Incremental Scheduling to Maximize Quality in a Dynamic Environment

Anthony Gallagher, Terry L. Zimmerman and **Stephen F. Smith**
The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{anthonyg,wizim,sfs}@cs.cmu.edu

Abstract

We present techniques for incrementally managing schedules in domains where activities accrue quality as a function of the time and resources allocated to them and the goal is to maximize the overall quality of actions executed over time. The scheduling problem of interest is both over-subscribed and dynamic; there is generally more to do than is possible within imposed deadlines, and opportunities to execute new, potentially higher payoff activities continually arrive. Like other dynamic domains, schedule stability and computational cost concerns argue for the use of incremental techniques in this context. The novel emphasis on maintaining schedules that produce "high value" results when faced with a changing environment differentiates this problem focus from that of previous research. We develop and evaluate methods for incrementally maintaining schedules that maximize the quality (or utility) of executed activities. We contrast the performance of our incremental techniques to that of comparable schedule (re)generation techniques with respect to quality, stability and cost considerations. The results clearly favor incremental scheduling in this context, and suggest opportunities for broader schedule improvement search.

Introduction

An ability to manage schedules incrementally is crucial to effective performance in many dynamic environments. In some cases (e.g., when humans are involved in executing scheduled activities), this is due to the value that schedule stability brings to execution processes in the face of unexpected events that force changes. The performance advantage that might accrue via schedule re-optimization at each required change must be balanced against the potential disruption to execution processes caused by a discontinuous sequence of schedules. In other cases, basic scalability concerns and computational demands tied to keeping pace with execution may simply prohibit schedule recomputation. Accordingly, prior research has devoted considerable effort toward the design and development of incremental scheduling procedures (Smith 1994; Zweben *et al.* 1994; Becker & Smith 2000; Sakkout & Wallace 2000; Bartak, Muller, & Rudova. 2004).

Our focus in this paper is on incremental scheduling of so-called "knowledge-intensive dynamic systems" (KIDS), which are concerned with the production and manipulation of knowledge products. Central to a definition of this problem class is the objective of maximizing process quality: the quality (or utility) gained by executing any given activity is a function not only of its ascribed worth (or priority), but also of the time and resources that are allocated to it, and the goal is to maximize the return on investment of planned activities. For example, consider the production of a news story by a news agency. The quality contribution of a given story to an overall news broadcast will depend on the significance of the event being covered, the amount of time spent on site, and on the expertise of the reporter assigned to the story. At any point in time, there are a number of disparate events that might be covered, and decision making will focus on maximizing the overall quality of the broadcast. Furthermore, additional news events to be covered arise over time and introduce potentially higher payoff activities whose execution must be weighed against that of currently scheduled activities. While already allocated resources may be redirected to respond to new higher priority events, there is typically some trade-off cost (e.g. travel change penalties).

Such quality-centric scheduling problems differ significantly with respect to key assumptions that are traditionally made when considering over-subscribed scheduling problems. With very few exceptions, prior research on over-subscribed problems has assumed that activity durations are fixed, and the objective, subject to problem constraints on resource availability and activity timing, is either to maximize the number of activities that get into the schedule (possibly weighted by priority), or minimize the extent to which deadlines must be relaxed (e.g., minimizing weighted tardiness). In contrast, activity durations in our quality-centric problem are not fixed; instead they are decision variables, subject to specified minimum and maximum bounds. In this case, the objective is to produce a schedule that maximizes the cumulative quality of all scheduled activities, where the quality of a given activity is determined by its duration, its priority and the level of expertise of the resource assigned. Many diverse application domains present this sort of quality-centric scheduling problem. In the area of quality testing of large software systems, the confidence that a given component is robust varies with time spent in testing it, and both test results and delivery deadlines will force choices as to level of testing of different components. In planetary exploration, the time spent by robotic rovers on various sampling and test activities will differentially affect overall scientific return of the mission, and ongoing analysis of results will continually reprioritize and augment the set of existing tasks.

In this paper, we investigate the implications of this novel class of dynamic scheduling problem for the design of incremental scheduling procedures. We develop techniques for minimally-disruptive and priority-based schedule revision analogous to those developed in previous work, but with an alternative bias toward maintaining ”high value” schedules. Through experimental analysis in a representative KIDS domain, we show these techniques to exhibit scalable performance properties in relation to a baseline reschedule from scratch approach. We also demonstrate the potential utility of these techniques for more broad-based schedule improvement.

The remainder of the paper is organized as follows. First, we formulate more precisely the class of dynamic scheduling problem of interest. Next, we present a core set of techniques for incrementally extending and maintaining high quality schedules. We define procedures for inserting activities into an existing quality-based schedule, for substituting new higher quality activities for other currently scheduled activities, and for composing these basic operations into overall scheduling procedures. We then define a representative set of test problems and empirically evaluate the performance of these incremental techniques. We present results indicating the tradeoffs in incremental versus regenerative response to new events with respect to quality, cost and solution stability. The paper ends with a summary of related work and a few concluding remarks.

A quality-centric scheduling problem

The class of dynamic, incremental scheduling problems that we focus on in this paper can be defined as follows:

- At any point in time t there is a set $A(t)$ of activities that could be scheduled. Each activity a has a priority $pr(a)$, an earliest start time $est(a)$ and a latest finish time $lft(a)$, which defines the window in which a must occur. The set $A(t)$ evolves over time as new activities arrive and known activities that cannot be scheduled within their time windows become lost opportunities.
- A set R of resources are available for assignment to activities. Each resource r has a designated skill level $sk(r)$, reflecting the efficiency with which activities can be performed. (Higher skilled resources achieve a given task quality more quickly than lower skilled resources.) Each resource r also requires some amount of time, $setup(r, a, b)$, to change from the state required by activity a to the state required by activity b. Where convenient, we will refer to this sequence-dependent delay simply as a setup activity.
- Each activity a requires exclusive use of a single resource r to execute and the quality obtained by executing a increases as its duration increases in an anytime fashion. Each activity a has a “minimum quality” constraint. If resource r is assigned to a, then this implies a minimum duration constraint $mindur(a, r)$.
- The quality $q(a, r)$ obtained by executing activity a on resource r from t_1 to t_2 is defined as
$$q(a, r) = pr(a) \times sk(r) \times dur(a) \tag{1}$$
where $dur(a) = t_2 - t_1$. Hence larger $pr(a)$ and $sk(r)$ values imply higher priority and skill levels. Here we assume a simple linear relation between activity duration and quality. Note however, that the techniques introduced in this paper are easily generalized to more complex quality profiles (e.g., piece-wise linear) by decomposing activities into sequences of linear-profiled sub-activities.
- The overall quality of a set S of scheduled activities is then defined as
$$Q_S = \sum_{s \in S} q(a^s, r(a^s)) \tag{2}$$
where $r(a^s)$ is the resource assigned to a^s.

Extending and Maintaining Quality-Centric Schedules

The design of an incremental scheduler for quality-centric problems presents some interesting representational challenges. The need to dynamically place new activities into an existing schedule over time suggests a *flexible-times* schedule representation. In flexible-times schedules, the start and end times of activities are not anchored; activities are instead allowed to “float” within the time bounds determined by current problem constraints, and resource conflicts are avoided by sequencing pairs of competing activities. On the other hand, a schedule that maximizes quality will by definition be a *fixed-times* schedule. To leave any flexibility in a scheduled activity’s duration is to settle for a lower quality solution than would be possible if its duration was simply extended. Yet, representation of this higher quality solution may hide opportunities for better utilizing resource capacity and complicates the process of accommodating change.

We reconcile this representational dilemma by exploiting dual models of the current schedule. To provide visibility of options for accommodating schedule change, we employ a Simple Temporal Problem (STP) constraint network representation (Dechter, Meiri, & Pearl 1991), hereafter referred to as the *temporal network*. A flexible-times representation of each resource’s assigned activities, which we refer to as its *STP timeline*, can then be maintained. Here, only minimum duration constraints on quality-accumulating activities are enforced, together with *est* and *lft* constraints on execution, sequencing constraints and necessary resource setup activities. As these chains of activities get “pushed” back and forth within their constraints, time intervals that might feasibly be allocated to new activities can be identified.

We also maintain a fixed-times representation of each resource’s schedule, referred to as the resource’s *quality timeline*. The quality timeline of a resource designates the set of start and end times for the currently assigned sequence of activities that yield the ’sequence optimal’ quality schedule for this resource (optimal with respect to a particular sequence of chosen activities). By using information from both representations we are able to *incrementally* maintain

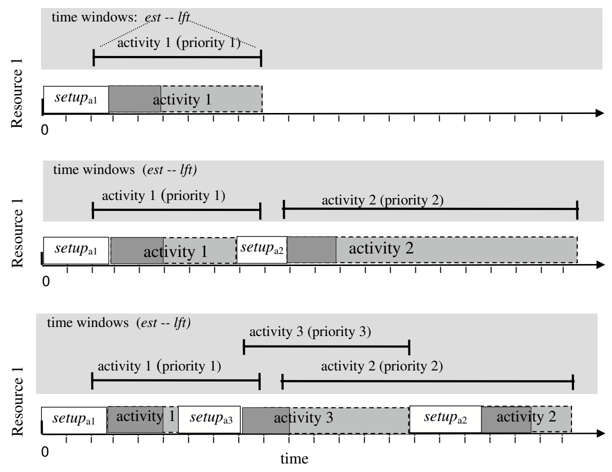

Figure 1: Consecutive allocation of 3 activities on a resource timeline. Darker portion of activity represents the minimum required duration while the dashed extension represents the duration giving highest quality.

this sequence-optimal fixed-times schedule as new activities are added and others are retracted. Figure 1 illustrates this dual schedule representation scheme as three activities postulated to arrive over time are allocated consecutively and optimally on a single resource. Each activity is assumed to have the same minimum quality requirement. Representation on the quality timeline is depicted by the dashed extent of activities while the temporal network tracks only the required minimum duration (shown in dark shading). For example, when activity 2 arrives its higher priority causes the scheduler to cut short resource effort on activity 1 (while respecting its minimum duration) such that it can change state in time to service activity 2 for its *entire* time window (maximizing overall plan quality).

We next summarize the incremental scheduling procedures for incorporating a new activity into the current schedule, and then adapt this operation to provide a quality-centric schedule generation capability.

Incremental Activity Insertion

Given these representational assumptions, let us first consider the basic operation of inserting a new activity into an existing schedule. Accommodation of a new activity consists of (1) determining a feasible insertion point on a suitable resource's timeline and then adjusting the durations of key activities so as to maximize overall plan quality. This operation is outlined in Figure 2. Step one is accomplished by interrogating the temporal network to identify the set of feasible insertion points for the new activity a^{new} on some resource. A feasible insertion point is a gap on a given resource r's STP timeline large enough to accommodate the minimum duration of a^{new} along with any sequence-dependent setup time constraints. For example, a feasible insertion option on resource r between two activities a^i and a^j must satisfy the condition $lst(j) - eft(i) \geq setup(r, a^i, a^{new}) + mindur(a^{new}, r) + setup(r, a^{new}, a^j)$ in addition to satisfying a^{new}'s est and lft constraints.

```
**Schedule-Activity**($Schedule, Activity$)
1. $Options \leftarrow$ Find-Options($Schedule, Activity$)
2. $Selected \leftarrow$ Choose-Option($Options$)
3. Insert-In-Temporal-Network ($Activity, Selected$)
4. Insert-In-Quality-Timeline ($Activity, Selected$)
5. Return($Selected$)
   **end**
```

Figure 2: High level algorithm for scheduling a new activity

Once feasible options are determined, $Choose - Option$ (Step 2) is applied to evaluate each and select one for implementation. This choice is of course central to quality and stability properties of the schedule over time, and we propose several candidate strategies below.

Having chosen an insertion option, Step 3 adds the appropriate sequencing, setup and (minimum) duration constraints associated with a^{new}'s placement in the chosen option to the underlying temporal network. An incremental STP network solver is used to propagate constraints and check consistency. [1]

Finally, Step 4 inserts the new activity into the corresponding quality timeline of the assigned resource. Upon insertion of a new activity a^{new} into a resource r's quality timeline, the durations of surrounding activities are recomputed to ensure that the duration assigned to a^{new} and r's overall schedule remains quality-optimal with respect to the sequence. This is accomplished by first determining the set of existing activities on the timeline whose durations may require adjustment as a result of the insertion.

Let $TL(r)$ be the timeline of resource r containing N indexed scheduled activities ($a_1..a_N$), and let the insertion point of a^{new} be i (i.e., located between activities a_i and a_{i+1} on $TL(r)$). Further assume that $st(a_j)$, $ft(a_j)$ and $dur(a_j)$ designate the current start time, finish time and duration respectively of any a_j on $TL(r)$. We can bound the set of scheduled activities possibly affected by the insertion of a^{new} in the timeline by identifying two activities on $TL(r)$: the nearest activity a_x preceding insertion point i whose start time cannot be pushed earlier (i.e., $st(a_x) = est(a_x)$) and the nearest activity a_y following insertion point i whose finish time cannot be pushed later in time. We define the bounding region $BR_{r,i}$ to be the set of activities on $TL(r)$ between a_x and a_y. More precisely,

$$BR_{r,i} = \{a_k \in TL(r) \mid x \leq k \leq y\} \qquad (3)$$

where

[1]The underlying network is robust enough to handle much more complex precedence relationships than this domain's single-resource precedence chains.

$$x = \arg\max_k \{a_k \in TL(r) \mid st(a_k) = est(a_k), 0 < k \leq i\}$$
$$y = \arg\min_k \{a_k \in TL(r) \mid ft(a_k) = lft(a_k), i+1 \leq k < N\}$$

Given the choice to insert a^{new} between a_i and a_{i+1} and having determined $BR_{r,i}$, the optimal duration of a^{new} is established by decreasing the time allocated to activities in $BR_{r,i}$ in reverse priority order (since the quality contribution of an activity is proportional to its priority). Let $SBR_{r,i}$ be a reverse-priority sorted queue of the activities in $BR_{r,i}$, and a_p be the head of $SBR_{r,i}$. Computation proceeds by repeatedly determining, for the current a_p, the amount δ by which a_p's duration should be shrunk and a^{new}'s duration expanded. If a^{new} has higher priority than a_p, then δ cannot exceed $slack(a_p) = dur(a_p) - mindur(a_p)$. However, δ will be less than $slack(a_p)$ if any intervening activity a_j between a_p and the insertion point i has a time bound that limits its ability to "slide" along $TL(r)$. Conversely, if a_p has higher priority than a^{new}, then δ is the maximum amount by which a^p can be shrunk in an effort to achieve a^{new}'s minimum duration.

More precisely, assume that a_p precedes insertion point i on $TL(r)$ and define $minslide(a_p, i)$ as follows:

$$minslide(a_p, i) = min\left[\{st(a_k) - est(a_k)\}_{k=p+1}^{i}\right]$$

Then the δ value by which a_p's duration is reduced is given by:

$$\delta = \begin{cases} min\big[\, slack(a_p), \\ \qquad minslide(a_p, i), st(a^{new}) - est(a^{new})\big] \\ if\ pr(a^{new}) > pr(a_p) \\ \\ min\big[\, slack(a_p), minslide(a_p, i), \\ \qquad st(a^{new}) - est(a^{new}), \\ \qquad mindur(a^{new}, r) - dur(a^{new}))\big] \\ if\ pr(a^{new}) \leq pr(a_p) \end{cases} \tag{4}$$

The δ value for the case where a_p follows insertion point i is defined similarly. The algorithm continues to shrink existing activities in this manner until $SBR = \emptyset$ or $pr(a_p) \geq pr(a^{new})$ and $dur(a^{new}) \geq mindur(a^{new}, r)$.

Considered overall, the activity insertion operation of Figure 2 provides a baseline approach to balancing quality and stability objectives as new activities are introduced into the schedule over time. The choice of which placement option to commit to is a heuristic one, but given the sequencing constraints established on a given resource, the durations for scheduled activities are optimized with respect to quality. Given that existing sequencing constraints are left intact, perturbation is also minimized (at the possible expense of achieving higher quality via sequence rearrangement). We examine the impact of this trade-off between quality and stability in the Experimental Analysis section.

Insertion Option Heuristics

As indicated previously, a feasible insertion option for activity a^{new} on a resource r's timeline includes an availability gap large enough to accommodate the sum of the following duration constraints: $mindur(a^{new}, r)$, $setup(r, a^{prev}, a^{new})$, if a^{prev} exists, and $setup(r, a^{new}, a^{next})$, if a^{next} exists. Given a set of feasible insertion options, there are a number of possible strategies for determining which option to take:

- *first-slot* - A naive, low-cost strategy is to simply insert a^{new} in the first such valid insertion option that is found (given some enumeration procedure). While this is likely to be the most efficient strategy, blindly inserting a^{new} in the schedule may actually *decrease* overall schedule quality.
- *first-gain-slot* - A more costly but more informed strategy is one which generates and assesses valid insertion options until it finds one that increases current schedule quality. This strategy will ensure at least some overall gain in quality before committing to a^{new}, and hence sometimes may reject a^{new} despite the fact that there are feasible options.
- *best-slot* - A third, more quality-greedy and also more costly strategy is one that generates all possible insertion options, assesses the quality gain for each (according to equations 1 and 2), and then chooses the highest quality gain option. If no option yields a gain in quality, a^{new} is rejected.
- *min-setup* - Finally, given that quality is not accumulated during periods in which the resource is occupied with performing setup activities, a strategy that attempts to minimize setup time may provide a lower-cost but still effective alternative to *best-slot*. Under the *min-setup* heuristic all possible options are generated, as above, but the option that minimizes setup time is selected.

A more disruptive insertion operation

Regardless of which option selection heuristic is in play, the basic activity insertion operation described above is fundamentally additive in nature. It can be constrained to add a new activity a^{new} to the schedule only if this addition results in a higher quality schedule. But it does not consider possibilities that involve retraction or rearrangement of existing activities. In this regard it is biased towards stability in the schedules that are generated over time. On the other hand, this stability can block the insertion of newly arrived, higher priority (and hence higher quality) activities.

To overcome this maximally conservative posture toward acceptance of new activities that arrive over time, we define a second, more disruptive activity insertion operation that allows currently scheduled activities to be supplanted by a new activity a^{new} if there is a net quality gain. This *bumping* insertion operator attempts to strike a better tradeoff between solution quality and stability objectives.

Given a new activity a^{new}, candidates for bumping are those activities that contend for a resource during a^{new}'s time window. The set of conflicting activities C is found by locating the set of activities scheduled during the time window of a^{new}:

$$C = \{a_k \in S \mid \quad est(a^{new}) \leq st(a_k) \leq lft(a^{new}) \vee est(a^{new}) \leq ft(a_k) \leq lft(a^{new})\} \quad (5)$$

More precisely, the bumping operator proceeds as follows. Potential bumping candidates on a given resource, r, are identified as the intersection of C (the set of all conflicting activities) with TL(r) the set of activities on the timeline of r. The complexity of this operation is quadratic in the size of the intersection set because only contiguous sequences of activities in C are considered as candidates to bump in favor of a^{new}. The operator utilizes the same *best slot* option selection heuristic as simple insertion, but in this case the quality that is gained from adding a^{new} at a particular position on $TL(r)$ is offset by the loss in quality incurred from the removal of any bumped activities from $TL(r)$. The set of options generated is a *superset* of the set considered by simple insertion; existing gaps on $TL(r)$ (for all r) are evaluated as before, as well as those gaps that would be created through the removal of subsequences of one or more activities currently on $TL(r)$. The combined set of options is assessed and ranked according to net quality gain, and the highest gain option is selected. If no option generated during this process offers a net gain, a^{new} is rejected.

Using Incremental Insertion to Build Schedules

At the other extreme along the quality/stability tradeoff spectrum we consider the strategy of responding to the arrival of a new activity by simply building a new schedule from scratch. This strategy is the most disruptive, as no explicit attention is given to keeping aspects of the current schedule intact. It can also can be seen to provide the best opportunity for maximizing quality, as all previous decisions are reconsidered in light of the newly received activity. Of course a downside to this strategy is relative computational cost; given the arrival rate of new activities, gains in schedule quality may be counterbalanced by difficulties in keeping pace with execution.

The activity insertion operations discussed above can be configured to provide a basic capability for schedule generation from scratch. Given the set $A(t)$ of known activities at time t, an initial schedule can be constructed by repeatedly (1) selecting an unscheduled activity a from $A(t)$ and (2) invoking the activity insertion operation to add a into the schedule. Since, by definition, the overall problem is oversubscribed, it is likely that not all activities in $A(t)$ will be successfully added to the schedule. Hence, the order in which activities are added can significantly impact the overall quality obtained.

Intuitively it makes sense to select activities from $A(t)$ in order of their expected quality contribution to the final schedule. In this respect, an activity a's priority $pr(a)$ is a simple surrogate measure of its quality (recall equation 1), and can be used as one basis for prioritization. We hereafter refer to activity ordering on this basis as $priority$. At the point of selecting an activity a from $A(t)$ for scheduling, both of the other two factors that influence a's quality contribution, the resource to which a will be assigned and its eventual assigned duration, are unknown. We do, however, know a's duration-constraining time window, $lft(a) - est(a)$, and use of this upper bound information leads to a second prioritization metric, $pr(a) \times (lft(a) - est(a))$. Below we refer to this second activity ordering strategy as $priority * maxdur$.

The next section quantifies the relative performance of these proposed incremental scheduling techniques.

Experimental Analysis

We first evaluate the relative performance of the various activity and option selection heuristics we have proposed when coupled with the basic activity insertion operation. We consider this first question in the context of schedule generation using different activity selection criteria. Our initial goals are twofold: to identify the best minimally disruptive insertion operation for subsequent evaluation in a dynamic setting, and (2) to establish a baseline reschedule-from scratch strategy for comparison with incremental strategies.

Following this initial analysis, we evaluate the efficacy of various scheduling strategies in responding to the dynamic arrival of new activities. This leads to consideration of some extended optimization procedures, in an effort to better understand the observed results.

Finally, we analyze the stability of the solutions produced by the various incremental scheduling strategies tested.

A Representative KIDS Problem Suite

To evaluate performance characteristics of the incremental scheduling procedures defined above, we here describe a domain that is prototypical of the KIDS problem outlined in the second section. In this domain, which we refer to as *News Agency*, the agency's objective is to provide high quality coverage of worldwide news events using a limited pool of reporters to provide on-location news reports. The agency defines an event's *est* and *lft* as a "news event window" (NEW) -which can be viewed as the time span around an event during which a news report (the basic activity of interest) is considered news-worthy. Thus, report tasks have deadlines since quality is only accrued if they are produced during the NEW. Sequence-dependent setups may arise because each news event is postulated to occur in one of five cities and reporters must travel to each event location to generate an on-site news report. All problems have 3 available resources (reporters), each with a randomly generated skill level between 1 and 3 ($sk(r)$ of equation 1) and a random initial location. News events have uniformly distributed priorities between 1 and 5 (5 being the highest priority) with minimum required quality of 1 (corresponding to 1 hr minimum duration for a reporter with skill=1). NEW spans are uniformly distributed between 1 and 10 hours and are situated within a 10-day time horizon.

Two sets of News Agency problems were generated representing different degrees of oversubscribed conditions; one set consists of ten 100 activity problems and the other has ten 200-activity problems. For the 100-activity problem set roughly 80% of its activities are typically allocated by the

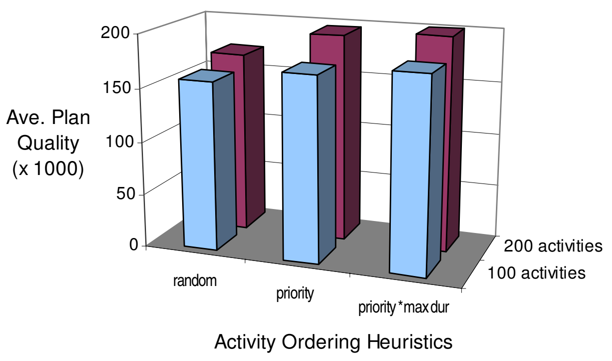

Figure 3: Activity sorting impact on plan quality

scheduler and the schedules average over 20% remaining free space on the timelines. For the 200-activity problem set slightly more than 50% of activities are typically installed in the schedule and the schedules contain, on average, less than 5% free space.

Activity Selection and Option Ordering Heuristics

Turning first to the ordering activities for scheduling, Figure 3 shows the average quality of schedules generated for each of the two problem sets defined above using the $priority$ and $priority * maxdur$ activity selection heuristics defined earlier. To further calibrate the results, we include results obtained by just randomly selecting activities for scheduling -the $random$ selection heuristic. In all cases, the (*best-slot*) heuristic (analyzed further below) was used to select among feasible insertion options and determine final placement of selected activities in the schedule.

As can be seen, the *priority*maxdur* ranking modestly outperforms the others in terms of average plan quality returned. This heuristic is most closely aligned with factors contributing to quality, reflecting both activity priority and time window size. The $priority$ and $random$ heuristics ranked second and third, respectively. Since differences in runtime overhead for different heuristics is negligible, we use the $priority * maxdur$ heuristic to rank activities for all experiments reported in the remainder of the paper.

Table 1 compares average quality and runtime for each insertion option selection heuristic on our two problem sets. The low-cost heuristics (*first-slot* and *min-setup*) exhibit roughly the same runtimes, but *min-setup* dominates *first-slot* in terms of quality. Both quality-informed heuristics (*first-gain-slot* and *best-slot*) clearly outperform the other heuristics in terms of overall quality at a 4-5 times increase in runtime. Thus, our hypothesis that minimizing setup activities (which are non-quality accumulating) might be an effective surrogate for maximizing quality is shown not to be the case. Given the reasonable computation times, we have chosen the more quality-informed heuristic of these two, *best-slot*, as the default option-ordering heuristic for the remainder of this paper.

Heuristic	100 Activities		200 Activities	
	Quality	cpu (s)	Quality	cpu (s)
first-slot	1.0	1.06	1.0	2.78
first-gain-slot	3.4x	4.05	3.2x	12.13
best-slot	3.9x	4.63	3.4x	12.51
min-setup	2.3x	0.78	1.9x	2.78

Table 1: Insertion option heuristics on 100 and 200 activity problems (Quality is shown relative to *first-slot* baseline)

Scheduling Approach	% of Optimal Quality	# Solutions at Optimal
min-setup	58.7	0
First-gain option	73.9	0
best-gain slot	89.7	2

Table 2: Comparison of option heuristics to the optimal quality solution on 10 random problems

To further calibrate the performance of our insertion option selection heuristics, we compared them with the optimal quality solution for a set of smaller problems. Specifically, we used an exhaustive enumerative procedure to determine the optimal quality solution for a randomly generated set of 10 problems having up to 10 activities each (the largest sized problems we are able to optimally solve in reasonable time). Table 2 gives the percent of optimal obtained with each heuristic, as well as the number of problems for which each approach found the optimal solution. *priority*maxdur / best-gain slot* scheduling is seen to perform well on these problems, coming within 10.3% of optimal on average and finding the optimal solution for 2 of the problems.

Responding to New Events

We now turn attention to the dynamic problem of responding to the arrival of a new activity. Specifically we compare the performance of three incremental dynamic scheduling strategies in adapting a base schedule to account for the arrival of a newly arrived activity. Base schedules are created using the best quality-centric scheduling strategy found above (i.e., $priority * maxdur$ activity selection, *best-gain slot insertion*), again using our 100 and 200 activity problem sets. Each problem in both sets is then augmented with 10 new activities (randomly generated under the parameters used for the baseline activities) which are then given one at a time to the scheduler after it completes the baseline schedule. The incremental strategies studied are: *insert-only*, a pure insertion method that seeks the insertion option with the highest quality gain, *insert-or-bump*, the insertion-plus-bumping method previously described, which selects the highest quality gain choice amongst available insertion options and bumping options, and *insert-or-bump+*, an extended insertion-plus-bumping strategy that additionally attempts to reinsert any bumped activities. This latter strategy looks for opportunities to reinsert any activities that are bumped in favor of a new activity *without* bumping addi-

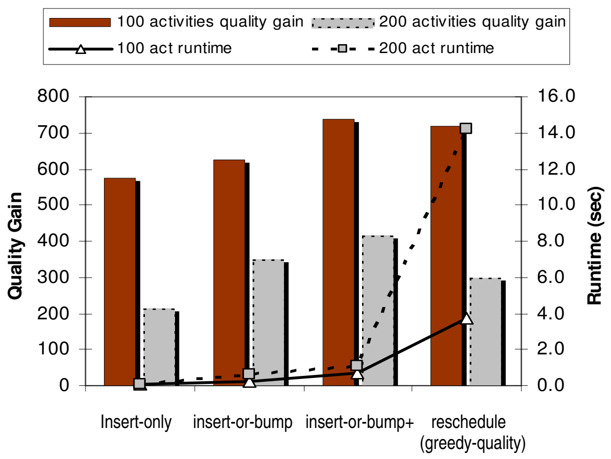

Figure 4: Dynamic incremental strategies vs. rescheduling. (Averages charted are for a single new activity insertion: Quality gain and runtime are averaged for each of the 10 new activities per problem, and these problem averages are then averaged across the 10-problem set.)

tional activities on the timeline. A *reschedule-from-scratch* strategy, which simply adds the new activity to its original set and reschedules using the greedy-quality heuristic, is included for comparison.

Figure 4 compares the dynamic incremental scheduling strategies with rescheduling from scratch in terms of the average quality gain (quality increase over baseline schedule given a new activity) and runtime. As might be expected, the *insert-only* strategy produces the lowest quality gains, its effectiveness degrading as the resource timelines become more heavily loaded (i.e. in the 200-activity set). Not surprisingly, rescheduling generates higher quality plans than the former and is computationally more expensive than the other strategies by factors of between 5 and 160 times. However, the fact that the bumping enhanced strategies, *insert-or-bump* and *insert-or-bump+*, perform so well relative to rescheduling is unexpected. Rescheduling is slightly outperformed by *insert-or-bump+* in the average quality of schedules produced over the 100-activity problem set, and *both* bumping strategies outperform rescheduling from scratch on the 200-activity set. In part, the efficacy of the bumping strategy search for opportunities to replace low quality-producing activities in the schedule constitutes a local repair capability that heuristic-guided schedule construction alone cannot match.

We confirmed the Figure 4 implication that the advantage afforded by bumping grows as the problem becomes more oversubscribed, by generating two additional 10-problem sets of 300 and 500 activities while fixing the number of reporters at three. These experiments showed that the quality-gain factor for scheduling under the *insert-or-bump* heuristic versus rescheduling grows in nearly linear fashion as problems become more oversubscribed. Predictably, performance of the *insert-only* heuristic degrades as the likelihood of open insertion options on the timelines diminishes.

Extended Schedule Optimization

Considering this observation in more detail, we can identify two ways in which the greedy *reschedule-from-scratch* strategy can be misled into making suboptimal placement decisions. First, the presence of sequence-dependent setup times can lead to situations where an activity rated highly by the *priority*maxdur* activity sorting heuristic gets placed at some position on a resource r's timeline without proper anticipation of the future opportunities to minimize setup time, resulting in a final schedule with substantial non-value adding segments of time on $TL(r)$. Second, the fact that activities have unique time windows (*NEWS*) implies that the placement of one activity on a given $TL(r)$ will constrain the quality contributing potential of subsequently placed activities. Since the *priority*maxdur* heuristic does not consider current availability on $TL(r)$, it is susceptible to placement decisions that inadvertently cause fragmentation of availability on $TL(r)$. In both of these cases, *reschedule-from-scratch* has no means to recover from bad decisions, since once an activity is placed into the sequence of activities on $TL(r)$ for some resource r, its position relative to previously placed activities is fixed and there is no possibility to reconsider the choice of r. Application of the *insert-or-bump* operator, alternatively, provides just this sort of capability - to alter previously fixed sequencing relationships between pairs of activities and to change resource assignments. This suggests a most costly, but potentially more effective schedule generation (or reschedule from scratch) procedure. To explore this hypothesis, we define a new schedule generation strategy by altering our original *reschedule-from-scratch* procedure to utilize the *insert-or-bump* operator instead of simple insert-only.

Another approach to providing a more effective rescheduling procedure is to embed our basic *reschedule-from-scratch* generation procedure within a broader search process. There are a number of possibilities here; squeaky wheel optimization (Joslin & Clements 1999), genetic algorithms (Gilbert Syswerda 1991), heuristic-based stochastic sampling (HBSS) (Bresina 1996). We've explored this approach by embedding our greedy-quality schedule generator within a variant of HBSS called value-based stochastic sampling (VBSS) reported in (Cicerello & Smith 2002). In brief, the idea in VBSS is to apply a given search heuristic in a non-deterministic manner, biased according to how well the heuristic discriminates in a given decision context. If the heuristic clearly favors one choice over others, then the tendency is to follow the heuristic with high probability; if the heuristic assesses several choices equivalently, then the decision is made with greater non-determinism. Using this scheme to repeatedly generate different solutions, some number of solutions in the neighborhood of the heuristic's trajectory through the search space are sampled and the best is retained. This scheme has been previously shown to produce high performance solutions in other scheduling domains with classical scheduling objectives. Here we apply VBSS to randomize the activity sorting heuristic *priority * maxdur* defined earlier.

Figure 5 compares the performance of the basic greedy-quality *reschedule-from-scratch* procedure with the more ex-

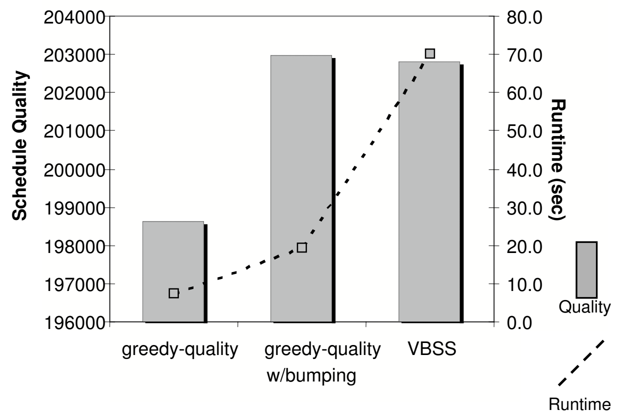

Figure 5: Extended rescheduling strategies (Quality and runtime for 3 schedule-from-scratch approaches over the 200-activity problems)

tended schedule optimization procedures just outlined. Here we chart just the baseline solutions for the same 200-activity problem set of Figure 4 (i.e. without the 10 additional activities added dynamically for each problem). As the results show, it is possible to produce a rescheduling strategy capable of outperforming our incremental techniques, but at significant increase in computational cost. Qualitatively, rescheduling with bumping takes about 3 times as long as the basic greedy constructor. The VBSS results shown are for 10 iterations which predictably takes about 10 times as long as the basic schedule generator. Indeed, the incremental *insert-or-bump* and *insert-or-bump+* strategies appear to strike a nice tradeoff between maintaining high quality solutions and reasonable computational cost.

Solution Stability

Finally, we consider our incremental quality-centric scheduling strategies from the standpoint of solution stability. In dynamic environments, an ability to minimize perturbation to the schedule when responding to new events can be fundamental to maintaining execution coherence. Characterizing schedule stability is a somewhat subjective matter, as user perception of it is likely to depend on the problem environment, work habits, and the impact of various changes on work process. Furthermore, definitions of schedule stability that have appeared in the literature (e.g., Policella et al. 2004) tend to center mainly on temporal flexibility, which has a different interpretation in classical scheduling domains than it does in quality-centric scheduling problems. For this paper, we adopt the following simple parameterized measure of schedule perturbation, $P_{schedule}$, which reflects our intuition of the importance of various decision variables in dynamic, quality-centric scheduling contexts.

$$P_{schedule} = w_1 \times b + w_2 \times r + d \qquad (6)$$

where *b* is the number of schedule activities removed (bumped), *r* is the number of scheduled activities reallocated to a different resource, *d* is the number of schedule activities

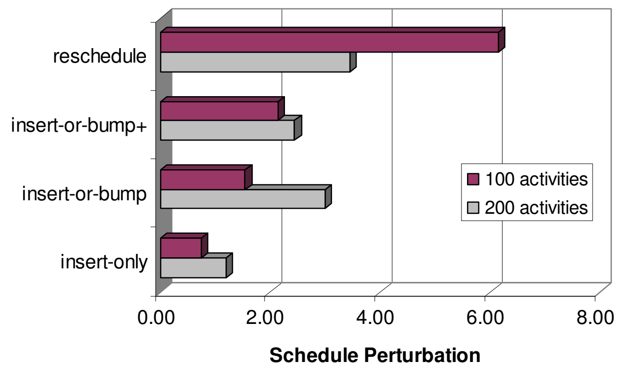

Figure 6: Perturbation of baseline schedule due to insertion, bumping, and rescheduling strategies

whose durations change, and w_1, w_2 are weighting factors. For this study we set w_1 and w_2 to 3 and 2 respectively, while acknowledging that these values reflect nothing more than our judgement of the impact of each type of disturbance in a *News Agency* domain schedule.

Using this metric, Figure 6 compares the average perturbation level relative to the baseline schedules for the incremental strategies and reschedule-from-scratch baseline. As one might expect, when the only option for accommodating a new activity is insertion into an existing timeline gap, perturbation is minimized. Perturbation increases with increasing problem oversubscription because fewer insertion opportunities necessitates increased adjustment to the durations of currently scheduled activities. As expected, rescheduling incurs the highest degree of baseline schedule perturbation. In this case, however, the level of perturbation diminishes as oversubscription increases. This is due to the fact that more heavily oversubscribed problems start with higher baseline schedule quality (due to the larger pool of activities to allocate from) and so fewer new activities are actually inserted.

Scheduling new activities with either of the bumping enhanced modes occupies the middle ground amongst these strategies. The results for the 200-activity set support the intuition that the *insert-or-bump+* mode is the less disruptive of the two, as it seeks to reinsert any activities that are bumped. However, this strategy exhibits *more* perturbation than the bump without reinsertion mode for the 100-activity set. Apparently reinsertions in this case incur significant resource and duration assignment changes.

Related Work

As mentioned at the outset of the paper, our work has much in common with previous research in reactive scheduling (Smith 1994; Zweben *et al.* 1994; Becker & Smith 2000; Sakkout & Wallace 2000; Bartak, Muller, & Rudova. 2004), which has similarly focused on the development of incremental techniques for managing schedules in dynamic environments. However, this research has centered on classical scheduling problems,with performance objectives relating to classical measures such as meeting due dates and maximizing throughput. Our work extends this line of research to scheduling problems where maximizing the quality of executed activities is central concern.

Our approach also shares similarities with several other models that have appeared in the literature. Our work and models such as *imprecise computation* (Liu *et al.* 1994), *IRIS* (Increasing Reward with Increasing Service) (Dey, Kurose, & Towsley 1996), *anytime scheduling* (Schwarzfischer 2004), *progressive processing* (Mouaddib & Zilberstein 1998) and *design-to-time scheduling* (Garvey & Lesser 1993) all share the common thread of addressing the tradeoff between service time on an activity and schedule quality. However with very few exceptions, other work has restricted attention to single resource (agent) scheduling processes. Much of the noted work also falls under CPU scheduling applications, where activities are assumed to be pre-emptable and resumable and switching from one job to the next can be accomplished without setup.

There have been some studies with these models, however, that incorporate some of the characteristics of our problem. For example, in (Kobayashi, Yamasaki, & Anzai 2003), wind-up activities were introduced within an imprecise computation model. Wind-up activities are roughly analogous to our setup activities, except that they must be performed after activity completion and have fixed durations. Another study (Tirat-Gefen, Silva, & Parker 1997) examines heterogeneous resources and non-preemptable activities in a CPU scheduling context. However, resources capabilities restrict which activities they can perform and different resource efficiencies are not modeled. More generally, none of these studies are concerned with optimizing quality.

The classic time/cost tradeoff problem in Operations Research (Kelley Jr. & Walker 1959), wherein the scheduler seeks to optimize time spent on each activity to minimize a project's cost and meet its deadline, is essentially an infinite capacity variation of our quality-centric problem. More recently (Wang & Smith 2005) and (Policella *et al.* 2005) have proposed alternative CSP scheduling models for solving a restricted version of our quality optimization problem that involves homogenous multi-capacity resources with no sequence dependent setup requirements. This work also focuses exclusively on the static version of the problem and does not consider dynamic arrivals.

Conclusions and Future Directions

In this paper, we have focused on methods for effective scheduling in dynamic environments where the utility of a schedule is measured by the quality that is accrued by its constituent activities. We have presented novel incremental techniques for generating, extending and maintaining such quality maximizing schedules, and have empirically evaluated their performance characteristics. Across a range of test problems, these techniques have been shown to accommodate dynamically arising events in a way that produces schedules comparable in quality to a reschedule-from-scratch baseline approach at significantly reduced computational cost while avoiding major disruption of the working schedule. To our knowledge, this is the first example of an incremental scheduler for dynamic quality-centric domains. We view it as a jumping-off point for investigations into quality-informed heuristics and broader-based search processes such as iterative improvement.

One such approach that appears promising involves use of our bumping operators as part of an anytime scheduling strategy. In preliminary experiments we have applied the *insert-or-bump* operator to the unscheduled activities which remain after first-pass scheduling on our testbed problem sets, and this action consistently produced additional quality gains. We anticipate extending this simple approach into a more ambitious post-processing strategy that heuristically bumps, inserts and swaps activities in the schedule within an iterative local search framework. Such tactics might serve to effectively break out of local quality maxima that the scheduler finds itself in due to biases in the quality-centric heuristics.

Acknowledgments

The work reported in this paper has been supported in part by the National Science Foundation under contract # 9900298 and by the CMU Robotics Institute.

References

Bartak, R.; Muller, T.; and Rudova., H. 2004. A new approach to modelling and solving minimal perturbation problems. *Recent Advances in Constraints*.

Becker, M., and Smith, S. 2000. Mixed-initiative resource management: The amc barrel allocator. In *Proceedings of the 5th International Conference on AI Planning and Scheduling*, 32–41.

Bresina, J. L. 1996. Heuristic-biased stochastic sampling. In *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference*, volume 1, 271–278.

Cicerello, V., and Smith, S. 2002. Amplification of search performance through randomization of heuristics. *CP*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 41(1–3).

Dey, J.; Kurose, J.; and Towsley, D. 1996. On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks. *IEEE Transaction on Computers* 45(7):802–813.

Garvey, A., and Lesser, V. 1993. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1491–1502.

Gilbert Syswerda, J. P. 1991. The application of genetic algorithms to resource scheduling. *Proceedings of the 4th International Conference on Genetic Algorithms*.

Joslin, D., and Clements, D. 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Kelley Jr., J., and Walker, M. 1959. Critical path planning and scheduling: An introduction.

Kobayashi, H.; Yamasaki, N.; and Anzai, Y. 2003. Scheduling imprecise computations with wind-up parts. In

Proceedings of the 18th International Conference on Computers and Their Applications, 232–235.

Liu, J.; Shih, W.-K.; Lin, K.-J.; Bettati, R.; and Chung, J.-Y. 1994. Imprecise computations. In *Proceedings of the IEEE*, volume 82, 83–94.

Mouaddib, A.-I., and Zilberstein, S. 1998. Optimal scheduling of dynamic progressive processing. In *European Conference on Artificial Intelligence*, 499–503.

Policella, N.; Wang, X.; Smith, S.; and Oddi, A. 2005. Exploiting temporal flexibility to obtain high quality schedules. In *Proceedings of the 20th National Conference on Artificial Intelligen*.

Sakkout, H. E., and Wallace, M. 2000. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints* 5(4):359–388.

Schwarzfischer, T. 2004. Closed-loop online scheduling with timing constraints and quality profiles on multiprocessor architectures. In *Proceedings of the 14Th International Conference on Automated Planning & Scheduling*. American Association for Artificial Intelligence.

Smith, S. 1994. Opis: An architecture and methodology for reactive scheduling. *Intelligent Scheduling*.

Tirat-Gefen, Y.; Silva, D.; and Parker, A. 1997. Incorporating imprecise computation into system level design of application-specific heterogeneous multiprocessors. In *Design Automation Conference*, 58–63.

Wang, X., and Smith, S. 2005. Retaining flexibility to maximize quality: When the scheduler has the right to decide durations. *Proceedings of the 15Th International Conference on Automated Planning & Scheduling*.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and rescheduling with iterative repair. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann Publishers.