Stable Multi-project Scheduling of Airport Ground Handling Services by Heterogeneous Agents

Xiaoyu Mao
Almende B.V.
Westerstraat 50, Rotterdam
3016 DJ, The Netherlands
xiaoyu@almende.org

Nico Roos
DKE, Universiteit Maastricht
P.O. Box 616, Maastricht
6200 MD, The Netherlands
roos@micc.unimaas.nl

Alfons Salden
Almende B.V.
Westerstraat 50, Rotterdam
3016 DJ, The Netherlands
alfons@almende.org

ABSTRACT

This paper addresses decentralized multi-project scheduling under uncertainty. The problem instance we study is the scheduling of airport ground handling services, where aircraft turnarounds can be seen as multiple projects, ground handling services as activities, and service providers as resources. In this environment aircraft requiring ground handling services and the corresponding service providers are self-interested autonomous parties. Moreover, the environment is well-known for its large number of disturbances.

We employ a *heterogeneous* multiagent scheduling framework with two types of autonomous agents representing aircraft and ground service providers respectively. We use *online scheduling* to cope with uncertainty in the release time of project: the uncertainty in aircraft arrival time at an airport. To balance the interests of the two types of agents in this heterogeneous multiagent system, we propose a market-based mechanism to assign time slots to aircraft turnaround activities. We study the use of this mechanism in a *cooperative* and a *non-cooperative* setting.

In a dynamic environment such as airport ground handling, the execution of project schedules may be invalidated by various disruptions. As a result project agents may incur high costs if they have to reschedule some of their activities. The insertion of *slack time* between activities is a well known solution. The delay cost incurred by inserting slack should balance the expected costs of rescheduling some activities. Since in a dynamic multiagent system it is hard to analytically calculate optimal slack time between activities, we propose that agents determine these slack time using a *co-evolutionary* learning approach.

Experiment show that our decentralized scheduling approach scores on average as high as well-established OR-based heuristics, and that slack times to keep a schedule stable can be learned.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*; I.2.11 [**Artificial Intelligence**]: Distributed AI—*Multiagent systems*

Cite as: Stable Multi-project Scheduling of Airport Ground Handling Services by Heterogeneous Agents, Xiaoyu Mao, Nico Roos, Alfons Salden, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 537–544

General Terms

Algorithms, Design, Experimentation

Keywords

Airport Ground Handling, Multiagent System, Uncertainty

1. INTRODUCTION

Aircraft turnaround defines the process of servicing an aircraft while it is on the ground between two connecting flights. Airport Ground Handling (AGH) refers to the planning, scheduling, and control of all aircraft turnarounds at an airport. During turnaround, an aircraft must undergo a whole set of ground handling activities consisting of disembarking/embarking passengers, unloading/loading luggage, maintenance checks, fueling, cleaning, catering, and so on. Many of these activities are handled by independent ground service providers. The interest of the service providers need not be in line with the one of aircraft. This makes scheduling the activities in AGH a challenging task.

An aspect that makes AGH scheduling even more challenging is the highly dynamic environment. The actual arrival time of an aircraft is often different from that foreseen in the original flight plan. Moreover, there are uncertainties during the execution of ground handling activities due to unforeseeable events such as machine breakdown. As a result, ground handling activities may take longer time than expected, invalidating a schedule that optimally assigns time slots to activities (the baseline schedule). Failing to meet the baseline schedule will induce rescheduling costs, which may include resource re-setup cost, inventory cost or various organizational costs.

The scheduling of AGH is an instance of a *multi-project scheduling* problem (MPSP) [12]. Project scheduling is concerned with the allocation of resources over time to perform a collection of activities [5]. In multi-project scheduling, the activities belong to multiple independent projects. In modern day industry, such a multi-project environment seems to be more common than a single-project environment. In the case of AGH scheduling, each aircraft turnaround is a separate project in a MPSP. The ground service providers that perform the activities of the aircraft turnarounds are the resources of the MPSP.

Aircraft and ground service providers at an airport are independent self-interested parties. Aircraft wish to minimize the time of their turnarounds. These objectives need not be in line with the objectives of ground service providers to level their resource utilization, i.e., to avoid peaks of activities.

We propose a *heterogeneous* multiagent system (MAS) in which aircraft and ground service providers are modelled as autonomous agents: *project agents* and *resource agents*, respectively. The projects agents make their scheduling decisions independently of each other, they negotiate with resource agents over the time slots and associated prices for performing their activities. The slot prices depend on the load of the resources while the costs of performing activities for project agents are also related to the flow times of the projects.

In order to handle the uncertainty in project *release time*, we present an *online scheduling* scheme that reduces the risk of re-scheduling by starting the scheduling process of a project when it is released. In online scheduling, projects are scheduled in an incremental way. This may lead to suboptimal schedules compared to an offline approach. To compensate for the inefficiencies that result from pure online scheduling, we investigate a *cooperative* online scheduling scheme. A project agent may reduce the cost of a time slot by cooperating with projects agents that have already reserved the resource for their activities during this time slot.

In a dynamic environment such as AGH, the execution of the project schedules may be invalidated by various disruptions that change the durations of activities. As a result, project agents may incur high costs if they have to reschedule some of their activities. In this context, the addition of slack time between activities is a well known solution. The delay cost incurred by the addition of slack time should balance the expected costs of rescheduling some activities. Since in a dynamic MAS it is hard to analytically calculate optimal slack time between activities, we propose that agents acquire these slack time using a *co-evolutionary learning* method. The empirical results appear to validate our claims.

The remainder of this paper is organized as follows. In Section 2, we provide a formal description of the MPSP in the presence of execution uncertainty. Section 3 describes the MAS solution framework, presents our online multiagent scheduling scenario, and describes the evolutionary learning approach employed by project agents to determine a slack time distribution across its activities. Section 4 analyzes some empirical experimental results. The related work and the contribution of this paper is discussed in Section 5. Finally, the paper is concluded in Section 6.

2. THE SCHEDULING PROBLEM

A MPSP is a generalization of job shop scheduling problem [4] and can be described as follows.

- A MPSP consists of M projects sharing K types of resources;
- Each project $i \in \{1, \ldots, M\}$ consists of N_i activities $a_{i,j}$ with $j \in \{1, \ldots, N_i\}$;
- Two fictitious activities $a_{i,0}$ and a_{i,N_i+1} are added for representing the 'start' and the 'end' of project i;
- Precedence constraints $\prec$ describe execution orders for pairs of activities of the same project: $a_{i,j} \prec a_{i,k}$;
- Each project i has a release time t_i^r, from which activities can be started;
- Each activity $a_{i,j}$ has a non-preemptive processing duration $p_{i,j}$;
- Each activity $a_{i,j}$ requires an amount $r_{i,j}^k$ of resource type $k \in \{1, \ldots, K\}$ for its completion.

A solution to a MPSP is a schedule $S = \{s_{i,j} \mid 1 \leq i \leq M, 1 \leq j \leq N_i\}$ specifying the start time $s_{i,j}$ of each activitie $a_{i,j}$. Not every assignment of start times constitutes a valid schedule. First, the schedule must respect the precedence constraints $\prec$. Second, specific scheduling objectives, such as minimizing the total flow time of projects or the variations in resource utilizations, must be realized.

In the AGH context, the projects in MPSP are managed by independent autonomous parties. These autonomous parties have conflicting preferences concerning the schedule of their own activities. To respect the autonomy of the parties, each with their own private objectives, a decentralized approach is preferred by those parties to a centralized solution concept in which all information is accessible and shared. Hence, we need a MAS approach in which agents negotiate a schedule. Moreover, the ground service providers who manage resources are also autonomous parties having their own preferences. The preferences need not be in line with the preferences of project agents. So the MPSP we are interested in is different from other multi-agent scheduling approaches (e.g., [1, 7, 15, 10]) in the sense that it consists of independent *projects agents* and of independent *resource agents*. The detailed MAS scheduling framework is described in Section 3.

The highly dynamic environment is an important aspect of AGH. This aspect manifests itself as the uncertainties in the project release times and in incidents that influence the duration of activities. As was pointed out in the introduction, we cope with uncertainties in the release times of projects by using an online scheduling approach. That is, we wait for scheduling a project until it is actually released. The choice for online scheduling completely eliminates the release time uncertainties. However, there are potential drawbacks. Scheduling online reduces the amount of available information and it can result in losing opportunities for better allocation of resource to activities. We argue that the dynamics of the AGH domain are so high that it is not warranted to schedule too far beyond a short horizon. For all practical purposes, we can assume that the release times t_i^r for $i \in \{1, \ldots, M\}$ are fixed.

Online scheduling does not offer a solution to incidents influencing the processing durations of activities, unless we wait with assigning time slots to activities until all preceding activities have finished. In AGH the time scale on which services are required by aircraft makes such an approach impractical. The ground service providers need some preparation time in order to provide their services. Therefore we have to model incidents that may influence the processing of activities.

Various types of incidents may happen during the execution of ground services, such as (a) resource inefficiency of ground service provider(s), (b) no show of passengers, (c) breakdown of machineries, etc. All these incident types will influence the execution of related ground services, and will consequently cause the prolongation of the activity processing durations to a certain extent. In the paper we investigate the *resource inefficiency* incident of ground service providers. We use the following incident model:

- A resource type k may subject to an efficiency decrease during a time window $[t_s^{IC}, t_e^{IC}]$;

- the inefficiency factor or efficiency drops is denoted δ ($0 \leq \delta \leq 1$), which literally means that this resource can only serve $1-\delta$ of its full strength;
- activities already scheduled on resource k during this time window (i.e, $\{a_{i,j}|r^k_{i,j} > 0 \wedge s_{i,j} < t^{IC}_e \wedge t^{IC}_s < s_{i,j}+p_{i,j}\}$) will have longer activity processing durations;
- the duration extension of an influenced activity is: $t^{ext}_{i,j} = p_{i,j} \cdot \delta/(1-\delta)$.

3. MULTIAGENT SCHEDULING

We use a multiagent scheduling approach in order to enable the aircraft (project agents) and the ground service providers (resource agents) to consider their personal interests in making scheduling decisions. Each *project agent* PA_i is responsible for a project i consisting of a set of activities, while information about internal precedence constraints among activities of other projects are hidden from them. Activities in these projects need certain resources provided by the resource agents for their completion. Each *resource agent* RA_k is responsible for a resource type k; it receives the resource requirement request concerning a certain activity from a project agent; and it negotiates with the latter project agent the start time of this activity.

Before we can discuss how project and resource agents come to an agreement on the utilization of resources, we first have to discuss the different agent objectives and the cost model of scheduling each individual activity.

3.1 Objectives and Cost functions

Each agent in the multiagent scheduling will try to achieve its own objective. The objective we consider for resource agents is the well-known *resource levelling* objective. Resource levelling strives for minimizing the resource usage variation over time. This objective is often achieved by minimizing the sum of the squared utilization cost [13].

$$f_{RA_k}(S) = c^u_k \cdot \sum_{t=0}^{\infty} u^2_{t,k}(S) \tag{1}$$

in which $c^u_k > 0$ is the resource utilization cost of resource type k per discrete time interval, and $u_{t,k}$ is the utilization of resource k during the discrete time interval labelled by t according to the schedule S. The utilization of resource type k at t is defined as:

$$u_{t,k}(S) = \sum_{s_{i,j} \in S | s_{i,j} \leq t < s_{i,j}+p_{i,j}} r^k_{i,j}$$

In order to create an incentive for project agents to reserve time slots that do not create resource utilization peaks, the measure $f_{RA_k}(S)$ will be used to determine the price of reserving a time slot on a resource.

Each project agent has as its objective the completion of all its activities as early as possible for a reasonable resource price. Project agents are therefore trying to minimize another type of objective functions than resource agents. This function is a trade-off between the makespan of a project and the the costs of reserving time slots on resources for the project's activities. We define the objective function of a project agent PA_i as the sum of project's delay cost, i.e., $c^d_i \cdot dl_i(S)$, and the total resource utilization cost of the project's activities, i.e., $\sum_{j=1}^{N_i} rc(a_{i,j}, S)$:

$$f_{PA_i}(S) \quad = \quad c^d_i \cdot dl_i + \sum_{j=1}^{N_i} rc(a_{i,j}, S) \tag{2}$$

Here, dl denotes the project delay compared to schedule without resource constraints, c^d_i denotes the delay cost per discrete time interval, and $rc(a_{i,j}, S)$ denotes the resource utilization cost of activity $a_{i,j}$ given a schedule S. Let k be the resource type needed by $a_{i,j}$ (i.e., $r^k_{i,j} > 0$), the resource utilization cost $rc(a_{i,j}, S)$ is therefore a price set by resource agent RA_k.

We now discuss how to divide the cost function $f_{RA_k}(S)$ over individual activities requiring resource type k. The online scheduling approach we employ is an incremental approach in which the order of scheduling projects is determined by the release times t^r_i. We label the index of projects by the order of their release times such that, for all $i, j \in \{1, \ldots, M\}$, $i < j$ if $t^r_i < t^r_j$. Furthermore, in order to schedule the activities of a project without backtracking, an activity $a_{i,j}$ can only be scheduled if all preceding activities w.r.t. $\prec$ have been scheduled. It means that activities of a project are also scheduled incrementally. Therefore, when an activity $a_{i,j}$ of project i is to be scheduled, the schedule $S^{i,j-1} = \{s_{1,0}, \ldots, s_{i-1,N_{i-1}+1}, s_{i,0}, \ldots, s_{i,j-1}\} \subseteq S$ of activities $a_{1,0}, \ldots, a_{i,j-1}$ of the projects $1, \ldots, i$ is already fixed. Therefore, a project agent will use the *marginal resource utilization cost* $MC_{RA_k}(a_{i,j}, S^{\leq i,j})$ of resource agent RA_k as the cost $rc(a_{i,j}, S)$ for scheduling $a_{i,j}$ at $s_{i,j}$.

$$MC_{RA_k}(a_{i,j}, S^{\leq i,j}) = c^u_k \sum_{t=s_{i,j}}^{s_{i,j}+p_{i,j}} [u^2_{t,k}(S^{\leq i,j}) - u^2_{t,k}(S^{\leq i,j-1})] \tag{3}$$

Besides deciding on the distribution of the resource utilization cost over activities, we must also decide how to distribute the delay cost of a project over activities causing the delays. The online incremental scheduling approach makes it possible to determine the *marginal delay* $dl^m(a_{i,j}, S)$ caused by scheduling an activity $a_{i,j}$.

$$dl^m(a_{i,j}, S) = est(a_{i,N_i+1}, S^{\leq i,j}) - est(a_{i,N_i+1}, S^{\leq i,j-1})$$

Here, the function $est(a_{i,N_i+1}, S^{\leq i,j})$ denotes the earliest possible start time of the fictitious activity a_{i,N_i+1} given the current schedule $S^{\leq i,j}$. That is, it denotes the *earliest possible finish time* of project i given the current schedule $S^{\leq i,j}$. To summarize, the marginal cost of the project agent:

$$MC_{PA_i}(a_{i,j}, S^{\leq i,j}) = MC_{RA_k}(a_{i,j}, S^{\leq i,j}) + dl^m(a_{i,j}, S) \tag{4}$$

Remark By choosing the same resource utilization cost per discrete time interval c^u_k for all resource types k and the same delay cost c^d_i for all projects, we can view them as weights, especially if we also ensure that $c^u_k + c^d_i = 1$. By varying the resource and delay cost between 0 and 1, we can either emphasize resource levelling or project delays.

3.2 Market-based scheduling mechanism

Based on the cost functions derived in the previous subsection, we now propose an market-based mechanism for scheduling the projects' activities. We describe in the following the communication scenario between project agent PA_i and resource agent RA_k concerning a slot reservation of the activity $a_{i,j}$ on resource type k.

1. Project agent PA_i starts to schedule activity $a_{i,j}$ when all preceding, w.r.t $\prec$, activities of $a_{i,j}$ have been schedule. PA_i sends the resource requirement (including $est(a_{i,j}, S^{\leq i,j-1})$, $p_{i,j}$ and $r^k_{i,j}$) to the corresponding resource agent RA_k.

2. RA_k by receiving this request uses its own pricing model (i.e., marginal cost function (3)) on the resource requirements to calculate a list of offers, starting from $est(a_{i,j}, S^{\leq i,j-1})$; RA_k then sends this list of offers to project agent PA_i.

3. PA_i receives this list of offers, and adds the potential marginal delay cost to these slots prices (i.e., function (4)). it selects the cheapest slot for scheduling activity $a_{i,j}$ and informs resource agent RA_k about its selection.

4. RA_k acknowledges this decision made by PA_i, and schedules $a_{i,j}$ on this selected slot, and keeps listening to the new upcoming requests from project agents.

5. PA_i will move on to schedule the next activity.

With this interactive agent negotiation scheme, we successfully distribute the decision making responsibilities and concerns to individual and different types of agents, and the overall schedule of all projects emerges (see Section 4).

3.3 Cooperative scheduling

In the above scheduling scenario, project agents communicate only the information necessary to make a slot reservation, no additional information is revealed. We denote this mechanism a *non-cooperative* scheduling scheme. In a *cooperative* scheduling scheme, both project and resource agents are willing to share part of their personal information to each other, under the condition that they don't suffer any loss from information sharing. Being cooperative is not contradictory to the selfish nature of individual agents; it appears to be beneficial not only for common good but also for the selfish individuals in a non zero-sum game [2], as in AGH.

To illustrate the cooperative agent scheduling scheme, we introduce the term *secure time window* for each scheduled activity. This secure time window starts from the latest scheduled completion time of this activity's predecessors, and ends by the earliest scheduled starting time of its successors. Any slot shifting within this secure time window will not cause any further delay of the project that the activity belongs to. By knowing the secure time windows of activities, resource agents are given more flexibilities to shift the slots for decreasing the resource levelling value.

Let us illustrate this agent cooperation scheme with an example in Figure 1, in which a time line of resource type k is presented. Assuming that resource agent RA_k managing such resource type and the resource unit utilization cost $c^u_k = 1$. Activity B ($r^k_B = 2$, $p_B = 5$) has been scheduled on this resource at the time window $[3, 8)$, the current resource levelling value is $5 \times 2^2 = 20$. A newly arrived activity A ($r^k_A = 2, p_A = 4$) is requesting the slot price starting from 4 (cf. Figure 1:original). According to the pricing model of resource agents presented in equation 3, the slot price of $[4, 8)$ is: $4 \times (4^2 - 2^2) = 48$ (cf. Figure 1:option 1).

In the cooperative scheduling scheme, project agent PA_B which manages activity B is willing to reveal some additional information to resource agent RA_k. For instance, when all

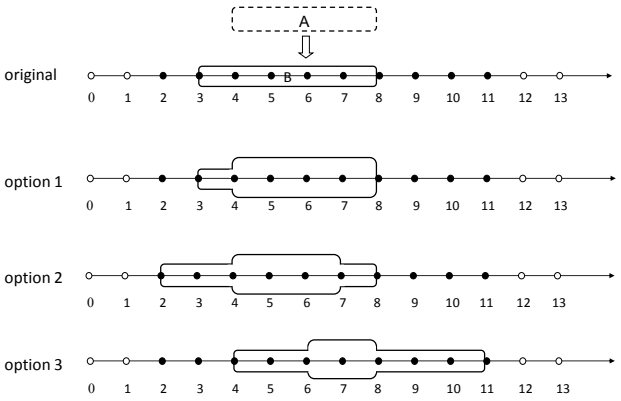

Figure 1: Activity A to be scheduled on a resource

the direct successors of activity B are scheduled (might be on other resources), RA_k will be informed a secure time window of activity B. In Figure 1, the black dots on the resource time line are indicating the secure time window of activity B. Option 2 in Figure 1 illustrates the case where B is shifted backward to $[2, 7)$, and A is scheduled on $[4, 8)$, this shifting will result in a lower total resource levelling: $3\times2^2+3\times4^2 = 60$, therefore, the marginal cost of scheduling activity A is only: $60 - 20 = 40$. Instead of shifting B backward, in Figure 1 option 3, B is shifted forward to $[6, 11)$, which gives an even lower marginal cost of scheduling activity A: $5 \times 2^2 + 2 \times 4^2 - 20 = 32$.

In this cooperative scheduling scheme, by providing more information to resource agents, project agents are not suffering any loss but helping resource agents for a lower levelling value. Subsequently, cheaper slot options are provided to other project agents.

3.4 Learning slack time to handle incidents

So far we have presented a heterogeneous MAS for scheduling AGH services. This multiagent scheduling approach is capable of handling uncertainties in the release times of projects (the aircraft's arrival times) but not in handling incidents that influence the normal processing durations of activities (the ground handling services).

In the domain of AGH the execution of ground handling services can be disrupted by events such as no show of passengers, resource inefficiency, machine breakdown, and so on. As a result the processing durations of activities may increase. The increase of some of these processing durations can invalidate the schedule of other activities in the same project, forcing the project agent to reschedule its activities. The cost of rescheduling can be high since resource capacity may already be reserved by other projects. A well known solution to this problems is the insertion of slack time between activities. This slack time must guarantee that in case an incident occurs during one activity, succeeding activities can still start as planned. Of course, the insertion of slack time between activities comes with a price — higher delay cost for the projects.

An important question that the project agents have to address is how much slack time should be added after each activity. If a project agent adds too much slack time, the delay cost will become unnecessarily high. If the project

agent adds not enough, it may have to reschedule some of the activities and incur high resource cost because of that. Thus, the project agent has to balance the delay cost of adding slack time and the expected resource rescheduling cost when the slack time is insufficient in some cases. In other words, the project agent has to minimize the sum of these two costs.

Analytically determining the optimal slack time to be inserted after an activity is an impossible task. In order to fulfill this task agents must know not only the frequency with which incidents occur, but also the resource requirement of other project agents. To make things worse, other project agents may at the mean time be pursuing their own optimal slack times. Hence, determining optimal slack time becomes a strategic game with no prior information about the project agents' payoff matrix. By playing the game the agents can, however, gain some information about their own rewards, although these rewards still depend on the employed strategies by the other agents.

Many multiagent learning techniques require that the payoff matrix is known by the agents. Techniques that can still be used when the payoff matrix is not known in advance are for instance Evolutionary Game Theory [17], Nash Q-Learning [6] and Genetic Algorithms using co-evolution [14]. In all cases convergence to a stable state is an important issue. The authors have chosen to use Genetic Algorithms (GAs) because (i) we are interested in a proof of concept, i.e., can appropriate slack times be learned, and (ii) we expect GAs to be less sensitive for requirements that must be met to guarantee convergence.

We employ a GA learner within each project agent. Therefore, project agents are co-evolving their individual strategies in a multi-project scheduling environment. In our co-evolutionary learning, we assume that project agents use the same problem specific encoding for an individual I of the populations used by their GA. An individual I_i of project agent i is represented by a vector of N_i real numbers and is denoted as:

$$I_i = (\alpha_{i,1}, \ldots, \alpha_{i,j}, \ldots, \alpha_{i,N_i}) \tag{5}$$

The scaling factor $\alpha_{i,j} \in [0,1]$ in I_i determines the slack time $t_{i,j}^{slk}$ to be inserted behind activity $a_{i,j}$. The slack time $t_{i,j}^{slk}$ can be computed as the product of $\alpha_{i,j}$ and the activity processing duration $p_{i,j}$: $t_{i,j}^{slk} = \alpha_{i,j} \cdot p_{i,j}$[1]. Each individual is thus transformed to a uniquely determined slack time configuration.

In order to determine the fitness value $f(I_i)$ of each individual I_i, we need to run simulations with incidents disrupting the execution of certain activities. While making scheduling decisions, each project agent will take the slack time configuration into account. When an incident occurs during the execution of activity $a_{i,j}$, and causes a duration extension $t_{i,j}^{ext}$, three different situations may arise:

1. $t_{i,j}^{ext} \leq t_{i,j}^{slk}$;
2. $t_{i,j}^{ext} > t_{i,j}^{slk}$ but none of the succeeding activities' schedule is invalidated;
3. $t_{i,j}^{ext} > t_{i,j}^{slk}$ and some schedules of succeeding activities are invalidated.

[1]We restrict the length of slack time to be less than the activity processing duration. This constraint can be relaxed by increasing the upper bound of α value.

PA_i handles the first two situations by simply requiring additional resources for the extended period. For the third situation, PA_k does a complete rescheduling of the rest unprocessed activities.

The fitness of an individual is determined by the sum of the *actual* project delay cost $c_i^d \cdot dl_i^{IC}$ and the *actual* resource utilization costs of all activities $\sum_{j=1}^{N_i} rc^{IC}(a_{i,j}, S)$ when the project is completed.

$$f(I_i) = c_i^d \cdot dl_i^{IC} + \sum_{j=1}^{N_i} rc^{IC}(a_{i,j}, S) \tag{6}$$

GA starts by computing an initial population, i.e., the first generation, containing individuals with random α values. After computing the fitness of all individuals by simulation, we can apply the usual genetic operators such as selection, crossover and mutation. By scheduling ground handling services repeatedly, GA whould create new generation of individuals that represent improved slack time configurations.

4. EXPERIMENTAL RESULTS

The performance of our proposed online multiagent scheduling approaches are first evaluated in a deterministic environment. We compare the generated schedules with those of some well known priority-based centralized heuristic approaches. The comparison has been made for both the case where project agents are non-cooperative and cooperative. The tests have been performed in an airport specialized MPSP (see Section 4.1).

Next we simulate a dynamic airport situation where certain service providers may suffer some resource inefficiency incidents during a period of time. Slack time needs to be added for activities related to such incidents in order to absorb the disruptions. We studied in this dynamic environment the performance of the genetic learning method employed by each project agent for approximating an optimal slack time distribution (see Section 4.2).

4.1 Experiments for deterministic problems

We can distinguish a few types of aircraft turnaround procedures based on the differences in aircraft models (Boeing 747, Airbus 320, etc), aircraft usages (cargo or passenger aircraft) and docking locations (terminal gate or remote stand). Different turnaround procedures may require different sets of ground handling activities. For instance, the precedence relations among activities may be different; same type of activities may require different types and amounts of resources; moreover, activity processing durations may also be different in different turnaround procedures. However, ground handling activities performed in the same type of turnaround procedure share a large scale of similarities.

Based on these observations, in our experiments, we have simulated an airport environment with 10 types of aircraft turnaround procedures. For each type of these procedures, we have created 10 identical aircraft instances. Therefore, we have a simulated airport environment with 100 aircraft requiring ground handling services. Aircraft arrive at the airport with a random order and with a frequency of every 5 minutes (release frequency).

Such an airport specialized MPSP instance is constructed by using 10 single-project instances ($J301\_1 \rightarrow J301\_10$)

from the benchmark Project Scheduling Problem Library — PSPLib [8]. These 10 project instances stand for 10 different types of aircraft turnaround procedures. Each of project instances consists of 30 activities, each of which can be used to represent a ground handling service and requires one of the 4 types of resources for execution. These shared 4 types of resources can be considered as ground handling service providers. The optimal schedules without resource constraints for these 10 projects range from 31 to 60, which resembles an aircraft turnaround duration in actual situation.

Schedules are made in a deterministic environment and the proposed two multiagent scheduling approaches are compared with the following three priority-rule based centralized heuristic methods.

- First Come, First Served (FCFS),
- Maximum Total Work Content first (MAXTWK),
- Shortest Activity from the Shortest Project (SASP).

The later two heursitics are proved to be more effective compared to other heuristics when the objective is to minimize the mean project delay in MPSP [9]. In the centralized heuristic methods, the differences in three methods are the priority rule(s) applied in selecting an eligible activity. For deciding the activity start times, they all use the same heuristic — choose the start time resulting in the minimum combined marginal cost function (4). By decreasing gradually the value of c_i^d and therefore increasing c_k if $c_i^d + c_k^u = 1$, the problem will switch from a time-optimization problem to a resource-optimization problem.

In multiagent scheduling methods, resource agents use the marginal resource cost function (3) as their pricing model. We achieve switching from reducing *project delay time* to *resource levelling* by gradually decreasing the delay cost per time units c_i^d. Recall that project agents are concerned about the delay incurred by scheduling an activity. Choosing a high c_i^d value emphasizes a time-based objective, while a low c_i^d stresses a resource-based objective.

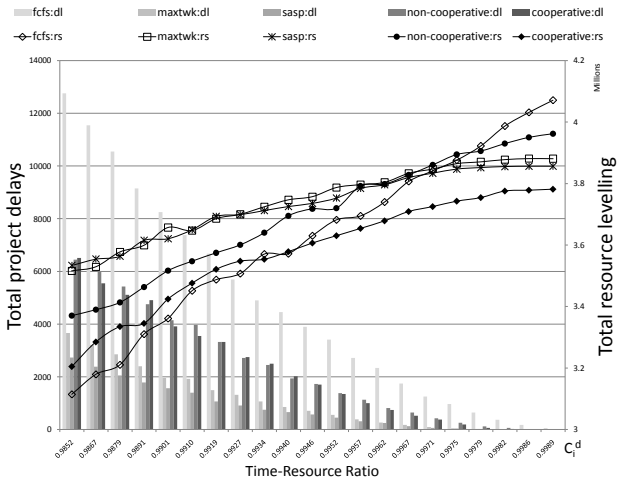

Figure 2: Various approaches in solving AGH problem with bi-criteria objectives

We demonstrate the experimental results in two different perspectives with two figures. Figure 2 shows the trends of total project delay (cf. left y-axis and bars in the diagram) and total resource levelling (cf. right y-axis and lines in the diagram) with respect to the change of emphasis from resource-optimization to time-optimization (x-axis). For all five scheduling approaches, it holds that with the increase of delay cost per time unit c_i^d, the total project delay decreases, while the resource levelling value increases.

Comparing the five scheduling methods, from the time perspective, we observe that MAXTWK and SASP heuristic perform the best; and our online multiagent scheduling approaches (both non-cooperative and cooperative) lead to a sub-optimal results compared with these two heuristics. They only outperform the naive FCFS heuristic. However, with respective to the resource levelling measures, multiagent scheduling in non-cooperative scheme results in lower resource levelling than MAXTWK and SASP heuristics when the resource-based objective is emphasized (a relatively low c_i^d). The total resource levelling is even significantly improved in the multiagent cooperative scheduling scheme.

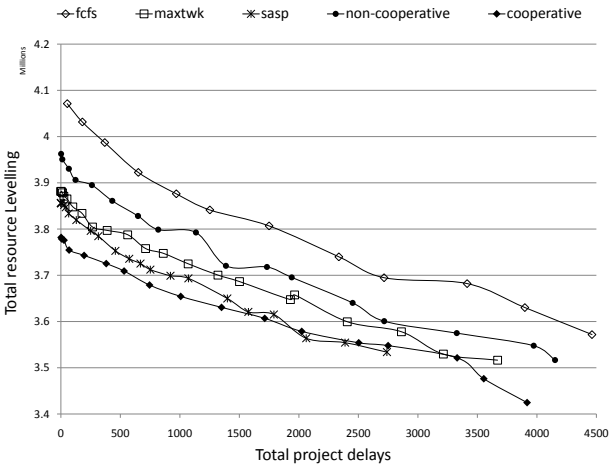

Figure 3: Trade-offs between time and resource

Figure 3 shows the trade-off between project delays and resource levellings. In all solution methods, the higher total project delay goes (cf. x-axis), the less total resource levelling value (cf. y-axis) will be paid. Non-cooperative online multiagent scheduling approach provides a schedule that is of comparable quality as the centralized heuristic methods. Moreover, by letting self-interested agents behave cooperatively, given the same total delay, we can even lower the resource levelling value. Therefore, with the same project delays, cooperative online agent scheduling approach can achieve the least resource levelling value, which means that it outperforms all other heuristics w.r.t the total cost of the projects.

4.2 Experiments for dynamic problems

In order to evaluate the proposed co-evolutionary learning scheme for assigning slack time distribution across ground handling activities, we have simulated a dynamic problem instance. In such a dynamic situation one of the ground service providers (resources) will suffer an incident of *resource inefficiency* ($\delta = 0.2$) within a certain period of time (30 minutes in our simulation). In consequence, all aircraft

ground handling activities that have been scheduled for using such a resource during this time period will have to extend their processing durations.

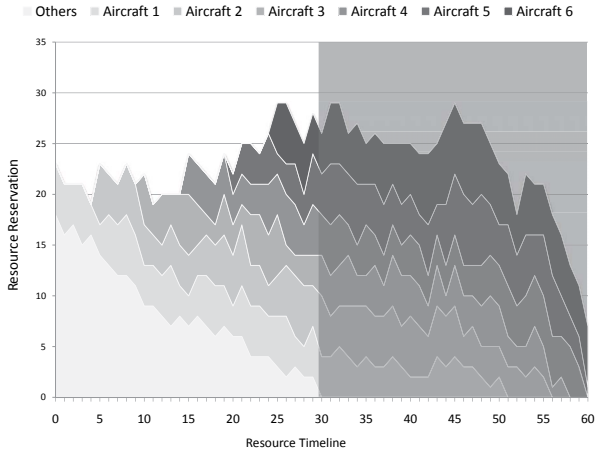

Figure 4: Incidents due to resource inefficiency

Figure 4 shows the concept of this simulated situation, where the timeline shows a schedule fragment of this problematic service provider. Multiple aircraft have already made their reservations for using such a resource during this time frame. Within this time frame, 6 aircraft were released sequentially with a release frequency of every 5 minutes. When the time arrives at the point 30, the simulated incidents occur, causing a drop in efficiency of the service provider 20% of its full strength. This incident renders the original scheduled activity processing duration insufficient for completion. Project agents try to insert slack time in between activities to avoid rescheduling. They do so by "co-evolving" the slack time distributions as proposed in section 3.4.

We implemented our genetic learning algorithm used by each project agent with two different parameter settings — *Dejong Settings* [3] and a customized parameter setting:

- population size: 150
- tournament selection size: 5
- chance of crossover: 0.6
- crossover type: uniform
- mutation type: Gaussian mutation
- mutation rate: 0.01

Figure 5 shows the learning curves of all 6 aircraft with two sets of parameters. The horizontal axis represents and number of generations and vertical axis shows the average fitness value of each generation. As observed in Figure 4, later arrived aircraft (such as aircraft 5 and aircraft 6) have more activities that fall into this incident zone than other aircraft. The learning results reflect the consequence of these incidents by the difference in fitness levels: later arrived aircraft suffer more disturbance than earlier ones. These learning curves show that such online co-evolutionary learning methods is capable of readily absorbing the uncertainties in the execution of the project activities and converge rapidly to a stable situation. Comparing two genetic algorithm parameter settings, the customized setting performs slightly better than the standard "Dejong Settings" in the sense of

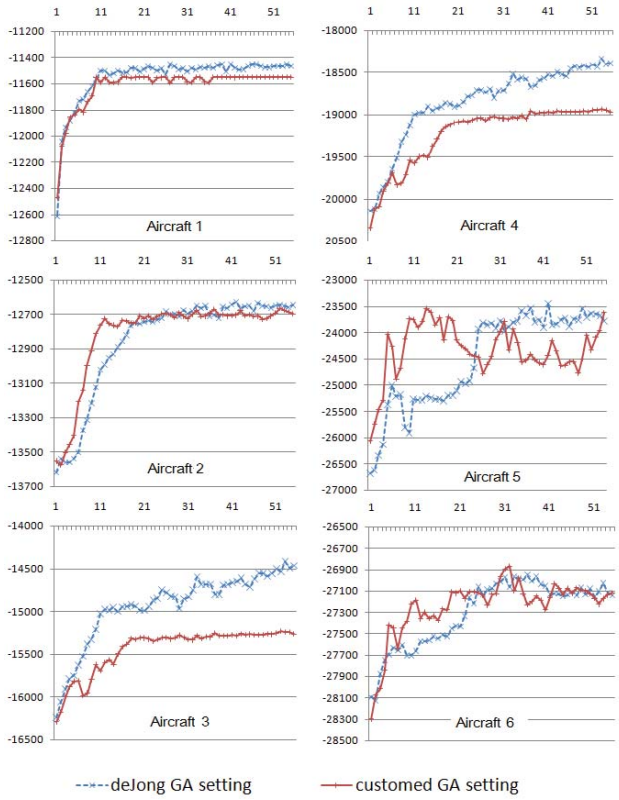

Figure 5: The learning curves of aircraft

stability and speed of convergence. The parameter setting for genetic algorithms influences the performance of learning significantly. Due to the fact that the chromosome of each individual in our genetic encoding consists of 30 real values, a relatively large population size would give the possibility of a wider search radius in a more explored search space to avoid local optima. Parameter refinement in co-evolutionary learning can be found in [14].

5. RELATED WORK

Many works published in the area of project scheduling make reference to the scheduling of a single project under resource limitations and time constraints [13, 4]. Scheduling multiple projects starts to draw more and more research attention both in OR and AI communities [12, 10, 1].

Using multiagent systems for solving decentralized scheduling problems is observed in recent research [15, 7, 10, 1]. In the project scheduling field, Lee et al. [10] have proposed a MAS for short-term rescheduling of resources shared by multiple projects. Confessore et al. [1] proposed a multi-agent model and a combinatorial auction mechanism for scheduling a multi-project problem. In both of their works, a coordinator agent is proposed for solving the resource conflicts by allocating the shared resource time slots to project agents. Our heterogeneous MAS model differs from these work in the sense that different types of resources, having each their own interests, are also modelled as agents. These resource agents make in turn their decisions based on their own interests. These decisions may conflict with those of the project

agents, which are subject to their own objective functions that are influenced by those of the resource agents. Allowing different types of agents to cooperate by sharing additional private information helps both resource and project agents to attain higher objective function levels.

Methods for generating a stable baseline schedule have been proposed for scheduling single project in the presence of activity disruptions. We observe two heuristic methods which add slack time *in front of* activities to protect the activity starting times. *Adapted Float Factor* (ADFF) heuristic [11] generates a stable baseline schedule under the condition of ample resources, and *Resource Flow-Dependent Float Factor* (RFDFF) heuristic [16] extends ADFF for the problem with limited resource capacities. These approaches scatter the time buffer for project makespan through out the baseline schedule. Our method is different from these two heuristics in two perspectives. First, our method inserts slack time at the end of activities instead of in front of them. This creates an incentive of associating longer slack time to those activities that have a higher probability of being disrupted. Second, our method allows the project agents to acquire this slack time distribution by means of co-evolutionary learning under a dynamic situation with disturbance during activity execution.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a heterogeneous MAS solution framework for scheduling AGH services in a dynamic environment. This AGH scheduling problem forms an instance of decentralized MPSP under uncertainty. A proposed market-based mechanism in such MAS for negotiating an acceptable resource time slot provides heterogeneous agents the possibility of making independent decisions based on their own objectives and preferences. An online scheduling scheme completely eliminates the risk of rescheduling under the uncertainties of aircraft arrival times. In the experiments, we observe that such an incremental scheduling scheme leads to a sub-optimal solution compared to some offline heuristic approaches. We try to compensate the inefficiencies resulted from the online scheduling by letting agents behave cooperatively — revealing additional information to other agents. The experimental results of such cooperative online scheduling scheme show a comparable performance as the one of the best centralized scheduling heuristics.

In addition, inserting slack time at the end of every activity has been chosen to be the method of absorbing the possible disruptions caused by incidents. In order to fulfill the task of assigning appropriate slack time for all activities under certain incidents, a genetic learning algorithm is employed by each project agent to approximate the stable situation, multiple agents are therefore co-evolving the solution in a shared environment. The empirical experiments we conducted in one incident situation show that our online co-evolutionary learning method is capable of readily absorbing the uncertainties in the execution of the project activities and converge to a stable situation.

Further research we envision is needed to empirically model in terms of agents the utilization costs of dynamic and finite-capacity resources of service providers; flow time delay costs and resource usage price for different types of aircraft; probability and severity of the different incident classes. As a consequence our MAS solutions will acquire upon evolutionary learning or other learning methods for the different incident classes.

7. REFERENCES

[1] G. Confessore, S. Giordani, and S. Rismondo. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150:115–135, 2007.

[2] R. Dawkins. *The Selfish Gene: 30th Anniversary Edition — with a New Introduction by the Author*. Oxford University Press, 2006.

[3] K. A. de Jong and W. M. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In H.-P. Schwefel and R. Männer, editors, *PPSN*, volume 496 of *Lecture Notes in Computer Science*, pages 38–47. Springer, 1990.

[4] E. L. Demeulemeester and W. Herroelen. *Project Scheduling — A Research Handbook*. Springer, 2002.

[5] U. Dorndorf. *Project Scheduling with Time Windows: From Theory to Applications*. Physica-Verlag, 2002.

[6] J. Hu and M. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[7] G. Knotts, M. Dror, and B. C. Hartman. Agent-based project scheduling. *IIE Transactions*, 32:387–401, 2000.

[8] R. Kolisch and A. Sprecher. Psplib — a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1997.

[9] I. S. Kurtulus and E. Davis. Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172, 1982.

[10] Y. Lee, S. R. T. Kumara, and K. Chatterjee. Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligence Manufacturing*, 14:471–484, 2003.

[11] R. Leus. *The generation of stable project plans*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2003.

[12] A. Lova and P. Tormos. Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. *Annals of Operations Research*, 102(1-4):263–286, Feb 2001.

[13] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2nd edition, 2001.

[14] J. Paredis. Coevolutionary algorithms. In T. Bäck, D. B. fogel, and Z. Michalewicz, editors, *Evolutionary Computation 2: Advanced Algorithms and Operators*, pages 224–238. Institute of Physics Publishing, Bristol, UK, 2000.

[15] K. Sycara, S. Roth, N. Sadeh, and M. S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, Nov./Dec. 1991.

[16] S. van de Vonder, E. Demeulemeester, W. Herroelen, and R. Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2):215–236, January 2006.

[17] J. Weibull. *Evolutionary Game Theory*. The MIT Press, 1997.