Dynamic Scheduling in Systems with Complex Resource Allocation Requirements

Junchao Xiao[1,2], Leon J. Osterweil[2], Qing Wang[1], Mingshu Li[1,3]
[1]Laboratory for Internet Software Technologies, Institute of Software,
Chinese Academy of Sciences, Beijing 100190, China
{xiaojunchao,wq,mingshu}@itechs.iscas.ac.cn
[2]Department of Computer Science University of Massachusetts, Amherst, MA 01003-9264 USA
ljo@cs.umass.edu
[3]Key Laboratory for Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100190, China

Abstract

Real world systems can be thought of as structures of activities that require resources in order to execute. Costs are reduced when resource levels are kept low, but this can lead to competition for resources that can cause poor system performance. Careful allocation of resources can improve system performance by enabling more efficient use of resources. This paper proposes that resource scheduling be done in a series of dynamic reschedulings that use precise and detailed information about the system and available resources to improve the quality of scheduling results. Each rescheduling is done over a relatively small set of activities, and a genetic algorithm is used to support scheduling computations. Simulations of healthcare systems are used to evaluate this approach. Results indicate that this approach can support effective dynamic rescheduling at affordable costs.

Categories and Subject Descriptors

D.2.9 [**Management**]: Software process models; I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling

General Terms

Algorithms, Management, Performance, Human Factors.

Keywords

Incremental resource scheduling, genetic algorithm, process simulation, healthcare process analysis

1. Introduction

Complex systems are central to more effective performance in many different domains such as healthcare, manufacturing, and software development. Such systems are typically comprised of a group of activities, each of whose executions requires different resources that have various capabilities and availabilities. Because resource availability is usually limited, resource contention problems often arise during system execution, sometimes leading to delays and inefficiencies. Careful resource scheduling can reduce or eliminate resource contention, thereby reducing delays and inefficiencies, and increasing the value the system is able to obtain from a given configuration of resources [2, 30].

A lot of work has investigated different approaches to determining optimal schedules of assignment of resources to system activities. One approach is static resource scheduling, in which a complete schedule of resource assignment is computed in advance based on advance knowledge of the sequence of activities to be performed and the size and duration of all these activities [6, 7, 22]. However, the world is a very dynamic place, and there is a great deal of uncertainty about the future course of the execution of any realistic system. Uncertainties such as the sudden arrival of rush orders, unexpectedly slow activity performance, and unexpected lack of resources [11, 20] all change the execution environment creating the potential for consequent schedule disruptions [12].

In response to the problems posed by the inevitability of such uncertainties, researchers have studied different kinds of dynamic resource scheduling approaches, including reactive scheduling, proactive scheduling, etc. [12] These methods typically attempt to schedule only the activities that are within a restricted part or phase of system execution. They may use extensive or exhaustive searching approaches to compute an optimal or near-optimal schedule of resource utilization for this reduced set of activities. This approach has some limitations, however. Most obviously, the scale of the scheduling effort can still be quite large if the schedule is to cover an extensive part of the system's activities. In addition, dynamically occurring events may still render the assumptions of the scheduling effort invalid, necessitating a rescheduling (this is especially problematic as the part of the system being scheduled becomes large).

These issues seem particularly troublesome in some fast changing environments such as the healthcare domain, where patient care systems must continually adapt in response, for example, to new patient arrivals and medical emergencies. This suggests the desirability of finding new ways to mitigate the problems that seem inherent in the incremental rescheduling approach. Our work investigates an approach that exploits detailed specifications of the activities, their needs for resources, and the characteristics of the resources themselves in order to achieve better resource scheduling results. Our approach decomposes the overall resource scheduling problem into a series of dynamic reschedulings happening at selected times and covering sets of activities for which access to detailed information could be the basis for more effective resource schedules. To pursue this we have explored:

(1) Using very complete and precise information about system activities and resources. This should provide scheduling schemes that produce results that are of higher quality, and

that should remain accurate over most or all of the activity set for which resources have been scheduled.

(2) Keeping the activity set for which resources are to be scheduled relatively small thereby keeping analysis costs relatively modest and enabling relatively quick response to changing execution environment conditions.

(3) Enabling dynamic changes in successive reschedulings. Earlier resource allocation decisions and unexpected events can alter the choice and importance of later activities, thereby affecting how resources might be allocated to them. Thus we explored using scheduling parameters (e.g. the constraint sets being used in scheduling) that may vary to make it easier to recognize and compensate for the effects that previous activities may have on resource allocation decisions for upcoming activities.

This paper explores these approaches by proposing a time window based incremental resource scheduling method. In this method, resource scheduling and rescheduling is performed incrementally at selected points during system execution. Our approach relies upon the availability of detailed specifications of both system activities and resources that are provided by using well-defined languages capable of supporting specifications that are both very precise and very detailed. This causes the characteristics and behaviors of the activities in the window, and the characteristics of the resources allocated to those activities, to be relatively predictable. Our expectation is that this should help us to generate results that are close to optimal. Though relatively small, we expect our rescheduling windows to still be large enough to contain quantities of activities and resources that are sufficiently large to require considerable scheduling computation. To address this problem we use a genetic algorithm (GA) [13] as the basis for our scheduling approach. In addition to its speed, another advantage of the GA approach is that it is capable of readily incorporating many types of constraints into the definition and solution of the scheduling problem.

This approach has been evaluated by running simulations of processes that define the ways in which health care systems are deployed and used. These simulations use different details of processes and resources, different constraints, and different GA parameters to compute different resource allocation schedules. The results obtained suggest that this approach shows promise of being a valuable system for supporting resource allocation.

The paper is organized as follows. Section 2 describes some related work. Section 3 presents our time-window based incremental scheduling method. Section 4 presents some details of the components and technologies used. Section 5 describes a simulation of a process in the healthcare domain and reports on some case studies aimed at evaluating this approach. Section 6 summarizes the observed benefits of the approach, and Section 7 presents conclusions and suggests future work.

2. Related work

The objective of resource scheduling is to achieve the maximum value (such as the shortest system execution time or the greatest system throughput) from a configuration of resources [4]. Resource scheduling research investigates two main approaches: static scheduling and dynamic scheduling. But the key assumption of static scheduling, namely that the execution environment is relatively fixed over the entire execution of a system [6] does not hold in domains such as healthcare [23, 29], manufacturing [15, 19, 26], software development [2, 3], and most other resource-constrained domains [10, 21]. Uncertainty about the key parameters needed to support resource scheduling is a major concern in resource scheduling in such domains [18]. Indeed dynamic change during system execution creates an environment of uncertainty [4] that can give rise to such difficulties as the need to address changing, and even conflicting, goals.

To address dynamic change in uncertain environments, researchers have proposed two dynamic scheduling approaches: reactive scheduling and robust scheduling [12]. Reactive resource scheduling deals with uncertainties arising during system execution by carrying out a complete or partial rescheduling as soon as unexpected events or other uncertainties are recognized [25, 31]. Reactive scheduling seems effective in addressing some rescheduling problems, but its effectiveness is reduced when activity estimates are unreliable, uncertainties are numerous, and when it attempts to reschedule large numbers of activities. Under such circumstances rescheduling may take a considerable amount of time, yet still necessitate frequent new reschedulings.

Robust scheduling aims to anticipate the effects of possible disruptions while still generating schedules that support a high level of performance [1, 9, 18, 27]. The results of robust scheduling are most effective when there are limited and predictable disruptions in system executions. If actual disruptions exceed expectations, rescheduling is needed. This approach should benefit greatly from access to system specifications that are as clear, complete, and as precise as possible about system execution disruptions. Our own work adopts this approach.

Considerable research has also addressed the need for good scheduling algorithms. Resource scheduling problems have high complexity time bounds, where even relatively simple heuristics have been shown to be NP-hard [24]. The genetic algorithm (GA) [13] approach has often been adopted in resource scheduling [8, 14]. Because GA approaches are heuristic, they cannot guarantee optimal, or even near optimal results, and so much attention has been directed to seeking appropriate parameters and evolution methods that improve convergence and avoid local maxima.

Finally we note that simulation seems to be a popular and effective method for evaluating scheduling approaches [7, 16, 17], and indeed we also have evaluated our approach by applying it to simulations of processes that define the use of complex systems.

3. Incremental Resource Scheduling Method

Our approach to resource scheduling combines the strengths of the robust incremental scheduling approach and the GA technology, with the exploitation of more complete and precise information about uncertainty that we derive from the analysis of a particularly detailed and precise definition of both the system being executed and the resources available for allocation. Currently our incremental rescheduling is carried out at fixed points in time. However, this approach also lends itself to support rescheduling either 1) reactively, when events occur that are beyond the scope of what we have been able to anticipate, or preferably, 2) proactively, at time points that may be dictated by recognition of upcoming uncertainty derived from analysis of precise and detailed system definitions. Each incremental rescheduling activity covers only the tasks that will occur within a specified time window. A key goal of our research is to study how to determine the optimal size of this window. If the window is too small more frequent (but perhaps more accurate), reschedulings may be needed. If the window is large, scheduling may be less frequent, but scheduling cost may be high, and accuracy low.

Determining the right window size and scheduling approach is facilitated by the availability of a system definition specification that contains clear indications of such uncertainties as locations of exceptions, possibilities for human decision-making, and the idiosyncrasies of execution agents. This information is used in the design of GA chromosomes that are more completely and precisely specified, thereby standing a greater chance of converging on more optimal results at lower cost.

The architecture of the incremental time-window rescheduling system that we have built is shown in Figure 1, which shows the following major components:

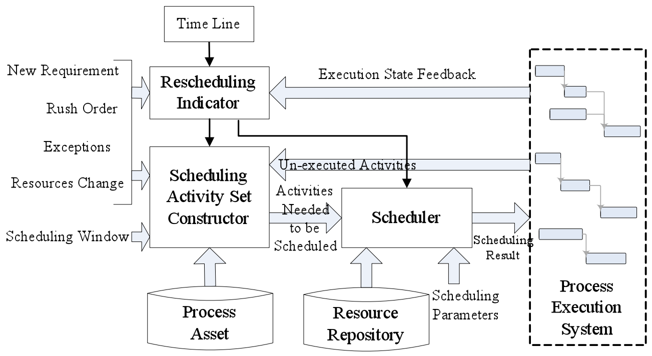

Figure 1. Incremental Resource Scheduling Framework

- **Scheduling activity set constructor.** This component assembles the rescheduling problem, which consists principally of a specification of the activities that may possibly be executed in the near future, their resource requirements, and a specification of the resources that are available for use by those activities. The scheduling activity set constructor relies upon a specification of the system, and a repository of specifications of available resources.
- **Scheduler** component, which uses the output of the scheduling activity set constructor and a Genetic Algorithm (GA) to identify the specific resources to be used to support the execution of each activity.
- **Rescheduling indicator** component, which determines when rescheduling should be done. Rescheduling is triggered when the rescheduling indicator determines that execution is about to proceed past the window over which the last rescheduling had been computed. This component could also be used to identify when certain types of unexpected events, such as low-probability exceptions, sudden unavailability of resources, and unexpectedly long task execution times occur, making rescheduling desirable or necessary.
- **System execution** component, which provides execution events needed to update the system execution state upon which the rescheduling indicator and the scheduler rely.

We now describe the system we built in order to evaluate our approach and this architecture.

4. The System Used for Our Evaluation

4.1 Process Activity Definition

To enable us to evaluate one of our central research hypotheses, namely that a more complete, precise, and detailed system definition can improve the quality of the resource scheduling approach, we have chosen a powerful process definition language, Little-JIL, with which to define the processes that use the system for which we will do our scheduling. Little-JIL [5, 28] is a language that was originally developed to support the definition of the processes by which software is developed and maintained. More recently it has been used to define processes that are central to systems in such domains as healthcare, government, and science. Wise [28] provides full technical details of the language. Here we outline its salient features that seem to be of most relevance to our work in resource scheduling.

A Little-JIL process definition consists of a specification of three components, an artifact collection (not described here due to space constraints), an activity specification, and a resource repository. A Little-JIL activity specification is defined as a hierarchy of steps, each of which represents an activity to be performed by an assigned resource (referred to as its agent). Each step has a name and a set of badges to represent control flow among its sub-steps, its interface, the exceptions it handles, etc. A leaf step (one with no sub-steps) represents an activity to be performed by an agent, without any guidance from the process. Each step specification also contains a collection of resource requests. Each request in the collection is described by the following definition.

Definition 1. $Req = (ResName, Capability_1, SkillLevel_1, ..., Capability_r, SkillLevel_r)$,

where,

- $ResName$ is the type of the resource being requested, (e.g. doctor, nurse, bed), which indicates the kinds of capabilities that this resource has.
- $Capability_i$ is a capability that is being requested.
- $SkillLevel_i$ is the minimum level of skill in $Capability_i$ that is being requested.

Figure 2 shows a Little-JIL activity definition that defines a process by which a single patient is treated in a typical hospital Emergency Department. Note that this process is instantiated for every new patient, and thus the workings of an actual ED are represented by the concurrent execution of several of these processes. Each process needs the same types of resources, which must be provided by one central resource repository. This sets up resource contention. The process is defined only at a very high level due to space limitations. The entire process is represented by the top step, "*TreatOnePatient*", whose three substeps provide elaborative detail about how a patient is treated. Note that this example process is kept rather small and simple due to space limitations. But a more complete process definition would show the use of such more powerful language features as specifications of concurrency, the throwing and handling of exceptions, step kinds that allow human agents to make choices, and pre- and post-requisites that function as guards for the performance of steps.

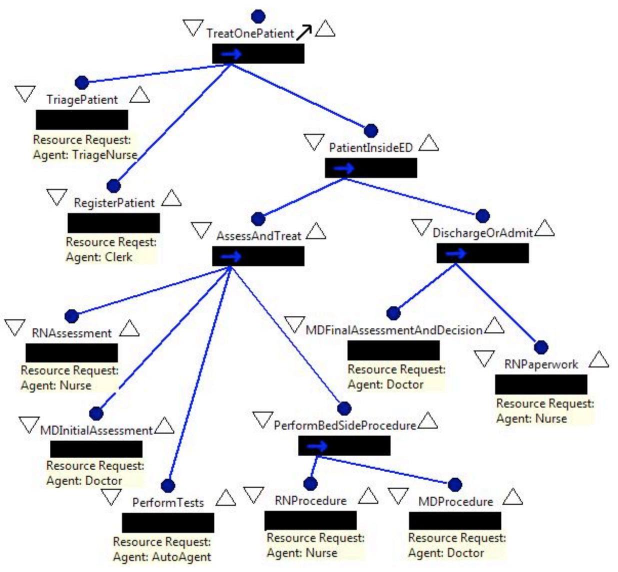

Figure 2. Process described by Little-JIL

In Figure 2 the right arrow in "*TreatOnePatient*" specifies that, in sequential order, the ED patient is first triaged by a triage nurse, then registered by a clerk, and then placed in a bed for assessment and treatment. This last step is further decomposed into two sequential substeps, each of which is decomposed still further. Not shown in this diagram (again due to space limitations) are the resource requirements for each step, and other process definition details some of which will be elaborated upon below.

4.2 Resource Repository

The resource repository is the second key component of a Little-JIL process definition needed to support the rescheduling approach described here. The resource repository contains a set of resource instances that are available for assignment to the tasks specified in the Little-JIL activity diagram.

Thus, $ResourceRepository = \{Res_1, Res_2, ..., Res_l\}$, where each element of this set has certain capabilities and availabilities. A resource is defined as follows:

Definition 2.

$$Res = (ID, ResName, Attributes, SchedulableTimeTable, Capability_1, SkillLevel_1, Productivity_1, Capability_2, SkillLevel_2, Productivity_2, ...)$$

where,

- *ID* is a prose identification of the resource.
- *ResName* is the type of the resource, which indicates the kinds of capabilities that this resource has.
- *Attributes* is a set of (name, value) pairs that describe the resource. Some example attribute names might be Age, Experience_Level, Pay_Rate, and Model_Number (for a resource that is a machine).
- *SchedulableTimeTable* is a representation of the times when a resource is available to be assigned to an activity. This consists of a set of schedulable time intervals, defined by a start time (*st*) and end time (*et*). The resource can be assigned to an activity during any of the time intervals. Thus,

$$SchedulableTimeTable = \{[st_1, et_1], [st_2, et_2], ..., [st_s, et_s]\}$$

- $Capability_i$ (i = 1, 2 …) is the *i*th kind of capability that the resource has to offer. Two examples of capabilities of a resource that is a doctor or a nurse are 1) the capability to triage patients and 2) the capability to assess patients.
- $SkillLevel_i$ (i = 1, 2 …) is the level of quality at which the resource is able to perform $Capability_i$.
- $Productivity_i$ (i = 1, 2 …) is the average productivity that the resource is able to achieve in performing $Capability_i$.

In the above, $SkillLevel_i$ and $Productivity_i$ are attributes of $Capability_i$, and are used to determine whether a given resource has both the skill to perform a certain activity and the quantity of available capacity needed to complete the activity. Thus, specifically, assume that an activity specifies that *S* is the quantity of $Capability_i$ required in order to complete the activity. Then $S/Productivity_i$, is the time resource *R* needs to do the activity, where $Productivity_i$ is *R*'s productivity in doing $Capability_i$. Only if this amount of time is contained within R's *SchedulableTimeTable* attribute, can *R* be assigned to that activity.

4.3 Scheduling Activity Set Constructor

The Scheduling Activity Set Constructor assembles all of the information needed in order to make scheduling decisions. This function determines which upcoming activities fall within the scheduling window (whose size has been previously determined), and assembles the activities into a graph called the Dynamic Flow Graph (DFG). The DFG is derived from an analysis of another graph called the resource utilization flow graph (RUFG). Space limitations prevent a full discussion of how these graphs are built and used, and so we summarize descriptions of them.

The RUFG is a directed graph, derived from a Little-JIL activity diagram, which represents all the possible execution sequences of the process. Each node in the RUFG represents a step in the Little-JIL process definition and is annotated with precise and detailed information about the resource needs of the step. Thus, the RUFG is a representation of the static structure of the process.

Our rescheduling approach, however, uses dynamic process state information to take dynamic constraints and uncertainties into account. Thus when a rescheduling is needed we use the static RUFG and dynamic state information to generate a dynamic flow graph (DFG) that is then used as the basis for the rescheduling. The DFG incorporates both the RUFG's static information and runtime state information such as which activities are currently executing in parallel, which resources are allocated to them, and the priority level assigned to each activity.

The size and shape of the DFG is determined by a specification of the time window, which dictates how many of the future execution possibilities are to be considered in the rescheduling. At present we define the size of a time window by an integer that represents the length of the longest sequence of possible future activities that make resource requests. Thus, specifically, if *L* is the integer used to define scheduling window W and CURRACT is the set of activities that are currently being performed,

$$CURRACT = \{activity_1, activity_2, ..., activity_n\},$$

then node *NODE* is included in W if and only if *NODE* requires the allocation of resources, and, for some *i*, $1 \le i \le n$, there is a path, *P*, in the RUFG

$$P = (activity_i, n_1, n_2, ..., n_k, NODE)$$

such that the number of nodes n_j, $1 \le j \le k$ that make resource requests is less than L-1.

Each node in DFG contains two runtime attributes. One is the collection of resources that are candidates for assignment to the activity represented by the node. This set is drawn from the collection of available resources in the resource repository. The other attribute enumerates the resources that have actually been allocated at the conclusion of the scheduling process.

Further details about the definition of the RUFG and DFG are omitted due to space constraints

4.4 Using a GA To Do Resource Scheduling

The first step in using the GA approach is to represent the scheduling problem as an initial population of chromosomes. Through population evolution over a number of subsequent generations, increasingly optimal scheduling results can be obtained. This GA process is specified more precisely as follows.

(1) Generate initial population that contains a certain number of chromosomes. Each chromosome is encoded to represent a possible solution to the scheduling problem.

(2) For each generation, decode each chromosome in the population as a scheduling problem solution, applying constraints to eliminate some, and evaluating the quality of those remaining using some predefined measure of solution quality to determine the fitness of the chromosome.

(3) Select chromosomes with the highest fitness value(s) as the seed(s) for the next generation.

(4) Make crossovers and mutations to the selected chromosomes thus generating a new generation.

(5) Return to (2) and continue until satisfying some stopping criterion (e.g. completing some number of generations). The chromosome with the highest fitness in the final generation is selected and decoded to yield the scheduling result.

4.4.1 Encoding and decoding

We used the binary representation of integers to help encode the rescheduling problem as a chromosome. Note that because the set of DFG nodes waiting to be scheduled changes during process execution, new chromosomes must be built for each rescheduling.

For each DFG node, *N*, in a scheduling window (i.e. waiting for the assignment of resources) at time t, we establish a set of resource genes and a set of priority genes. Suppose *N* requires m resources. Then each of the *m* resource requests that have B_m candidate resources is encoded as a set of binary genes (i.e. genes whose value can be only 0 or 1), where the size of the set is the smallest integer greater than or equal to $\log_2 B_m$. The binary values of these $\log_2 B_m$ genes are used to represent the decimal number of a candidate resource, namely one that could be selected to satisfy the corresponding request. The initial population contains chromosomes that represent different combinations of such candidate resources, where the combinations are selected using constraint specifications described below. Similarly, the priority level of the node is represented by a set of *g* binary priority genes. When two or more DFG nodes contend for the same resource, the node with highest priority is assigned the resource. The structure of such a chromosome is shown in Figure 3.

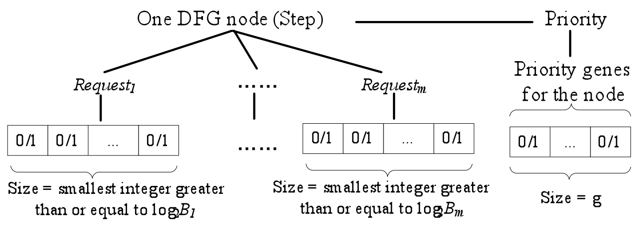
One DFG node (Step)
Priority
Request1
......
Requestm
Priority genes for the node
0/1
0/1
...
0/1
......
Size = smallest integer greater than or equal to log2B1
Size = smallest integer greater than or equal to log2Bm
Size = g

Figure 3. Chromosome structure

Each chromosome encoded by the above method can, subject to the application of constraints, be decoded as a scheduling scheme, namely the assignment of a specific resource to each of the requests made by each of the activities in the time window. Decoding is done essentially by reversing the encoding process.

4.4.2 Scheduling constraints

Full details about exactly how encoding and decoding are done are omitted due to space limitations. But the role of constraints is particularly important. Thus we now indicate how three types of constraints are used to enhance the efficiency and quality of our GA-based scheduling approach.

- **Capability constraint**: Only resources with needed capability and skill levels can be scheduled to satisfy a resource request. During the encoding process, none but such resources are determined as candidate resources for a request. This involves searching the resource repository to identify resources that have the capability to satisfy the request, using the *Capability* and *SkillLevel* attributes described in section 4.2.
- **Availability constraint**: A resource can be assigned to a step for a certain time period only if the resource is available at this time period, and has the capacity to provide enough effort to complete the step. This constraint is enforced during the decoding process by first determining the time period required using the *Capability* attribute of the step and the *Productivity* attribute of the candidate resources, and then examining the *ScheduledTimeTable* of each assigned resource to be sure the resource is available for that time period.
- **Step execution order constraint**: Steps can be executed only after all of their preceding steps have completed. Thus resources must be assigned to steps in a time window in an order dictated by the execution sequencing defined by the DFG. This constraint is applied during the decoding process. In particular, the start of the execution of a step must begin at a time after the time of completion of all of its predecessor steps. If a resource allocated to a step is no longer available because it has been allocated to another step (e.g. one executing in parallel), the schedule defined by this chromosome is rejected and this chromosome is not carried over to the next generation.

4.4.3 Fitness function

The role of the fitness function is to evaluate the relative desirability of each of the chromosomes as a solution to the resource rescheduling problem. The chromosomes with higher fitness are selected to be used in the next generation of the GA. This is how the survival-of-the-fittest mechanism is used to move the GA towards optimal solutions. The fitness function reflects an optimization goal for the resource allocation. Thus, for example, one possible goal of resource allocation in a hospital Emergency Department is to minimize total patient waiting time. In this case, the fitness function must quantify the waiting time that is to be expected for each of the resource assignments specified by a chromosome. One example of how this might be done is as follows. Suppose the set of steps in the time window is:

$$SchedulingStepSet = \{Step_1, Step_2, ..., Step_N\}$$

A scheduling scheme set *SSS* for *SchedulingStepSet* is the set of all the scheduling schemes corresponding to a set of chromosomes that represent possible resource allocations for *SchedulingStepSet*. Now suppose that the finishing time for the latest-finishing of all of the steps that immediately precede a step is time P_i. Then, P_i is defined as the "*Can be started time*" of $Step_i$. Assume that analysis of the availability of resources assigned by the scheduling scheme to $Step_i$ determines that $Step_i$ cannot be started until time S_i. Then the waiting time for $Step_i$ is defined as $(S_i - P_i)$. If scheduling scheme SS_k is the one that has the minimum total waiting time, then SS_k satisfies the following equation:

$$\neg((\exists SS_i \in SSS) \wedge (\sum_{a \in SS_i}(S_a - P_a) < \sum_{b \in SS_k}(S_b - P_b)))$$

Note that this fitness function does not attempt to minimize the total waiting time for all steps, only the total waiting time for the steps that are to immediately follow the currently executing steps. Thus this example is only one of many possible fitness functions, some of which will be harder to compute than others, and some of which will minimize overall waiting time more effectively. Experimentation (perhaps domain specific), will be needed to determine which fitness functions are most cost-effective.

4.4.4 Running GA

Before running GA, the following parameters must be set:

- **Population scale (*PS*)** is the number of chromosomes in each generation. When PS is larger the computation of each generation will take longer.
- **Crossover rate (*CR*)** is the number and possibility of crossover among chromosomes in a population. If CR is large, chromosomes with higher fitness might be destroyed. If CR is small, evolution and optimization rates may be slower.
- **Mutation rate (*MR*)** is the probability that a chromosome will be subject to mutation. If the mutation ratio is high unstable evolution may result. If it is low, there is less chance of avoiding local optima and finding a global optimum.
- **Generation number (*GN*)** is the number of generations (iterations) that the GA is to compute. Fewer generations will take less time, but may not come close to an optimum.

Research is needed to establish reliable guidelines for specifying how these parameters should be set. In simulation experiments to be described shortly, we will present the results of using some specific choices of parameters.

4.5 Rescheduling Indicator

The rescheduling indicator collects such runtime state information as the activities currently being executed, the resources being used to support those activities, resource capacity that is available, new arrivals, changes in priorities, and constraint changes. The following are examples of criteria that could be used in determining whether a rescheduling should be performed:

- If an activity that needs to be executed has not been allocated resources, a rescheduling should be carried out.
- If resources have been scheduled to an activity, yet the resources are not available when the activity should begin, a rescheduling should be carried out.
- If key attributes of some resources (e.g. cost or availability) have changed, a rescheduling should be carried out.

Research should determine the rescheduling criteria to be used for any resource allocation problem. Some criteria (e.g. the need to perform an activity for which no resource has previously been identified) seem universally applicable. Other criteria may be domain or application specific. And, indeed, different criteria may trigger reschedulings based upon time windows of different sizes, and rescheduling decisions may be made differently under different execution circumstances. Finally, note that in the work described in this paper rescheduling is done only at fixed points in time, with the more dynamic rescheduling triggers suggested in this section being left to be experimented with in future work.

5. Evaluation

To support analysis of the effectiveness of our approach, we used it to allocate resources during the running of simulations of processes that define how hospital emergency departments (EDs) perform their activities and utilize their resources.

A hospital ED requires the use of many different kinds of resources--human, mechanical, and automated--to support the treatment of patients. Since the costs of most of these resources (e.g. doctors, MRIs) are high, only limited numbers of these resources are available. But since many patients are typically being treated in an ED at the same time, contention for these resources can be expected. This contention usually leads to patient waiting time that can be excessive. Waiting time can be reduced by providing more resources, but there is a reticence to incur the sizeable expenses of these resources unless it can be shown that this will lead to dramatic reductions in waiting times. All of this indicates that there is much potential value in identifying effective hospital ED resource allocation strategies. Thus, the application of our time-window based rescheduling approach to the ED domain can be viewed both as a vehicle for evaluation of our approach and a contribution to more ED efficiency.

5.1 The Simulation Setting

The process that was used as the principal basis for the case studies presented here is the Little-JIL process shown in Figure 2. This process is a very high level abstraction of a process that specifies how a typical ED goes about treating patients. Note that, "*PatientInsideED*" needs a bed resource while the other steps do not need physical resources. But most steps need human resources. Note that non-leaf steps are used essentially to create scopes, and "real work" is done only by leaf steps. Thus, the size (namely an estimate of the relative length of time an activity requires in order to execute) and resource requests are shown in Table 1 only for each leaf step in this process.

Table 1. Size and resource requests of all the leaf steps

Step	Size	Request		
		ResName	Capability	SkillLevel
TriagePatient	11	TriageNurse	Triage	3
RegisterPatient	11	Clerk	Register	2
RNAssessment	11	Nurse	Assessment	2
MDInitialAssessment	11	Doctor	Assessment	3
PerformTests	31	AutoAgent	Test	2
RNProcedure	16	Nurse	Assessment	2
MDProcedure	16	Doctor	Assessment	3
MDFinalAssessment AndDecision	6	Doctor	Assessment	4
RNPaperwork	6	Nurse	Paperwork	3

The different kinds of resources available in an ED, such as doctor, nurse, clerk, and auto agent (a capability provided by the process execution or simulation system itself) are described in Table 2. Note that we assume that bed resources are sufficient, and thus do not explore their allocation in this study. For convenience we use time units rather than actual wall clock times and set the productivity of all resources to 1.

Table 2. Available resource descriptions

ID	Name	Human Name	Schedulable Time Table	(Capability, Skill Level, Productivity)
1	TriageNurse	TriageNurse1	[0, 10000]	(Triage, 4, 1)
2	Doctor	Doctor1	[0, 10000]	(Assessment, 5, 1)
3	Nurse	Nurse1	[0, 10000]	(Assessment, 4, 1), (Paperwork, 5, 1)
4	Nurse	Nurse2	[0, 10000]	(Assessment, 5, 1), (Paperwork, 3, 1)
5	Clerk	Clerk1	[0, 10000]	(Register, 3, 1)
6	AutoAgent	AutoAgent1	[0, 10000]	(Test, 4, 1)
7	AutoAgent	AutoAgent2	[0, 10000]	(Test, 4, 1)
8	AutoAgent	AutoAgent3	[0, 10000]	(Test, 4, 1)

5.2 Simulation Experiment

The complete set of inputs required in order to run a simulation of the ED process comprises 1) a process description, 2) a resource repository, 3) a specification of patient arrival rates and distributions of types, and 4) parameters needed to specify the execution of the GA. For our evaluative work we varied each of these inputs in order to support analysis of how sensitive the results obtained are to these variations. The settings and parameters we used initially are listed in Table 3.

Table 3. Initial simulation settings and parameters

Settings and Parameters	Value
GA population scale	32
GA crossover rate	1
GA mutation rate	0.1
Patient number	50
First patient arrival time	2

5.2.1 Case Study 1: The effect of process detail on scheduling effectiveness.

One hypothesis of this paper is that more complete and precise system specifications can support the computation of better scheduling schemas. To evaluate this hypothesis, we compared the results obtained from running simulations of the process defined in Figure 2, but using resource scheduling results obtained based on analysis of a less precise process definition. To do this we supposed that the assessment work done by the nurse and doctor is done in some unspecified way, rather than sequentially, as in Figure 2. A step named "*Assessment*" describes this activity. It includes requests for two resources, a doctor and a nurse. The *AssessAndTest* sub-tree is then as shown in Figure 4.

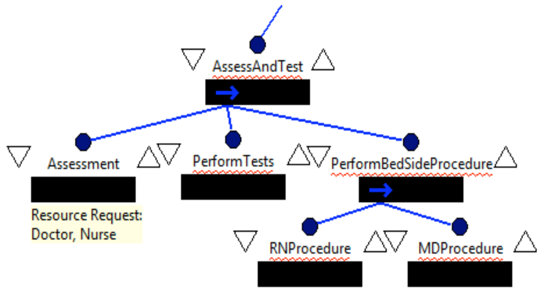

Figure 4. ED process with less precise details

We set the scheduling time window to 2 and used a patient arrival interval of 20. We estimated the execution time of the *Assessment* step to range from 22 the time that would be taken if assessment is done sequentially, down to 11, for the extreme case where assessment is done completely concurrently by the doctor and nurse. Other lengths of time between 11 and 22 are possible for cases where the overlap of the efforts of the doctor and nurse is not complete. The total simulated patient waiting time obtained for all these lengths of time is shown in Figure 5. Note that the additional detail about how *Assessment* is performed leads to substantial improvement when the step is defined to be the sequential performance of two substeps, and there is increasingly good improvement as the concurrency of the actions of the doctor and nurse is decreasingly complete. For completeness we also show the results of using the process shown in Figure 4 both as the basis of scheduling and as the basis for the simulation used to compute waiting time. The results of using this less complete and detailed process in this way are still less satisfactory, giving still more support to our hypothesis that greater process detail seems to provide important improvements in scheduling quality.

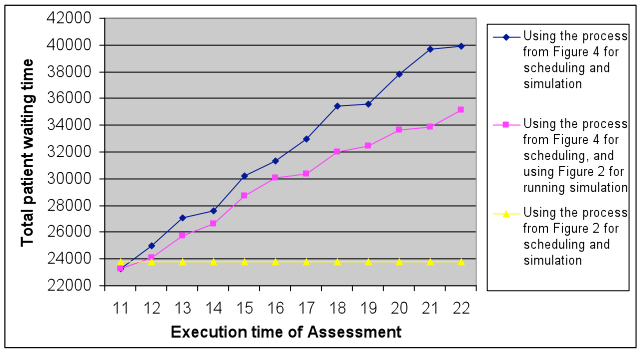

Figure 5. Total waiting time of less precise process under different execution time of assessment

Improvement is most dramatic in the case where the elaboration of the step is as sequential execution, suggesting the particular value of this type of elaborative detail. Interestingly, domain experts say that assessment is indeed usually performed sequentially by a doctor and a nurse. Thus, the greater detail in the definition shown in Figure 2 seems to support the possibility of scheduling that could reduce waiting time in a real-world ED.

5.2.2 Case Study 2: The effect of resource specification detail on scheduling effectiveness.

Another hypothesis of our approach is that complete and precise resource availability and capability specifications are the basis of better scheduling schema. To evaluate this hypothesis, we executed our rescheduling approach using resource specifications that did not include the *SchedulableTimeTable* attribute described in Section 4, and compared the results to those obtained when this information attribute was specified. We applied a first come first serve discipline for resource assignment, and compared results for patient arrival intervals ranging from 25 to 34. The results are shown in Figure 6.

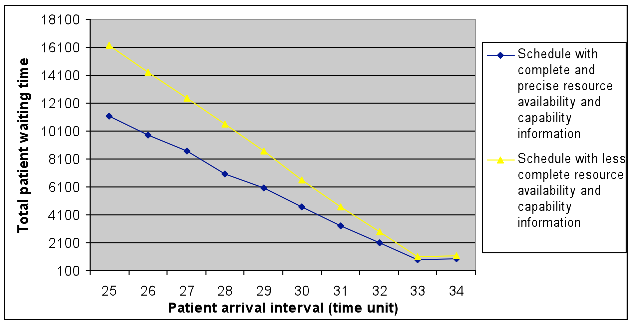

Figure 6. Total waiting time using precise and less precise resource descriptions

These results suggest that when the patient arrival rate is higher resource contention increases and more precise resource descriptions provide better support for scheduling. With decreasing patient arrival rates, resource contention decreases, and less precise resource descriptions support resource schedules that are increasingly close to those obtained with more precise resource descriptions.

5.2.3 Case Study 3: Scheduling cost variation with changing window size

Another goal of this research was to gain insight into the question of what window size would prove to represent a good compromise between lower costs of scheduling over smaller windows vs. better schedules resulting from consideration of more uncertainty by using larger windows. Figure 7 shows the influence of different window sizes on the number of reschedulings, the total time of a simulation run, and the quality of scheduling results obtained. The patient arrival interval in this experiment is set at 20 time units.

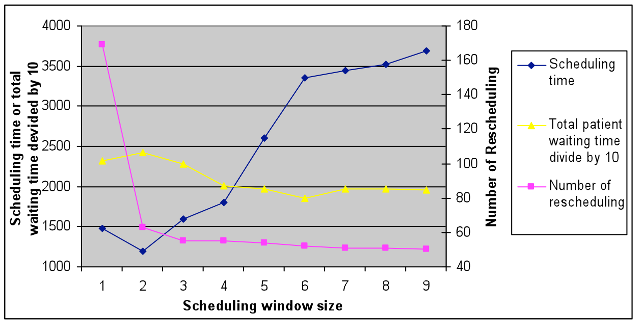

Figure 7. Scheduling time and number of rescheduling under different window size

Note that when the size of the scheduling window increases from 1 to 2, the number of reschedulings decreases sharply and the total time for all schedulings also decreases. As the window size keeps increasing, the number of reschedulings decreases far more slowly, but total time spent scheduling increases markedly, presumably because the number of steps in each rescheduling is large, making the cost of each rescheduling large as well.

Interestingly, note that when the window size reaches the number of patients being processed concurrently some reschedulings will be triggered while significant amounts of scheduling information from the previous rescheduling has not yet been used. Rescheduling thus causes some previous data to be superseded, thereby wasting effort. Moreover, the diagram shows that scheduling quality (as measured by total patient waiting time) does not necessarily improve as window sizes increases. Thus this case study suggests that window size selection should be carefully considered, and in fact might well best be determined dynamically, based upon the state of process execution.

5.2.4 Case Study 4: GA cost and accuracy

Because GA is essentially a heuristic, it is not possible to be sure that the results obtained are optimal, or even near-optimal. To help us gain confidence in the quality of the results obtained using GA, we compared them to results obtained using an exhaustive search (ES) of the space of all scheduling possibilities. As the computational complexity of ES is exponential, ES is possible only for relatively small scheduling problems. But we used these small scheduling problems to form a basis for comparison with results obtained using GA.

We ran a number of simulations with the number of patients set to 8, patient arrival interval set as 40 time units, setting the GA generation number to be 100. We noted that GA consistently obtained the exact same scheduling results as ES, indicating that GA found the global optimum for all of these small problems. Indeed GA invariably found the global optimum within the first 10 generations. On the other hand, GA offers substantial speed advantages, as expected. Figure 8 shows the time required to do a series of scheduling problems. In this figure, the X-axis represents the number of nodes in a rescheduling window. The primary Y-axis represents the amount of time consumed in the corresponding scheduling (in seconds) by ES and the secondary Y-axis represents the amount of time consumed in the corresponding scheduling (in seconds) by GA. The value of each point is gotten from the average of several runs.

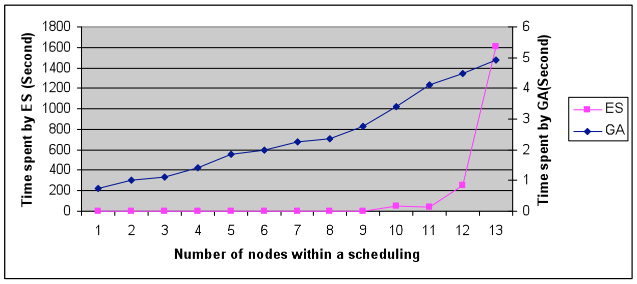

Figure 8. Scheduling time comparison of GA and ES

6. Analysis and Discussion

The time-window incremental rescheduling approach that we have proposed seems to promise the following advantages:

- The approach seems to be able to use sufficiently complete and precise specifications of processes and resources to deliver effective scheduling results. The case studies in section 5.2.1 and 5.2.2 show that complete and precise specifications can improve scheduling results, although these case studies also suggest that some details seem to be of more potential value than others. More research is needed to understand better which details are worth specifying.
- The window size used matters. The case studies in sections 5.2.3 and 5.2.4 suggest that if the window size is appropriately set, the benefits of lower scheduling cost and higher scheduling quality can be both obtained. This research is still quite preliminary, but it suggests that this window size may be context dependent and that more research is needed to understand better what features and state information should be used (and how) to suggest optimal window size.
- Continuous scheduling decision support can be provided in a process environment where frequent changes lead to continuous uncertainty. Our case studies suggest that relatively small time windows are likely to be most effective, perhaps because they enable relatively rapid reaction to changes (e.g. the sudden arrival of a new patient) and their attendant uncertainties.
- The GA scheduling heuristic seems effective. Our case study showed that GA can produce optimal results quickly for small scheduling problems. While this makes no assurance of GA efficacy for larger problems, the initial results are encouraging.

7. Summary and Future Work

This paper has presented a time window based incremental resource scheduling method that uses a genetic algorithm. We used this method to develop a scheduling tool that was integrated with an existing discrete event simulation system in order to study the effectiveness of the approach in creating good resource allocation schedules in affordable time. We used this system to support a variety of simulations of hospital emergency department processes. These initial case studies suggest that this approach can be effective. In order to gain more confidence in the approach further research is needed, however.

Future work:

Determining optimal window size: As noted in the previous section research is needed in order to determine how to determine what window size should be used in rescheduling.

Which details matter: Also suggested in the previous section is careful study of which process details, and which resource details are actually valuable in increasing the effectiveness of this scheduling approach. We have seen evidence, for example, that more details about process step sequentiality can lead to better schedules, but that elaborating the details of concurrently running steps may be less valuable. We need to determine which details are worthwhile, and which seem to be less useful.

Dynamic triggering of rescheduling: In this work rescheduling was triggered at fixed, predetermined intervals. But our architecture is designed to support dynamic determination of when to reschedule based upon various runtime parameters. Future work should explore when to carry out such dynamic rescheduling, and how to use runtime parameters to define the rescheduling problem parameters (e.g. the rescheduling window).

Analysis of different processes and parameters: this paper mainly focuses on changing parameters of window size and patient arrival interval in the specific hospital ED process domain. Different processes from different domains should be studied as well. In addition, GA parameters, such as crossover rate, mutation rate, and generation number should also be the subjects of further study to determine which combinations of these parameters are likely to be most cost effective. Furthermore, it seems important to study how to provide a mechanism for dynamically adjusting these various parameters depending upon the state of the system being supported.

Combine different value objectives in one scheduling: In this work schedules suggested by different chromosomes were evaluated using a single fixed objective function. But objectives may change during the running of a system (especially a long-running system). Thus it seems interesting and important to evaluate the efficacy of our approach when there is more than one objective function, and different objective functions are weighted differently at different times during the execution of a system.

8. Acknowledgments

We would like to thank Bin Chen and Heather Conboy for their help with the transformation from Little-JIL to RUFG, and Prof. Lori A. Clarke, Dr. M. S. Raunak, and Sandy Wise for their valuable feedback about this work. This paper is supported by the National Natural Science Foundation of China under grant Nos. 90718042, the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2007AA010303, 2007AA01Z186, as well as the National Basic Research Program (973 program) under grant No. 2007CB310802. This work was also supported by the National Science Foundation under Awards No. CCR-0205575, CCR-0427071, IIS-0705772 and CCF-0820198. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

9. References

[1] Al-Fawzan, M. A. and Haouari, M. 2005. A bi-objective model for robust resource-constrained project scheduling International Journal of Production Economics, vol. 96, pp. 175-187

[2] Alba, E. and Chicano, J. F. 2007. Software Project Management with GAs. Journal of Information Sciences, vol. 177, pp. 2380-2401.

[3] Barreto, A., Barros, M. d. O., and M.L.Werner, C. 2008. Staffing a software project: A constraint satisfaction and optimization-based approach. Computer & Operations Research, vol. 35, pp. 3073-3089.

[4] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., and Grünbacher, P. 2005. Value-Based Software Engineering: Springer Berlin Heidelberg.

[5] Cass, A. G., Lerner, B. S., McCall, E. K., Osterweil, L. J., Stanley M. Sutton, J., and Wise, A. 2000. Little-JIL/Juliette: A Process Definition Language and Interpreter. In Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, pp. 754-757.

[6] Cowling, P. and Johansson, M. 2002. Using real time information for effective dynamic scheduling. European Journal of Operational Research, vol. 139, pp. 230–244.

[7] Fowler, J. W., Monch, L., and Rose, O. 2006. Scheduling and Simulation. In Handbook of Production Scheduling, J. W. Herrmann, Ed.: Springer US, pp. 109-133.

[8] Gen, M., Gao, J., and Lin, L. 2009. Multistage-Based Genetic Algorithm for Flexible Job-Shop Scheduling Problem. Intelligent and Evolutionary Systems, SCI 187, pp. 183-196.

[9] Ghezail, F., Pierreval, H., and Hajri-Gabouj, S. 2009. Analysis of robustness in proactive scheduling: A graphical approach. Computers & Industrial Engineering.

[10] Goncalves, J. F., Mendes, J. J. M., and Resende, M. G. C. 2008. A Genetic Algorithm for the Resource Constrained Multi-project Scheduling Problem. European Journal of Operational Research, vol. 189, pp. 1171-1190.

[11] Herrmann, J. W. 2006. Handbook of Production Scheduling in International Series in Operations Research & Management Science: Springer US.

[12] Herroelen, W. and Leus, R. 2005. Project Scheduling under Uncertainty: Survey and Research Potentials. European Journal of Operational Research, vol. 165, pp. 289-306.

[13] Holland, J. H. 1992. Adaptation in natural and artificial systems. MIT Press Cambridge.

[14] Iima, H. 2005. Proposition of Selection Operation in a Genetic Algorithm for a Job Shop Rescheduling Problem. In EMO 2005, LNCS 3410, pp. 721-735.

[15] Jayamohan, M. S. and Rajendran, C. 2004. Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research, vol. 157, pp. 307–321.

[16] Jeong, K.-Y. 2000. Conceptual frame for development of optimized simulation-based scheduling systems. Expert Systems with Applications, vol. 18, pp. 299–306.

[17] Kutanoglu, E. and Sabuncuoglu, l. 2001. Experimental Investigation of Iterative Simulation-Based Scheduling in a Dynamic and Stochastic Job Shop. Journal of Manufacturing Systems, vol. 20, pp. 264-279.

[18] Li, Z. and Ierapetritou, M. G. 2008. Robust Optimization for Process Scheduling Under Uncertainty. Industrial and Engineering Chemistry Research, vol. 47, pp. 4148-4157.

[19] Mascis, A. and Pacciarelli, D. 2002. Job-shop scheduling with blocking and no-wait constraints. European Journal of Operational Research, vol. 143, pp. 498–517.

[20] Moratori, P., Petrovic, S., and Vazquez, A. 2008. Match-Up Strategies for Job Shop Rescheduling. IEA/AIE 2008, LNAI 5027, pp. 119-128.

[21] Peteghem, V. V. and Vanhoucke, M. 2009. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research.

[22] Pfeiffer, A. s., Ka´da´r, B., and Monostori, L. s. 2007. Stability-oriented evaluation of rescheduling strategies, by using simulation. Computers in Industry, vol. 58, pp. 630–643.

[23] Pham, D.-N. and Klinkert, A. 2008. Surgical case scheduling as a generalized job shop scheduling problem. European Journal of Operational Research, vol. 185, pp. 1011–1025.

[24] Pinedo, M. 2005. Scheduling: Theory, Algorithms, and System - Second Edition: Pearson Education, Inc.

[25] Rangsaritratsamee, R., Jr., W. G. F., and Kurz, M. B. 2004. Dynamic rescheduling that simultaneously considers efficiency and stability. Computers & Industrial Engineering, vol. 46, pp. 1–15.

[26] Wang, J.-B. and Liu, L.-L. 2009. Two-machine flow shop scheduling with linear decreasing job deterioration. Computers & Industrial Engineering, vol. 56, pp. 1487–1493.

[27] Wang, J. 2004. A fuzzy robust scheduling approach for product development projects. European Journal of Operational Research, vol. 152, pp. 180–194.

[28] Wise, A. 2006. Little-JIL 1.5 Language Report Department of Computer Science, University of Massachusetts, Amherst UM-CS-2006-51.

[29] Wong, G. Y. C. and Chun, A. H. W. 2004. Constraint-based rostering using meta-level reasoning and probability-basedord ering. Engineering Applications of Artificial Intelligence, vol. 17, pp. 599–610.

[30] Xiao, J., Wang, Q., Li, M., Yang, Q., Xie, L., and Liu, D. 2009. Value-based Multiple Software Projects Scheduling with Genetic Algorithm. In International Conference on Software Process 2009 (ICSP2009), LNCS 5543, Vancouver, Canada, pp. 50-62.

[31] Yang, B. 2007. Single Machine Rescheduling with New Jobs Arrivals and Processing Time Compression. International Journal of Advanced Manufacturing Technology, vol. 34, pp. 378-384.