the current partial schedule. The FFFS algorithm shows the shortest ADPT because it always shortens the current partial schedule by scheduling the first-finished task. The ADT-PC algorithm shows the longest ADPT because it always schedules discarded tasks instead of the first-finished one if there is a possibility of coexistence.

As the slack window becomes larger, the ADPT decreases and the gap between the ADT-PC algorithm and the others tends to increases.

C. Number of Tasks to Be Scheduled

The number of tasks to be scheduled (NTS) is calculated as

$$\begin{aligned} NTS &= \frac{1-ESR}{ADPT}\,|MaxTimeWindow| \\ &= \frac{UR}{ADPT}\,|MaxTimeWindow|. \end{aligned} \tag{5}$$

In case of ADT-PC, the UR is always the largest; therefore, it is advantageous for ADT-PC to increase the NTS. However, the ADPT of ADT-PC is generally the longest; therefore, it is disadvantageous for ADT-PC to increase the NTS. Regarding

Duration window		Number of requested tasks									
		20		40		60		80		100	
[0, 60]		FS	19.3	FS	34.2	FS	45.05	FS	51.9	FS	59.85
		PC	19.9	PC	36.7	PC	47.25	PC	53.4	PC	60.3
		EB	19.85	EB	35.95	EB	47.45	EB	54.6	EB	62.85
[10, 50]		FS	19.25	FS	35.15	FS	44.5	FS	51.4	**FS**	**57.05**
		PC	19.7	PC	37.15	PC	46.05	PC	52.15	**PC**	**55.1**
		EB	19.55	EB	36.65	EB	46.3	EB	53.3	**EB**	**58.95**
[20, 40]		**FS**	**19.5**	FS	35.7	FS	43.5	**FS**	**49.85**	**FS**	**52.15**
		PC	**19.95**	PC	37.65	PC	44.8	**PC**	**49.55**	**PC**	**50.7**
		EB	**19.95**	EB	37	EB	44.9	**EB**	**51.2**	**EB**	**53**
[30,30]		FS	19.3	FS	35.45	FS	42.9	**FS**	**45.95**	FS	46.25
		PC	19.95	PC	37.7	PC	44.5	**PC**	**46.2**	PC	46.45
		EB	19.85	EB	36.65	EB	43.5	**EB**	**46.2**	EB	46.5

(a) Expected duration per requested task = 30

Duration window		Number of requested tasks									
		20		40		60		80		100	
[0, 80]		FS	18.3	FS	30.75	FS	38.95	FS	45.3	**FS**	**52.1**
		PC	19.3	PC	32.4	PC	40.3	PC	45.7	**PC**	**50.1**
		EB	19.2	EB	32.2	EB	41.35	EB	48.05	**EB**	**54.95**
[10, 70]		FS	18.65	FS	30.85	FS	38.25	**FS**	**44.05**	**FS**	**47.95**
		PC	19.45	PC	32.3	PC	39	**PC**	**42.4**	**PC**	**44.55**
		EB	19.25	EB	32.35	EB	39.85	**EB**	**46.05**	**EB**	**49.65**
[20, 60]		FS	18.3	FS	30.35	FS	36.75	**FS**	**40.95**	**FS**	**44.3**
		PC	19.45	PC	32.2	PC	37.2	**PC**	**39.75**	**PC**	**41.4**
		EB	19.25	EB	31.85	EB	38.05	**EB**	**42.2**	**EB**	**45.45**
[30, 50]		FS	18.35	FS	31.0	FS	35.45	**FS**	**37.8**	**FS**	**39.0**
		PC	19.5	PC	32.65	PC	35.7	**PC**	**36.8**	**PC**	**37.55**
		EB	19.3	EB	32.4	EB	36.2	**EB**	**38.45**	**EB**	**39.5**
[40, 40]		FS	18.8	FS	30.5	FS	33.45	FS	34.35	**FS**	**34.55**
		PC	19.85	PC	31.85	PC	33.75	PC	34.5	**PC**	**34.5**
		EB	19.55	EB	31.35	EB	33.8	EB	34.55	**EB**	**34.6**

(b) Expected duration per requested task = 40

Duration window		Number of requested tasks									
		20		40		60		80		100	
[0, 100]		FS	17.75	FS	27.6	FS	34.3	**FS**	**39.85**	**FS**	**45.95**
		PC	18.8	PC	29.25	PC	35.0	**PC**	**39.05**	**PC**	**42.55**
		EB	18.55	EB	29.4	EB	36.45	**EB**	**41.9**	**EB**	**48.05**
[10, 90]		FS	17.75	FS	26.7	**FS**	**33.35**	**FS**	**38.65**	**FS**	**42.25**
		PC	18.85	PC	28.35	**PC**	**33.1**	**PC**	**36.95**	**PC**	**38.3**
		EB	18.7	EB	28.5	**EB**	**34.9**	**EB**	**40.1**	**EB**	**44.05**
[20, 80]		FS	17.6	FS	27.55	**FS**	**32.05**	**FS**	**35.45**	**FS**	**39.7**
		PC	18.5	PC	28.3	**PC**	**31.6**	**PC**	**32.65**	**PC**	**36**
		EB	18.25	EB	28.85	**EB**	**32.9**	**EB**	**36.05**	**EB**	**40.65**
[30, 70]		FS	18.05	FS	27.0	**FS**	**30.55**	**FS**	**32.5**	**FS**	**35.4**
		PC	19.3	PC	27.55	**PC**	**30.25**	**PC**	**31.7**	**PC**	**32.85**
		EB	19.05	EB	28.05	**EB**	**31.45**	**EB**	**33.55**	**EB**	**35.6**
[40, 60]		FS	17.95	FS	26.15	**FS**	**28.95**	**FS**	**30.25**	**FS**	**31.15**
		PC	18.95	PC	26.85	**PC**	**28.3**	**PC**	**29.5**	**PC**	**30.1**
		EB	18.9	EB	27.1	**EB**	**29.3**	**EB**	**30.7**	**EB**	**31.35**
[50, 50]		FS	17.6	FS	26.05	FS	27	**FS**	**27.65**	**FS**	**27.7**
		PC	18.65	PC	26.55	PC	27	**PC**	**27.4**	**PC**	**27.55**
		EB	18.35	EB	26.35	EB	27.2	**EB**	**27.7**	**EB**	**27.75**

(c) Expected duration per requested task = 50

Duration Window		Number of requested tasks									
		20		40		60		80		100	
[10, 110]		FS	15.9	FS	24.8	**FS**	**30.05**	**FS**	**35.55**	**FS**	**39.05**
		PC	16.85	PC	25.15	**PC**	**29.55**	**PC**	**32.95**	**PC**	**33.95**
		EB	16.95	EB	26.05	**EB**	**31.7**	**EB**	**37.4**	**EB**	**40.55**
[20, 100]		FS	16.6	FS	24.6	**FS**	**29.4**	**FS**	**32.85**	**FS**	**35.55**
		PC	17.75	PC	25.1	**PC**	**27.5**	**PC**	**30.4**	**PC**	**32.25**
		EB	17.65	EB	25.55	**EB**	**30.15**	**EB**	**34.0**	**EB**	**36.85**
[30, 90]		FS	16.15	FS	24.15	**FS**	**27.15**	**FS**	**29.95**	**FS**	**31.75**
		PC	17.1	PC	24.3	**PC**	**26.4**	**PC**	**28.15**	**PC**	**29.65**
		EB	17.15	EB	25.0	**EB**	**27.85**	**EB**	**30.4**	**EB**	**32.3**
[40, 80]		FS	16.9	FS	22.95	**FS**	**26**	**FS**	**27.6**	FS	27.85
		PC	17.7	PC	23.2	**PC**	**25.25**	**PC**	**26.55**	PC	28.4
		EB	17.8	EB	23.65	**EB**	**26.25**	**EB**	**27.95**	EB	27.25
[50, 70]		FS	16.6	FS	22.45	**FS**	**24.2**	**FS**	**25.05**	FS	25.7
		PC	17.8	PC	22.95	**PC**	**23.95**	**PC**	**24.55**	PC	24.95
		EB	17.65	EB	23.3	**EB**	**24.5**	**EB**	**25.3**	EB	25.75
[60, 60]		FS	16.9	FS	21.8	FS	22.75	**FS**	**22.85**	FS	22.9
		PC	17.9	PC	22	PC	22.75	**PC**	**22.75**	PC	22.85
		EB	17.85	EB	22.15	EB	22.85	**EB**	**22.85**	EB	22.9

(d) Expected duration per requested task = 60

Duration window		Number of requested tasks									
		20		40		60		80		100	
[20, 120]		FS	15.35	**FS**	**23.3**	**FS**	**25.95**	**FS**	**30.9**	**FS**	**32.6**
		PC	16.1	**PC**	**23.2**	**PC**	**24.75**	**PC**	**28.25**	**PC**	**29.55**
		EB	16.3	**EB**	**24**	**EB**	**27.45**	**EB**	**32**	**EB**	**33.75**
[30, 110]		FS	15.6	FS	21.85	**FS**	**25.1**	**FS**	**27.85**	**FS**	**29.65**
		PC	16.6	PC	22.2	**PC**	**23.45**	**PC**	**25.95**	**PC**	**27.1**
		EB	16.7	EB	23	**EB**	**26**	**EB**	**28.35**	**EB**	**30.55**
[40, 100]		FS	15.05	**FS**	**20.9**	**FS**	**23.4**	**FS**	**25.75**	**FS**	**26.45**
		PC	15.8	**PC**	**20.65**	**PC**	**22.65**	**PC**	**24.25**	**PC**	**24.75**
		EB	15.85	**EB**	**21.35**	**EB**	**23.75**	**EB**	**25.8**	**EB**	**26.75**
[50, 90]		FS	15.15	**FS**	**20.2**	**FS**	**22.25**	**FS**	**23.4**	**FS**	**24.2**
		PC	16.25	**PC**	**20.05**	**PC**	**21.75**	**PC**	**22.25**	**PC**	**23.1**
		EB	16.3	**EB**	**20.65**	**EB**	**22.7**	**EB**	**23.55**	**EB**	**24.35**
[60, 80]		FS	14.8	**FS**	**19.3**	**FS**	**20.6**	**FS**	**21.3**	FS	21.75
		PC	15.8	**PC**	**19.45**	**PC**	**20.35**	**PC**	**21**	PC	21.2
		EB	15.65	**EB**	**19.8**	**EB**	**20.7**	**EB**	**21.4**	EB	21.75
[70, 70]		FS	15.8	**FS**	**18.75**	**FS**	**19.2**	**FS**	**19.75**	FS	19.8
		PC	16.65	**PC**	**18.85**	**PC**	**19.05**	**PC**	**19.7**	PC	19.65
		EB	16.55	**EB**	**18.85**	**EB**	**19.25**	**EB**	**19.85**	EB	19.75

(e) Expected duration per requested task = 70

Fig. 11. Effect of the interval of duration window when slack window = [0, 300].

ADT-PC, considering Fig. 9 and formula (5), we estimate that as the environment becomes tougher, the gain from the UR is reduced while the loss from the ADPT increases. As the environment becomes softer, the gain from the UR increases while the loss from the ADPT is reduced. In this regard, the FFFS algorithm shows a characteristic opposite to that of ADT-PC. The characteristic of the ADT-PCEB algorithm is between the other two.

4. Effectiveness of the Interval of Duration Window

Figure 11 shows the variation of performance with respect to the interval of the duration window when the expected duration per requested task is the same. The slack window is [0, 300]. The meaning of colored areas and bold font style is the same as in Fig. 9.

When the number of requested tasks is small, that is, 20, the interval of the duration window does not cause any significant difference in performance. However, as the number of requested tasks increases (to, for example, 100), the interval of the duration window affects the performance much more. The longer the interval is, the better the performance is.

A. Variance of Duration

A longer duration window interval causes greater variance of duration; greater variance results in more tasks having longer or shorter duration compared to those in the lower variance case. We are interested in tasks having shorter duration.

The FFFS, the basic algorithm on which the other two algorithms rely, is always trying to shorten the current partial schedule. It schedules the task having the earliest finish time. Therefore, tasks having shorter duration have a higher probability to be scheduled. Furthermore, higher variance implies that more requested tasks will have shorter duration. Consequently, although the expected duration per requested task is the same, more tasks can be scheduled by the three algorithms as the interval of duration window increases.

B. Defects of ADT-PCEB

In most cases, the performance gap between the algorithms decreases as the interval of the duration window shortens. This is because the heuristic, ADT, has little benefit as the variance of duration decreases. For example, the benefit of ADT-PCEB over FFFS decreases as the interval of duration window lengthens.

In Fig. 11, the areas outlined by double lines (duration window = [40, 80] or [70, 70] when the number of requested tasks = 100) indicate the case in which the ADT-PCEB algorithm shows the worst performance. Although the gap is trivial, this is interesting because performance of ADT-PCEB is better or equal to that of FFFS in all cases shown in Fig. 9.

C. Performance Analysis with the COMS Request Patterns

In case of COMS, as of October 2007, around 60 tasks are expected to be requested per day, which is 1,440 minutes. For more than 90 percent of task requests, duration and slack shall be between 30 and 50 minutes and between 0 and 300 minutes, respectively.

Figure 11(c) shows the case when the number of requested tasks is 60, the duration window is [30, 50], and the slack window is [0, 300]. In this case, ADT-PCEB shows the best performance, ADT-PC the next best, and FFFS the worst. However, this does not guarantee that ADT-PCEB would always be the best algorithm for COMS because the parameters, such as the number of requested tasks, duration window, and slack window can vary as the requesters' requirements change.

VI. Conclusion

We evaluated the performance of the proposed ADT algorithm using various parameters and analyzed the results. In general, ADT-PCEB performs very efficiently in most situations. The FFFS shows the worst performance in most cases; however, in a few extreme cases, particularly in a tough environment with large duration and low variance of duration, FFFS sometimes performs better than ADT-PCEB. The ADT-PC algorithm performs better in a soft environment but performs even worse than FFFS in a tough environment with large degree of freedom. The effects of the empty slot ratio, the average duration per task, and the interval of the duration window on performance was also analyzed.

Both of the ADT-PC and ADT-PCEB algorithms have been adopted for COMS because at least one of the two algorithms is better than FFFS in most cases. In planning, MPS runs both algorithms and selects the better mission schedule of the two. Moreover, although the proposed task scheduling algorithms are designed for COMS, they can be used for other satellites.

Appendix

We compared the performance of ADT-PC and ADT-PCEB with other task scheduling algorithms, such as Greedy_DP, hill climbing, and genitor [5]. Table 2 shows the evaluation parameters used for the experiments of both [5] and ours. We repeated evaluation procedures 30 times (as in [5]) to acquire the averages.

Figure 12 shows the average performance difference from the near-optimal solution. Here, the near-optimal solution is the

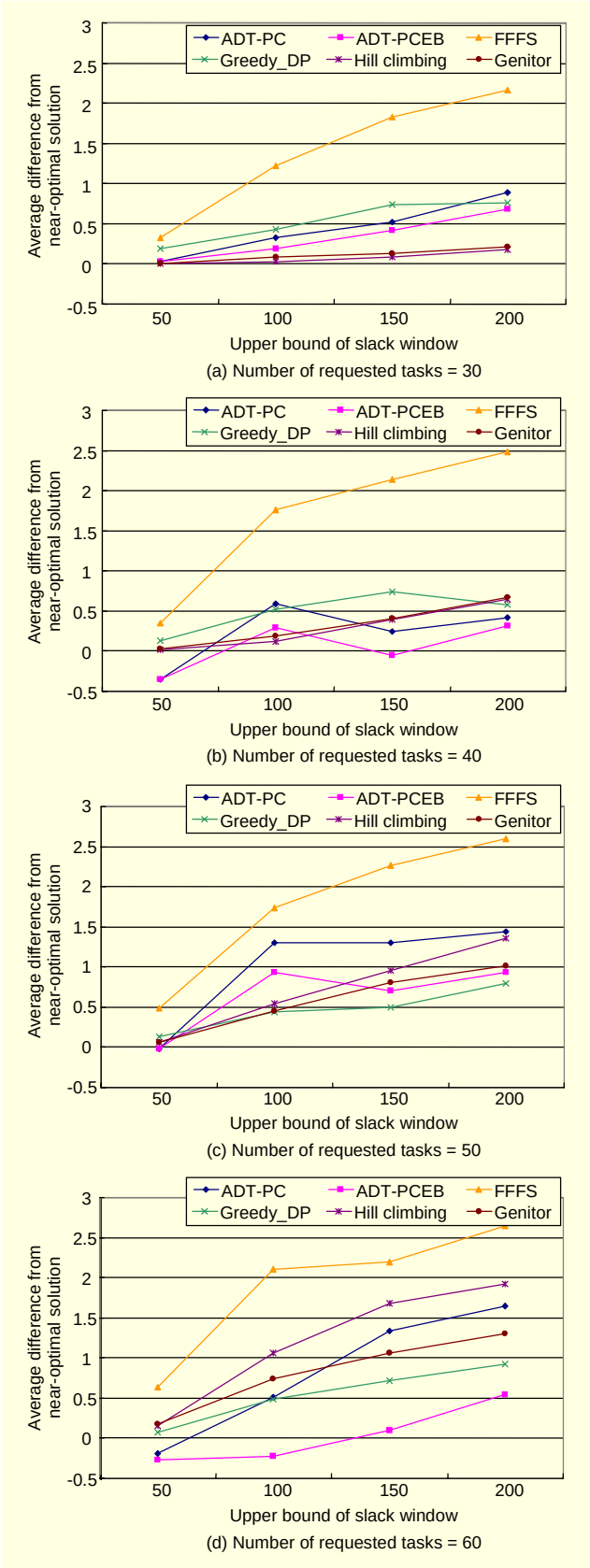

Fig. 12. Average performance difference from near-optimal solution.

Table 2. Evaluation parameters.

Parameter	Value
Max time window	[1, 1440]
Slack window	[0, 50], [0, 100], [0,150], [0,200]
Duration window	[20, 60]
Number of tasks	30, 40, 50, 60

task schedule created by the near-optimal algorithm, namely, the branch-and-bound algorithm [11], which was also used as a measure of optimality in [5].

Throughout Fig. 12, two common characteristics are observed in these experiments. First, ADT-PCEB always shows better performance than ADT-PC. Second, FFFS always shows the worst performance.

When the number of requested tasks is relatively small, that is, 30, local search algorithms, such as hill climbing and genitor, show better performance as described in Fig 12(a). However, as the number of requested tasks becomes larger, that is, 60, the performance of constructive heuristic algorithms such as ADT-PC, ADT-PCEB and Greedy_DP improves as shown in Fig. 12(d). In this case, among the constructive heuristic algorithms, ADT-PCEB shows the best performance.

It is interesting that the performance of ADT-PC and ADT-PCEB sometimes overpasses that of the near-optimal solution as shown in Figs. 12(b) and (d). For example, in the case of ADT-PC, it is beyond the near-optimality when the slack window is [0, 50] and the number of requested tasks is 40 or 60. Likewise, in the case of ADT-PCEB, it is when slack window is [0, 50] or [0, 150] and the number of requested tasks is 40, and when slack window is [0, 50] or [0, 100] and the number of requested tasks is 60. Moreover, the near-optimal algorithm usually takes several minutes or sometimes more than an hour to produce the near-optimal solution with 60 requested tasks, while ADT-PC and ADT-PCEB take just tens of milliseconds with the same number of requested tasks.

References

[1] S. Lee et al., "A Simple and Efficient One-to-Many Large File Distribution Method Exploiting Asynchronous Joins," *ETRI Journal*, vol. 28, no. 6, 2006, pp. 709-720.

[2] S. Lee et al., "Realization of a Scalable and Reliable Multicast Transport Protocol for Many-to-Many Sessions," *ETRI Journal*, vol. 29, no. 6, 2007, pp. 745-754.

[3] J. Chen, "Satellite Mission Scheduling Algorithm," Technical Report, Louisiana State University, 2004.

[4] www.cs.umbc.edu/671/fall01/class-notes/jobshop.ppt

[5] L. Barbulescu, J.P. Watson, L.D. Whitley, and A.E. Howe, "Scheduling Space-Ground Communications for the Air Force Satellite Control Network," *Journal of Scheduling*, vol. 7, Issue 1, 2004, pp. 7-34.

[6] R.J.M. et al, *Parallel Problem Solving from Nature 2*, North-Holland, Amsterdam, 1992.

[7] Y. Caseau and F. Laburthe, "Cumulative Scheduling with Task Intervals," *Proc. Joint International Conference on Logic Programming*, 1996, pp. 363-377.

[8] L.A. Kramer and S.F. Smith, "Task Swapping for Schedule Improvement: A Broader Analysis," *Proc. International Conference on Automated Planning and Scheduling*, 2004, pp. 235-243.

[9] Amotz Bar-Noy, Sudipto Guha, Joseph (Seffi) Naor, and Baruch Schieber, "Approximating the Throughput of Multiple Machines in Real-Time Scheduling," *SIAM Journal on Computing*, vol. 31, no. 2, 2002, pp. 331-352.

[10] http://www.python.org

[11] P. Baptiste, C. Le Pape, and L. Peridy, "Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraints," *Principles and Practice of Constraint Programming*, Springer, 1998, pp. 87-101.

Soojeon Lee received the BS degree in computer science from Korea University, Korea in 2003, and the MS degree in computer engineering from Information and Communications University (ICU), Rep. of Korea in 2005. He has worked as a researcher in Electronics and Telecommunications Research Institute (ETRI) since 2005. His research interests include operation and mission planning of satellite ground control system.

`

Won Chan Jung received the BS degree in computer science at Henderson State University in 1986, and the PhD degree in computer science at Louisiana State University in 1992. He joined ETRI in 1992 and is a principal member of engineering staff. He has been developing the satellite ground control system for KOMPSAT-1 and KOMPSAT-2, and he is currently developing a satellite ground control system for the COMS satellite.

Jae-Hoon Kim received the PhD degree in computer engineering from Chungbuk National University, Cheongju, Rep. of Korea, in 2001. He joined ETRI in 1983, where he was involved in developing the Intelligent Network and KOREASAT Projects. From 1992 to 1994, he was an OJT Engineer in Martra-Marconi Space in the U.K. for the KOREASAT Project. From 1995 to 1999, he participated in the KOMPSAT-1 Ground Mission Control Project as a principle member of engineering staff in system engineering. From 2000 to 2005, he participated in the KOMPSAT-2 Ground Mission Control Project as a team leader. He is now working for the COMS-1, KOMPSAT-3, and KOMPSAT-5 Ground Mission Control Projects as a team leader. His research interests are security in satellite communications, fault diagnosis of satellites using AI technologies, and system modeling using objected-oriented technologies.