Scheduling in the Face of Uncertain Resource Consumption And Utility

Jeremy Frank and Richard Dearden*

{frank,dearden}@email.arc.nasa.gov
Computational Sciences Division
NASA Ames Research Center, MS 269-2
Moffett Field, CA 94035

Abstract. We discuss the problem of scheduling tasks that consume a resource with known capacity and where the tasks have varying utility. We consider problems in which the resource consumption and utility of each activity is described by probability distributions. In these circumstances, we would like to find schedules that exceed a lower bound on the expected utility when executed. We first show that while some of these problems are $\mathcal{NP}$-complete, others are only $\mathcal{NP}$-Hard. We then describe various heuristic search algorithms to solve these problems and their drawbacks. Finally, we present empirical results that characterize the behavior of these heuristics over a variety of problem classes.

1 Introduction

In this paper we discuss scheduling problems for tasks that use quantities of consumable resources. Traditional scheduling problems consider tasks that have known resource consumption and known utility. However, we often encounter problems in which the resource consumption of a task is not known exactly, and instead we have only a probability distribution over resource use. For example, consider a scenario in which a remote sensing platform must analyze a set of data from different events that it has recieved. The utility of each event is unknown, as is the total power required to perform the analysis. Time is also limited, as new events will occur. The task is to schedule the analysis steps in order to maximize the expected utility.

Due to the uncertainty of the resource consumption, some scheduled tasks may not actually be performed when a schedule is executed. If we assume that we have accurate knowledge of the distribution of resource consumption and job utility, we can compute the *expected* utility of a schedule by accounting for both the uncertain resource consumption and utility. Under these circumstances, we can either find a schedule that maximizes the expected utility, or find a schedule whose expected utility exceeds a lower bound. We are also interested in cases where solving the decision problem exactly may be impossible due to the amount

* Research Institute for Advanced Computer Science

of time required. This is often the case in on-board planning that is tightly integrated with plan execution and limited by resource utilization [1, 2].

The paper is organized as follows. In §2 we present both theoretical and empirical results on the problem of scheduling tasks that have uncertain resource consumption, utility, or both. Initially, we consider the case of consumable resources only. We then show how to extend the results to reusable resources, such as power, and replenishable resources, such as fuel. Finally, we show that the crucial aspect of this problem is the uncertainty in resource consumption; problems in which the resource consumption is known and the utility is probabilistic are trivially reducible to the Knapsack problem.

The resulting problems are quite difficult to solve, and often the solution cost is higher still because of the need to do Monte-Carlo sampling to compute the convolution of probability distributions. Because of this cost and because of our interest in performing scheduling in time-limited and computational resource-bounded environments, in §3 we investigate heuristics for greedy search algorithms to solve these problems. We first describe some likely heuristics, and discuss drawbacks to them. We then perform an empirical study on simple versions of the problem. Our aim is to explore the space of problem features and possible heuristic approaches to the problem. We discuss the results of these investigations in §4. In §5 we describe some related work, and in §6 we conclude and discuss future work.

2 Theory

We begin with some informal definitions. A *consumable* resource is one that can be depleted but not renewed. A *replenishable* resource is a resource that can be depleted and renewed by different activity. A *reusable* resource is used for the duration of an action and returned when the action completes. Furthermore, we assume that activities consume resources when they begin execution; this is a bounding above approximation, but is useful in many circumstances, such as power use or block memory allocation. Similarly, if an activity replenishes a resource, we assume that the resource is replenished at the end of the activity. This defines the mapping from activities to *events*, and allows us to consider only the sequencing of the events of any activities that affect the level of the resource. In the case of consumable resources, there will only be one such event per task, while in the case of replenishable or reusable resources there may be two such events per task, one that consumes a resource and another that returns the resource.

In this section we describe a total of 4 problem classes. Problems are distinguished based on whether the resource is consumable or replenishable. We use an event based representation instead of an action based representation, since the case of reusable resources is subsumed by the more general case of replenishable resources.

Problems are also distinguished based on whether the schedule is forced to be a *permutation* of events, or whether the schedule is allowed to be a *partial*

order of events. This distinction is motivated by recent results that show then it is possible to validate schedules consisting of a partial order of events when resource consumption is known [3]. It turns out that the distinction matters when applied to problems with uncertain resource consumption.

We will now introduce some notation. A scheduling problem consists of a set of events, J, that consume a set of resources, $\boldsymbol{R} = R_1...R_i$. Let M_i be the maximum amount of resource R_i available at any time (i.e. it's capacity); without loss of generality let the minimum amount of resource available at any time be 0. We will denote a schedule by π. We will denote the x^{th} event in a schedule by π_x. Let j be an event that modifies a resource R. Let $P_j(R)$ be the probability distribution for the amount of resource R consumed by event j. Let $A_{x,\pi}(R)$ be the probability distribution over the amount of resource consumed after the execution of the x^{th} event in schedule π conditioned on the successful execution of all activities. Thus, $A_{0,\pi}(R)$ is the probability distribution over the initial resource distribution and $A_{x,\pi}(R)$ is the convolution of $A_{0,\pi}$ and P_{y,π_y} for all $y \leq x$ subject to the condition that the events succeed. Let $\overline{A}_{x,\pi}(R)$ be the probability distribution over the amount of resource consumed after the execution of the x^{th} event in schedule π conditioned on the successful execution of activities $1..x-1$. Let $P_j(U)$ be the probability distribution for the utility of job j. The task is to find a permutation of the events in J such that the expected utility of the schedule exceeds a bound U^*. In order to compute the expected utility, we need a model of task execution that defines job failure. We assume that if the resource used by the task is exhausted by the task that it fails, and provides no utility. Similarly, if a task requires a resource that is exhausted, it provides no utility upon execution.

2.1 Consumable Resources

We will now consider the case of consumable resources, that is, resources that are consumed by tasks and never returned or replenished. For now, we assume events have no absolute or relative temporal constraints. We also assume that the consumption of each event is determined only when the event is executed. Finally, we initially assume that events must be totally ordered, but we will consider the implications of relaxing this assumption. Thus, a schedule π is a total order of the events. Our goal is to find a schedule whose expceted utility exceeds a bound B which is an input to the problem. We refer to this problem as the *Uncertain Consumable Resource Scheduling Problem* (UCRSP).

Initially, we will assume that there is only one resource R with maximum capacity R. We will also assume without loss of generality that the probability that there is less than 0 resource initially available is 0. The expected utility is computed as follows: for each job in the permutation, convolve the probability distributions of the resource consumptions of the events that have executed to determine the probability distribution over the remaining resource amount. Next, determine the probability that the amount of resource remaining is greater than or equal to zero. This is the probability that the task succeeds based on its modification of resource R. We will denote the probability that task $\pi_x = j$

succeeds by $S_{x,\pi}(R)$, which is simply the probability that there is some resource remaining after the execution of π_x, or $\int_0^M \overline{A}_{x,\pi}(R)$. Using this value, we can now compute the expected utility of the job, which is simply $(S_{x,\pi})(EP_{\pi_x}(U))$. Thus, the expected utility of schedule π is

$$\sum_{i=1}^{n}(S_{x,\pi}(R))(Ep_{\pi_x}(U)) \tag{1}$$

Theorem 1. *UCRSP is in $\mathcal{NP}$.*

Proof. First, note we only need to convolve a linear number of distributions and compute a linear number of event utilities to compute the schedule utility. The multiplications and sums in the formula presented above are all polynomial time operations. All that remains is showing that the convolution operation is a polynomial time operation. In the worst case, we can do each convolution using Monte Carlo Integration, which takes constant time for a fixed error [4].

Theorem 2. *UCRSP is $\mathcal{NP}$-Hard.*

Proof. We will reduce the Knapsack problem to UCRSP. This problem consists of jobs with fixed resource consumption and utility, where the task is to find a partition of the items into those in the Knapsack and those not in the Knapsack, such that the Knapsack capacity is obeyed and the utility of the items in the Knapsack exceeds a bound. Thus, a Knapsack problem is simply a UCRSP where there is no uncertainty on the resource consumption, utility or the initial resource amount. A Knapsack item $j* = (r*, u*)$ where $r*$ is the capacity and $u*$ is the utility. Thus, we map $j*$ to a UCRSP event j with $P_j(R) = (p(r*) = 1)$ and $P_j(U) = (p(u*) = 1)$. The initial amount of resource R in the UCRSP is the bound on the Knapsack size $R*$. The utility bound of the Knapsack is mapped to the expected utility bound of our problem. This mapping is easily seen to be a linear time mapping.

Now consider a schedule π that satisfies the expected utility bound of the UCRSP. The job failure model provides no utility to any job whose resource consumption would exceed the resource capacity. Any permutation can be mapped into a partition of jobs by the following linear time procedure: For any event j having a non-zero utility in π, the corresponding $j*$ is in the Knapsack, and all other events are not in the Knapsack. Since all j in the UCRSP, the only possible way for an event j to contribute zero utility is for the event's resource consumption to exceed the resource bound. Thus, the set of Knapsack items must obey the Knapsack constraint. Further, by construction of the UCRSP, each event j that contributes utility is guaranteed to contribute all of is utility, since all such events execute with probability 1. Thus, the set of Knapsack items is also guaranteed to exceed the desired utility bound. It is clear from the simplicity of this mapping that the expected utility of the transformed job is the value of a solution to the Knapsack. Furthermore, a solution to the UCRSP is a solution to the Knapsack problem. Thus, UCRSP is $\mathcal{NP}$-Hard.

Corollary 1. *UCRSP is $\mathcal{NP}$-Complete.*

A further observation is that we can add temporal constraints to UCRSP and preserve $\mathcal{NP}$-completeness, by means of a reduction from UCRSP without temporal constraints. The only additional machinery needed is to observe that we can validate the temporal constraints in polynomial time using the results of Dechter, Meiri and Pearl [5].

Finally, we observe that scaling the UCRSP up to multiple resources does not increase the difficulty of the problem. Suppose there are multiple resources, $R_i ... R_j$. We now need the probability that task j succeeds given the state of all of the resources, which we denote $S_{x,\pi}(\boldsymbol{R})$, which is simply $\prod_i S_{x,\pi}(R_i)$. Using this value, compute the expected utility of the job, which is simply $S_{x,\pi}(\boldsymbol{R})(EP_j(U))$. The expected utility is

$$\sum_{i=1}^{n} S_{x,\pi}(\boldsymbol{R})(Ep_j(u)) \tag{2}$$

Now let us consider some relaxations. The first relaxation is to allow two events to be scheduled at exactly the same time. In this case, we have to modify the task execution model, and thus the failure model. One option is the "conservative" model, in which two events scheduled simultaneously result in a single resource allocation. In this case, if the joint resource request exhausts the resource, both tasks fail. Under these assumptions, the problem is still in $\mathcal{NP}$. However, this model is unlikely to be realistic, so we do not consider it of interest.

An alternative is to assume that two events scheduled simultaneously are executed in arbitrary order. Thus, it is equiprobable that either event occurs first. In this case, the problem is no longer known to be in $\mathcal{NP}$. The reason is that the certificate, a set of events such that the execution order is not determined, may not be verifiable in polynomial time. Consider an arbitrary set of simultaneous resource allocations. Is there a permutation of this set that exceeds a utility bound U^*? This is simply a version of UCRSP, which we have just shown is in $\mathcal{NP}$ under the assumption that we enforce a permutation of event occurences. Thus, if $\mathcal{P} \neq \mathcal{NP}$, then UCRSP with the relaxed certificate and the liberal event execution model is not in $\mathcal{NP}$.

The second relaxation is to permit the scheduler to return a partial ordering of the events rather than a total ordering. It is easy to see that this puts us in the same position as allowing two or more simultaneous events in a schedule. We can no longer guarantee that a schedule is a solution in polynomial time, because the validatation problem requires solving an $\mathcal{NP}$-complete problem. Note that there is an additional complication, which is determining the probability of any permutation of the unordered events when computing the expected utility.

2.2 Replenishable Resources

Now let us consider the situation where resources are replenishable. We also begin with the assumptions that there are no temporal constraints, and that

all events must be totally ordered in a schedule. Both of these assumptions will be relaxed in turn. We refer to this problem as the *Uncertain Replenishable Resource Scheduling Problem* (URRSP).

When computing the expected utility of a schedule, we must compute the probability that tasks fail because they allocate too much resource, as was true with consumable resources. In the case of replenishable resources, we must also compute the probability that tasks produce too much of resource R. Note that this is admirably handled by the fact that the convolution of the resource distributions is well defined for both increasing and decreasing the amount of resource available after event execution.

As before, we will denote the probability that task $\pi_x = j$ succeeds $S_{x,\pi}(R)$, which is simply the probability that there is neither too much nor too little resource remaining after the execution of the job at position x in schedule π. Again, this is $\int_0^M \overline{A}_{x,\pi}(R)$. Using this value, we again can compute the expected utility of the job, which is simply $(S_{x,\pi})(EP_{\pi_x}(U))$. Thus, the expected utility of π is

$$\sum_{i=1}^{n}(S_{x,\pi}(R))(Ep_{\pi_x}(U)) \tag{3}$$

The generalization to multiple resources is also straightforward; $S_{x,\pi}(\boldsymbol{R}) = \prod_i (S_{x,\pi}(R_i))$.

Theorem 3. *URRSP is in $\mathcal{NP}$.*

Proof. As we saw for UCRSP, the expected utility computation only requires convolutions, additions and sums, all of which are accomplished in polynomial time.

Theorem 4. *URRSP is $\mathcal{NP}$-Hard.*

Proof. We reduce the UCRSP to the URRSP by observing that a UCRSP is a URRSP with no events that produce the resource.

We observe as before that adding temporal constraints preserves $\mathcal{NP}$-completeness using the same argument we used for the UCRSP.

We now consider relaxing the assumptions on the total ordering of events in the schedule. Just as before, schedules are no longer verifiable in polynomial time if we allow events to be simultaneous, or if we are forced to try and verify schedules where sets of events are unordered.

Finally, we must say a few words about the distinction between renewable resources and reusable resources in the context of uncertainty. A reusable resource is one that is allocated by an activity for a period of time, then returned for other activities to use. Reusable resources can be modeled using renewable resources quite easily; an event that consumes the resource represents the reusable activity start, while an event that represents the replenishment represents the end. The replenishment is constrained to replenish the *same* amount of resource that the

start event used. This causes a minor problem in modeling for the case of uncertain resource impacts; the underlying events must be constrained so that there is no uncertainty in the amount of replenishment of the resources. The complexity results are identical as for the case of URRSP. However, the underlying problems are different enough that a different representation and heuristics may be useful when solving the problems.

2.3 Uncertain Utility is Uninteresting

As a final point, we note that the interesting aspects of these problems are the uncertainty in the resource consumption. First, consider the problem of uncertainty in the utility with certainty in the resource consumption and no temporal constraints. The problem now is identical to the Knapsack problem. The task is to find those tasks that can be executed (i.e. put in the Knapsack) whose expected utility exceeds a bound. From probability, $E(P_j(U)P_k(U)) = EP_j(U) + EP_k(U)$, so this is simply another Knapsack problem. The problem with temporal constraints added simply limits the tasks that can simultaneously be in the Knapsack. Another aspect of the problem with uncertain resource consumption that is of interest is that tasks in a schedule can be partitioned into roughly 3 sets: those guaranteed to execute, those guaranteed not to execute, and those that may execute if tasks scheduled earlier do not overconsume (or overproduce) the resource. This is a more interesting problem than the traditional scheduling problem with job utility, where tasks are either accepted or rejected. It is not possible to reject tasks out of hand until the resource is exhausted with probability 1.

2.4 Summary

In summary, the problem of finding totally ordered schedules of activities with uncertain resource impact and uncertain reward such that the expected utility exceeds a bound is $\mathcal{NP}$-complete. However, the problem of producing a flexible job ordering is not in $\mathcal{NP}$ if $\mathcal{P} \neq \mathcal{NP}$ because the problem of validating the flexible schedule is itself an $\mathcal{NP}$-complete problem. This is in stark contrast to the case of scheduling jobs whose resource consumption is known for certain [1].

Figure 1 summarizes these results. Note that we have omitted the results assuming the conservative event execution model where simultaneous events make a joint resource demand and the simple version of the problem with uncertainty only in the utility of the events.

3 Practice

The above results place us in the unenviable position of coming up with algorithms and heuristics that are likely to perform well on this problem. We have

[1] Note that the reference is for non-rejectable jobs without utility. As long as jobs are definitely included or not included in a schedule, the exact order of the jobs can be left up in the air and certificates can still be validated in polynomial time for the case of scheduling jobs such that the reward exceeds a bound

Problem	Resource	Ordering	Complexity	Reduction
UCRSP	Consumable	Total	$\mathcal{NP}$-complete	Knapsack
UCRSP	Consumable	Partial	$\mathcal{NP}$-Hard	Knapsack
URRSP	Replenishable	Total	$\mathcal{NP}$-complete	UCRSP
URRSP	Replenishable	Partial	$\mathcal{NP}$-Hard	UCRSP

Fig. 1. Complexity of various problems of scheduling with uncertain resource consumption.

the usual panoply of choices; constructive search algorithms driven by heuristics or local search algorithms driven by gradients and neighborhoods. As we mentioned before, we are currently interested in fast approximation algorithms for on-board scheduling. In the following sections, we will focus on heuristics to drive greedy sampling approaches.

3.1 Heuristics and their Discontents

In this section we describe some heuristics that might be used to drive solutions to this problem. For the following discussions we adopt the following notation. Suppose we have built a schedule prefix π_x. Now suppose that the jobs left are denoted by H and that the current utility is u_{π_x}. Recall that the distribution of the remaining resource amount after x executes is denoted $\overline{A}_{x,\pi}(R)$.

Intuitively, since our task is to maximize the expected utility of the schedule, we could employ a heuristic that chooses the next event by greedily maximizing the utility for all remaining events. Since the prefix π_x is fixed, this policy greedily maximizes the utility of the resulting schedule. We can compute the utility u_h of each $h \in H$ given $\overline{A}_{\pi,x}(R)$. The heuristic, then, would select $h'|u_{h'} \geq u_h$. Note that this heuristic has an obvious generalization to multiple resources.

This heuristic doesn't work well for a situation in which the expected resource consumption conditioned on the event succeeding is larger for the event whose utility is higher. So even though the expected value of the schedule looks better for this event alone, looking ahead would reveal that it is better to take the event with lower utility. For example, suppose we have a problem instance where the resource R has 3 units initially, and the following events:

Event	$P_j(R)$	$P_j(U)$	u_j
1	U [1.5, 4.5]	3	1.5
2	U [0.5, 1]	1	1
3	U [0.5, 1]	1	1
4	U [0.5, 1]	1	1

In this case, the heuristic would have us choose event 1 first; even though it has a non-zero chance of failure, its expected utility is 1.5, while the expected utility of the other events is 1. However, it's clear that any permutation of events 2,3 and 4 will exceed the expected utility of event 1; we can put event 1 last and get a schedule of higher expected utility than any schedule with event 1

scheduled eariler. Furthermore, this heuristic fails to distinguish the case where two jobs have the same expected utility: for example,

Event	$P_j(R)$	$P_j(U)$	u_j
1	U [1.5, 4.5]	10	5
2	U [2, 5]	15	5

In this case, if there are more jobs, the downstream impact of the higher resource consumption on the schedule might warrant choosing event 1 first, even though event 2 has the higher utility. We shall call this heuristic E.

Suppose we choose the event that minimizes the expected resource consumption over all events without considering the state of the resource, i.e. $h'|EP_{h'}(R) \leq EP_h(R)$. Again, this has the result that the resource consumption of the resulting schedule is minimized, and consequently, maximizes the probability of success of the resulting schedule. This heuristic doesn't work well for a problem that is only a slight modification of the previous problem: again, let the resource R have initial endowment of 3 and change the event data as follows:

Event	$P_j(R)$	$P_j(U)$	u_j
1	U [1.5, 4.5]	9	4.5
2	U [0.5, 1]	1	1
3	U [0.5, 1]	1	1
4	U [0.5, 1]	1	1

It is now better to schedule event 1 first because scheduling the lower resource consuming events first does not overcome the high utility of the first event. Furthermore, this heuristic does not have an obvious generalization to many resources; it may not be possible to minimize the expected resource consumption on all resources simultaneously. We shall call this heuristic R.

A related heuristic contemplates the resource impact assuming that an event succeeds, that is, does not deplete the resource. We choose the event that minimizes the expectation over these distributions, i.e. $h'|E(\overline{A}_{\pi,x}(R)|\pi_x = h') \leq E(\overline{A}_{\pi,x}(R)|\pi_x = h)$. This heuristic will try and leave as much resource left as possible conditioned on event execution success. However, it does so by ignoring the event utility. It also doesn't generalize well to multiple resources. Finally, it may require a more complex computation to compute and thus may not be worth the cost. We shall call this heuristic S.

4 Empirical Results

To test the performance of the heuristics we described above, we performed a number of experiments on relatively simple, random domains. We considered problems with between ten and 20 jobs to be scheduled, and with approximately half that many constraints. Each of the jobs had a Gaussian distribution for the quantity of resources it consumed, with a range of values for the means. We considered problems in which the jobs had uniformly low variance, uniformly high variance, and random variance, and we varied the resource limit between

ten percent and 50 percent of the expected resource requirement for all the jobs. For each setting of these parameters, we generated 100 problems, and ran each of the heuristics on each problem.

We evaluated the heuristics by using them greedily to select a single valid schedule. We then computed the expected value of that schedule as shown in Equation 1. The performance of the three heuristics was consistent over all sizes of problems and resource limits, so we show the results for a single setting of those parameters in Table 1. In this case, the problems had 20 jobs, ten constraints, mean resource usages for the jobs uniformly distributed between ten and 50, job utilities uniformly distributed between one and ten, and a resource limit of 60 (ten percent of the expected resources required by all the jobs). We were particularly interested in the effects on the algorithms of changing the variance of the resource usage of the jobs, so we present results for three different resource usages.

Table 1. Performance of the three heuristics on problems with 20 jobs, 10 constraints, and a resource limit of 60.

Job Variance Range	Heuristic	Mean schedule value	Variance
0.1–1.0	E	20.35	26.20
	R	18.52	33.92
	S	17.53	32.45
0.1–0.2	E	13.88	32.88
	R	11.71	45.39
	S	11.18	26.55
0.8–1.0	E	13.91	37.71
	R	11.72	44.91
	S	11.63	43.91

As the table shows, the E heuristic (choose the job that maximizes the expected utility of the schedule built so far) considerably outperforms the other two on essentially all these problems. The only exception is on a few very small problems on which both E and R are finding optimal, or very close to optimal schedules.

In §3.1 we described an example of where the E heuristic performs badly (the first table). This corresponds to problems where there is a strong correlation between the resource requirements of a job and its utility. Intuitiion suggests that this may be a fairly common situation, so we did some additional experiments on problems with this property, as well as the opposite, where there is anti-correlation between resource usage and utility. The anti-correlated case is intuitively easy to solve as the jobs with high utility are cheap to perform, and vice versa. The results of these experiments, with the variances ranging from 0.1 to 0.2, are shown in Table 2.

Table 2. Performance of the three heuristics on problems with correlated and anti-correlated resource usage and utility. The other parameters are the same as Table 1, but variance is fixed at 0.1 — 0.2.

Problem Type	Heuristic	Mean schedule value	Variance
Uncorrelated	E	13.88	32.88
	R	11.71	45.39
	S	11.18	26.55
Correlated	E	10.08	0.07
	R	7.22	0.57
	S	8.02	0.78
Anti-correlated	E	27.7360	50.03
	R	27.7364	50.04
	S	23.64	51.13

As Table 2 shows, even in problems deliberately designed to exhibit characteristics that make E perform badly, it still tends to outperform R and S. On the anti-correlated (easy) problems R actually performed slightly better than E, but these results are not statistically significant. In fact, both heuristics produce very similar schedules for these problems, and appear to perform very close to optimal (on small problems we have computed the optimal for).

One problem with using the E heuristic is that it takes approximately 15 times as long to find a schedule as the other two, due to the complexity of the Monte Carlo estimate of the value of the whole schedule at each step. As we said in Section 2.1, the Monte Carlo estimate is necessary because we need to compute the probability of success of each job, given that all previous jobs succeeded, to compute the utility of a schedule. One possible approximation to this is to ignore the condition that previous jobs succeeded, and instead use the probability that the schedule up to a particular job will complete in the given amount of resources. This is easily computed for Gaussian resource usage distributions as it it simply the sum of the usages for the jobs, which is itself a Gaussian. However, it overestimates the value of each schedule. Table 3 shows results on the same set of problems using this approximation, again only for the low variance case. There are two interesting results in this table. The first is that the approximation actually beats E for uncorrelated problems, by a statistically significant amount. We are currently investigating why this occurs, but our intuition is small jobs, that is those that use few resources, gain more from the approximation than large jobs, so it favours small jobs at the beginning of the schedule, which is good for cases such as this with tight resource bounds. We are currently running more experiemnts to test this hypothesis. The second interesting point is that on the other problems, the approximation performs comparably to R, and is in fact worse on anti-correlated problems. The computation time is still somewhat larger (a factor of around 2) for the approximation, which suggests that there is relatively little advantage to using the approximation over using R for many

Table 3. Performance of the approximation to the E heuristic on the same problems as Table 2.

Problem Type	Heuristic	Mean schedule value	Variance
Uncorrelated	E	13.88	32.88
	Approximation to E	16.07	26.75
	R	11.71	45.39
Correlated	E	10.08	0.07
	Approximation to E	8.01	0.78
	R	7.22	0.57
Anti-correlated	E	27.74	50.03
	Approximation to E	23.64	51.16
	R	27.7364	50.04

problems. Again, we are currently conducting additional experiments to quantify exactly when one heuristic should be preferred over the other.

5 Related Work

There has been considerable work on a variety of planning and scheduling problems with various assumptions of uncertainty and with various objective functions. Early papers on this topic discuss the computational complexity of simple shop scheduling problems with exponentially distributed release times, due dates, and durations, where the goal is to minimize the expected makespan of the schedule or a weighted time objective [6, 7]. One result of interest in [7] is that minimizing the weighted sum of *late* jobs under randomly distributed job characterisitcs is in $\mathcal{P}$. At first glance this looks similar to the problems we describe; however, UCRSP and URRSP are different because we impose a fixed limit on the total resource use, which is equivalent to saying that the deadlines of all jobs are fixed; we also impose precedence and resource constraints. A nice extension to this work is the Time-Dependent MDP formulation of [8].

Other work discusses modification of standard algorithms like A* to work in settings where job data is randomly distributed [9]. Additional work describes an MDP-based approach to handling production scheduling where the job duration and utility of each action is uncertain [10]. Again, UCRSP and URRSP differ from this problem in that they cannot be modeled as an MDP; the actual utility of any action is a function of the resource state, which is a function of the entire schedule to date, which violates the Markov property. Finally, the limited amount of work on maximizing schedule utility under resource uncertainty in an on-board setting does not handle the problem by posing it in the manner that we have considered it here [2, 1].

With respect to planning, many previous approaches also make Markov-like assumptions about action failure [11, 12]. A recent exception is the work of [13], in which the goal is to find contingent plans under conditions where the probability of action failure depends on the state of a resource that is affected by the entire

plan prefix. Our work differs in that we limit ourselves to the case of scheduling; this makes it tractable to compute the utility of actions, and allows us to provide complexity results.

Finally, we should point out that the maximization of expected utility is a global constraint unlike any considered by previous work in this area. Most such constraints handle situations in which there is no uncertainty, and concentrate on various types of constraint satisfaction. Thus, this paper should serve as a challenge to members of the CP community to develop global constraints for maximizing expected utility.

6 Conclusions and Future Work

We have described a series of problems involving scheduling under uncertainty of resource consumption and event utility. We provide complexity results for these problems, and demonstrate that different heuristics provide different results under a variety of problem classes.

The results in this paper are given for general probability distributions, and the experiments focus on one example, the Gaussian distribution. However, it is important to determine whether the probability distributions for resource consumption have an impact on the results. While it seems that the most important feature may be the relationship between the relative means and variances of the resource consumption and the utilities of the events, there may be significant secondary effects due to the shape of the distribution. Thus, further empirical and theoretical analysis is warranted.

A useful heuristic is a *strict dominance* criteria for eliminating some candidate events from consideration as the next choice, such as those discussed in [9]. Suppose we have two events with the same resource consumption distribution but different (known) utilities. Clearly it is better to choose the event of higher utility. However, it is not clear how likely this is to happen in practice. Also, it is possible that no single event will be strictly dominated by all other events, and thus makes it even less likely that any reduction in the number of heuristic evaluations can be done. Finally, dominance must be extended to the case of totally ordering events with respect to uncertain resource consumption and utility to be generally applicable.

We can imagine situations in which many tasks are identical. In this case, the number of total schedules is smaller than $N!$; if there are k distinct classes of tasks such that we have N total tasks, and n_i of each task from $i = 1..k$, then the total number of schedules is $\frac{N!}{\prod_{i=1}^{k} n_i!}$ Suppose now that we have a set of jobs such that some of them are not identical, but are very close to identical. Rather than searching all possible permutations, it might be beneficial to cluster jobs and reduce the search space. We would like to find a bound on the error in the expected utility under approximations like this. Unfortunately, it seems difficult to do this. It is also uncertain if there are classes of problems that have this property.

We can form various combinations of the heuristics by forming linear combinations and using one as the tie-breaker for another and so on. Some promising combinations are as follows:

- Sort by E first, then by R.
- Sort by E first, then by S.
- Linear combination of E and R.
- Linear combination of E and S.

Linear combination requires that we convert the heuristics from a relative ranking scheme into a function. This can be done using the utility or the expectation as the basis of the heuristic, and dividing the result for each event by the sum. Bias functions can also be used to emphasize the best choices if desired [14].

References

1. Khatib, L., Frank, J., Smith, D., Morris, R., Dungan, J.: Interleaved observation execution and rescheduling on earth observing systems. In: To Appear, Proceedings of the ICAPS Workshop on Plan Execution. (2003)
2. Shriver, P., Gokhale, M., Briles, S., Kang, D., Cai, M., McCabe, K., Crago, S., Suh, J.: A power-aware, satellite-based parallel signal processing scheme. In: To Appear, Proceedings of the DARPA Power Aware Computing Conference, Kluwer (????)
3. Muscettola, N.: Computing the envelope for stepwise-constant resource allocations. Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming (2002)
4. Hammersley, J.M., Handscomb, D.C.: Monte Carlo Methods. J. Wiley (1964)
5. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–94
6. Pinedo, M.: On the computational complexity of stochastic scheduling problems. In Dempster, M., Lenstra, J.K., Kan, A.R., eds.: Deterministic And Stochastic Scheduling. Springer Verlag (1982) 355–365
7. Pinedo, M., Schrage, L.: Stochastic shop scheduling: A survey. In Dempster, M., Lenstra, J.K., Kan, A.R., eds.: Deterministic And Stochastic Scheduling. Springer Verlag (1982) 181–196
8. Boyan, J., Littman, M.: Exact solutions to time-dependent MDPs. In: NIPS. (2000) 1026–1032
9. Wurman, P., Wellman, M.: Optimal factory scheduling using stochastic dominance a*. In: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence. (1996)
10. Schneider, J., Boyan, J., Moore, A.: Value function based production scheduling. In Shavlik, J., ed.: Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA (1998) 522–530
11. Kirman, J., Nicholson, A., Lejter, M., Dean, T., Jr., E.S.: Using goals to find plans with high expected utility. In: Proceedings of the 2d European Workshop on Planning. (1993)
12. Onder, N., Pollock, M.: Conditional probabilistic planning: A unifying algorithm and effective search control mechanisms. In: aaai99. (1999)

13. Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., Washington, R.: Incremental contingency planning. In: Submitted to ICAPS Workshop on Planning Under Uncertainty. (2003)
14. Bresina, J.: Heuristic-biased stochastic search. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence. (1996)