Resource-Bounded Scheduling Under Uncertainty

Jeremy Frank
Richard Dearden*
NASA Ames Research Center, MS 269-2
Moffett Field, CA 94035-1000,
{frank,dearden}@email.arc.nasa.gov

Abstract

We discuss the problem of scheduling tasks that consume uncertain amounts of a resource with known capacity and where the tasks have uncertain utility. In these circumstances, we would like to find schedules that exceed a lower bound on the expected utility when executed. We describe several different event execution models that motivate different formulations of the expected value of a schedule. We show that the problems are $\mathcal{NP}$-complete, and present results that characterize the behavior of some simple heuristics over a variety of problem classes.

Introduction

Consider a hypothetical space mission designed to observe objects of different characteristics with an instrument. The spacecraft has a limited amount of onboard data storage and power. Each observation requires an uncertain amount of power and data storage, and has uncertain scientific value. Data can be transmitted back to Earth, but transmission rates are uncertain. Finally, there may be dependencies among observations. Some observations may depend on calibration observations or other events. This both induces precedence constraints, and the failure of the calibration implies that the dependent observation need not be performed.

This scenario is not really hypothetical; many scheduling tasks derived from space missions and other applications have such characteristics. A wide variety of approaches have been used to address such problems. Simple dispatch schemes make decisions about the next task based on the current state of the system, but without considering the downstream impact of these decisions. More sophisticated approaches integrate a long-range planner with a reactive plan execution system using a deterministic action outcome model. The long-range planner generates a plan far into the future. The reactive planning system modifies this plan when the world changes (KFS+03) . Other techniques use explicit models of uncertainty to drive the search for schedules. MDP-based approaches can be used to generate "policies" that handle all possible execution paths (SBM98; BL00). MDP approaches face problems: usually these approaches cannot handle continuous action outcomes, scale only to limited problem sizes, and mappings from state to action are hard to understand. Other approaches include *contingent scheduling*, where a limited number of contingencies are generated to produce schedules with the best expected value (DBS94; MSW03).

In this paper, we consider a simple model in which the schedule execution system is unable to do rescheduling, and it is too expensive to store a contingent schedule or MDP. In this case, a schedule consists of a permutation of the tasks. Due to the uncertainty of resource consumption, some scheduled tasks may not actually be performed when a schedule is executed or may "fail" and provide no utility. If we assume that we know 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 the uncertain utility. We can then find a schedule that maximizes the expected utility, or find a schedule whose expected utility exceeds a lower bound.

Different scenarios motivate different schedule execution models and definitions of the expected value of a schedule. For example, it may be possible to *predict* the amount of resource required by a task before it is finished. An example of this is so-called quick-look analysis common in astronomy to predict the amount of time or data required by an observation. If the predicted resource use would exceed the resource available, execution of the event can be terminated without using the resource. Another example is memory usage—an image that requires more memory to store than is currently available is lost, but the memory remains free. In other situations this may not be possible. For example, power utilization often cannot be predicted beforehand. Thus, the execution system may be informed that the resource constraint is violated during event execution. These distinctions will generally lead to different amounts of resource availability, and thus affect the probability that an event succeeds. Finally, dependencies both define the set of legal schedules, and affect the probability that an event succeeds. One example of this is the calibration example described above: it may

*Research Institute for Advanced Computer Science

be pointless to perform an observation unless a calibration has been performed. Another example arises when modeling activities that affect *reusable* resources. If the start of the activity fails to execute, then clearly the end of the activity also should fail to execute.

In previous work (FD03) we describe an initial set of complexity results for the problem of finding a schedule that exceeds a given expected utility bound. In the present work we significantly expand the variety of execution models that underly the decision problem, and revise the complexity results to account for the numerous expectation formulae that result. Computing the expectation for these problems typically requires numerical integration of arbitrary probability distributions; we also precisely characterize the impact of this error on the decision problem.

The paper is organized as follows. We first present the different execution models. We then describe how numerical integration error affects the formulation of the decision problems. We then review the theoretical results on the decision problems. Next, we describe some likely heuristics, and discuss drawbacks to them. We then briefly discuss some empirical results for a simple problem under two different execution models.

Theory

We first introduce some notation. Let X be a set of events, and let R be a set of resources. Let r^{max} be the capacity of $r \in R$; at all times, the amount of available resource is bounded between 0 and r^{max}. Let $I_r(z)$ be the probability distribution over the initial amount of available resource r. We will assume without loss of generality that $z < 0 \Rightarrow I_r(z) = 0$ (that is, that the system always begins with positive amounts of resource). Define $C_{x,r}(z)$ as the probability distribution over the change in availability of resource r after executing x. We assume that all events' resource impact probabilities are independent; that is, the distribution for an event j does not depend on the distribution of any other event $k \neq j$. Define $U_x(w)$ as the probability distribution over the utility received from executing x. Let $P = \tau_i(x, y)$ be a set of binary precedence constraints over pairs of events x, y. These constraints require that x precedes y, and that if x fails then y fails as well.

We will denote a schedule by π and the j^{th} event in a schedule by π_j. We will assume that only schedules that are guaranteed to satisfy all $T = \tau_i(x, y)$ can be executed. However, these schedules are not guaranteed to satisfy the resource bounds or the conditional failure constraints. During execution, an event *fails* if its resource requirement (sampled according to $C_{x,r}(z)$) violates the resource bound for r given the currently available resource. Similarly, if $x \in \tau_i(x, y)$ fails to execute, then y also fails to execute. We assume that any event that fails to execute contributes no utility to the schedule. Our task is to build a schedule π to either maximize the expected utility $E(\pi)$, or such that $E(\pi) \geq B$. Throughout this paper we will focus on the decision problem.

Initially, we will assume that there is only a single resource r, and that r has maximum capacity r^{max}. We also consider two different classes of problem. If $C_{r,j}(z) > 0$ only when $z < 0$, we will call this problem the *Uncertain Consumable Resource Scheduling Problem* (UCRSP). If either $C_{r,j}(z) > 0$ only when $z < 0$, or $C_{r,j}(z) > 0$ only when $z > 0$ but not both, we refer to this problem as the *Uncertain Replenishable Resource Scheduling Problem* (URRSP). In principle, we could consider a model in which an event has a nonzero probability of either consuming or replenishing an event, but this scenario doesn't strike us as realistic.

The precise definition of $E(\pi)$ depends on the *event execution model* as described in the introduction. We provide formal definitions for these two models below.

Open Loop Model

Our first execution model assumes that the execution system blindly dispatches events without knowledge of those events' demands on the resource. Some underlying system informs the execution system of the resulting resource state and event success or failure, and if a job fails due to a resource r, it uses all the remaining resource. We call this the *Open Loop Model*.

We define $A_{\pi_j,r}(z)$ as the probability distribution over the availability of resource r after the execution or rejection of the first j events of π. For convenience, we define $A_{\pi_0,r}(z) \equiv I_r(z)$. Again for convenience, we define $T_{\pi_j,r}(z)$ as follows:

$$T_{\pi,r}(z) = A_{\pi,r,j-1}(z) * C_{\pi_j,r}(z) \tag{1}$$

Intuitively, $T_{\pi_j,r}(z)$ is the resource availability distribution after accounting for the impact of $C_{\pi_j,r}(z)$. This can violate the resource bounds for r, in that $T_{\pi_j,r}(z)$ may exceed 0 for $z < 0$ or $z > r^{max}$.

We can now compute the probability that π_j successfully executes, conditioned on the previous $j-1$ events. The probability π_j fails solely due to insufficient resources is

$$R(\pi_j, r) = 1 - \int_0^{r^{max}} T_{\pi_j,r}(z)dz \tag{2}$$

Let $\mathcal{P}(\pi_j) = x | \tau_i(x, \pi_j) \in P$. We always know any x precedes π_j in π, so we can now compute the probability π_j fails:

$$S(\pi_j, r) = \left(\prod_{x \in \mathcal{P}(\pi_j)} S(x, r) \right) R(\pi_j, r) \tag{3}$$

This formula says that event π_j will be rejected if it attempts to allocate more resource than r has available after the successful execution of the first $j-1$ events of π, and succeeds otherwise. Thus, execution of event π_j succeeds if the resource bounds will not be exceeded, which happens with probability $S(\pi_j, r)$. The formula also accounts for the probability that the conditional failure constraints are satisfied.

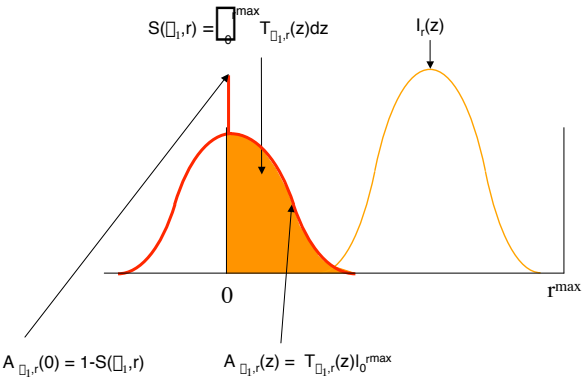

Figure 1: Computing $A_{\pi_1,r}(z)$ under the open loop model.

For simplicity, we will first develop the expectation formula for UCRSP, then use this to develop the formula for URRSP. In UCRSP we assume that resource is always consumed, and so the recurrence for $A_{\pi_j,r}(z)$ is as follows[1]:

$$A_{\pi_j,r}(z) = T_{\pi_j,r}(z)|_0^{r^{max}} \tag{4}$$

$$A_{\pi_j,r}(0) = \int_{-\infty}^{0} T_{\pi_j,r}(z)dz = R(\pi_j, r) \tag{5}$$

Figure 1 shows how this works for computing $A_{\pi_1,r}(z)$.

For the UCRSP, the failure of event π_j implies failure of $\pi_k, k > j$. The probability of successfully executing only the first i events of schedule π is given by

$$X(\pi_i) = (1 - S(\pi, r, i+1)) \prod_{j=0}^{i} S(\pi, r, j) \tag{6}$$

(where we define $S(\pi, r, n+1) = 0$). The expected utility of these i events is $\sum_{j=1}^{i} E(U(\pi_j))$. So the expected utility of the schedule π for UCRSP is given by

$$\mathrm{EU}(\pi) = \sum_{i=1}^{n} X(\pi_i) \left(\sum_{j=1}^{i} E(U(\pi_j)) \right) \tag{7}$$

In preparation for writing the expectation of URRSP we introduce the following definition.

[1]Our earlier work (FD03) implicitly used this model but the update formula was incorrect; it has been corrected in the present paper.

Definition 1 *Let π be a permutation of jobs for a URRSP. A* monotone subsequence *of π is a sequence of events all of which modify the resource in the same way, i.e. all producers or all consumers. Let $M_i(\pi)$ be these subsequences. In linear time, we can identify all $M_i(\pi)$, and there are $m \leq n$ of them. Let $M_i(\pi)$ be a* consuming subsequence *if $\forall j \in M_i(\pi) C_{r,j}(z) > 0$ only when $z < 0$, and a* producing subsequence *if $\forall j \in M_i(\pi) C_{r,j}(z) > 0$ only when $z > 0$.*

We can now compute the expected value $\mathrm{EU}(\pi)$ for URRSP as follows. First, observe that each $M_i(\pi)$ defines a UCRSP problem. If $M_i(\pi)$ is a consuming subsequence, this is obvious. If $M_i(\pi)$ is a producing subsequence, we can treat productions as consumptions and invert the resource bounds. Equation 7 shows how to compute $E(M_i(\pi))$ But $E(\pi) = \sum_{i=1}^{m} E(M_i(\pi))$; this is because the contribution of each $M_i(\pi)$'s utility is solely dependent on the resource availability distribution and conditional probabilities when the subsequence begins. Another way of thinking about this is that we are taking $E(\prod_i X_i)$ for all X_i independent. Each X_i corresponds to the UCRSP derived from a monotone subsequence $M_i(\pi)$; it takes only polynomial time to construct each X_i, since all we need to do is find $A_{\pi,r,j}(z)$ to set up each $I(z)$, and $\mathcal{P}$ for each element of the new subsequence, which we have shown that we can do above. From elementary probability, we know $E(\prod_i X_i) = \prod_i E(X_i)$ where all of the X_i are independent.

Closed-Loop Model

Our second execution model assumes that, prior to actual execution, the execution system is informed of the exact impact on the resource. We assume that if executing an event would lead to a violation of the resource bounds that the event is discarded, the resource is unmodified, and that no utility for the job is accrued. In other words, jobs that fail due to any resource consume no resource[2] We call this the *Closed Loop Model*, since (at least implicitly) the execution system performs a sensory action on the event and then decides how to proceed. The essential difference between the two models for UCSRP is that here you can attempt to execute jobs even after one has failed, assuming that all their constraints are met.

We preserve the intuitive definitions of $T_{\pi_j,r}(z), A_{\pi_j,r}(z)$, $R(\pi_j, r)$ and $S(\pi_j, r)$. We modify the recurrence of $A_{\pi_j,r}(z)$ as follows: if the event succeeds, we chop $T_{\pi_j,r}(z)$ so that it is non-zero only between 0 and r^{max}. The rest of the time the resource distribution remains unchanged. We can now write the following recurrence for $A_{\pi_j,r}(z)$:

$$A_{\pi_j,r}(z) = T_{\pi_j,r}(z)|_0^{r^{max}} + \frac{Y(z)}{1 - S(\pi_j, r)} \tag{8}$$

[2]Note that we could have different execution models for different resources, for example open-loop for power but closed-loop for memory. We will not consider this case here.

where $Y(z)$ can be thought of as the "part" of the distribution $A_{\pi_{j-1},r}(z)$ which can lead to exhaustion of the resource when π_j is executed. $Y(z)$ is defined below:

$$Y(z) * C_{\pi_j,r}(z) = T_{\pi_j,r}(z)|_{-\infty}^{0} \quad (9)$$

Under this model, failure of an event does not mean failure of the rest of the schedule. We are now in a position to write the expected value of a schedule π:

$$\mathrm{EU}(\pi) = \sum_{i=1}^{n} S(\pi_j, r) E(U(\pi_j)) \quad (10)$$

This formalization works for describing the expected value of a schedule for either UCRSP or URRSP.

Multiple Resources

In closing, we observe that scaling up to multiple resources does not increase the difficulty of the problem. Suppose there are q resources. For the closed-loop model, we define $S(\pi_j) = \prod_{r=1}^{q} S(\pi_j, r)$ and then generalize Equation 10:

$$\mathrm{EU}(\pi) = \sum_{i=1}^{n} S(\pi_j) E(U(\pi_j)) \quad (11)$$

For the Open Loop model, the probability of successfully executing only the first i events of schedule π is now given by [3]

$$X(\pi_i) = \left(1 - \left(\prod_{r=1}^{q} S(\pi_{i+1}, r)\right)\right)\left(\prod_{r=1}^{q}\prod_{j=1}^{i} S(\pi_j, r)\right) \quad (12)$$

.

The expected value equation is the same as Equation 7.

The Decision Problem and Numerical Error

Numerical integration is prevalent in the handling of these problems. In the worst of all worlds, we need to be concerned about how this error propagates when deciding what the expectation bound to be used in the decision problems is. Under these circumstances, we formulate the decision problem by requiring ϵ the allowed numerical error tolerance to be part of the problem description. We then return "Yes" if we find a schedule for which the lower bound on the expected value is $\geq B$.

We can perform an analysis on equations 10 and 7 to find out what the error bounds on the expected value do as a function of the number of events and the error tolerance of the numerical integration step. We demonstrate how to calculate the lower bound of Equation 10, the expected value of a schedule under the Closed Loop model. This equation is the sum and product of quantities all > 0; the lower bound is calculated by plugging the lower bound of each constituent quantity back into the equation.

The lower bound on $S(\pi_j, r)$ is calculated as follows. Let $p = |\mathcal{P}(\pi_j)|$. Let $\mathcal{D}(\pi_j) = R(\pi_j, r) \cup S(\pi_p, r)$. $\mathcal{D}(\pi_j)$ has $p+1$ elements. By expanding Equation 3 the lower bound on $S(\pi_j, r)$ is given by:

$$\begin{aligned} S(\pi_j)_{lb} = {} & (-\epsilon)^{p+1} \\ & +(-\epsilon)^{p}(D_1 + D_2 ... + D_{p+1}) \\ & +(-\epsilon)^{p-1}((D_1 D_2 + D_1 D_3 ... + D_p D_{p+1}) \\ & \quad ... \\ & -\epsilon(\prod_{j\neq 1} D_j + \prod_{j\neq 2} D_j ... \prod_{j\neq n} D_j) \\ & +\prod_{j} D_j \end{aligned} \quad (13)$$

Each sum in this equation is a sum of all possible products of k values from $\mathcal{D}$ [4]. $E(U(\pi_j))_{lb} = E(U(\pi_j)) - \epsilon$, so we can now expand Equation 10:

$$E(\pi)_{lb} = \sum_{i=1}^{n} S(\pi_j)_{lb} (E(U(\pi_i)) - \epsilon) \quad (14)$$

Thus, if we receive a "Yes" answer to the decision problem given B and ϵ we know that $E(\pi)_{lb} \geq B$.

Complexity Results

As we described in the previous section, the decision problem for either UCRSP or URRSP is posed given a set of events x_i, their corresponding $C_{x,r}(z)$, $U_x(w)$, a resource r with corresponding r^{max} and $I_r(z)$, a set of metric temporal constraints $\tau_i(x, y)$, a bound B and a numerical error limit ϵ. The problem is to find a permutation π such that $E(\pi)_{lb} \geq B$ or report that no such permutation exists. We now show that these problems are $\mathcal{NP}$-complete.

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

Proof 1 *Suppose that the UCRSP has no temporal constraints. 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, whether in the Closed-Loop or Open-Loop models. The multiplications and sums in the formula presented above are all polynomial time operations. (This includes Equation 14; the expectation bounds can be maintained in a constant number of operations for each event in the permutation.) All that remains is showing that the convolution operation is a polynomial time operation. In the worst case, we can do each convolution using numerical integration, which takes constant time for a fixed error (HH64). We can add temporal constraints back to the UCRSP and preserve $\mathcal{NP}$-completeness.*

[3] This formula was incorrect in (FD03); we thank Neil Yorke-Smith for identifying the error.

[4] The analogy to the binomial expansion of $(x-\epsilon)^n$ should be readily apparent.

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 (DMP91).

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

Proof 2 *We will reduce the Knapsack problem to UCRSP. A Knapsack item $j = (s, u)$ where s is the size and u is the utility. Thus, we map j to a UCRSP event j with $C_{r,j}(z) = \delta(s)$ and $U_j(w) = \delta(u)$ where δ is Dirac's delta function. 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. There are no temporal constraints in the resulting UCRSP. This mapping requires only linear time. Now consider a schedule π that satisfies the expected utility bound of the UCRSP. Any schedule can be mapped into a partition of jobs by the following linear time procedure: while there is still any resource available, add π_j to the Knapsack. If adding π_j violates the resource constraint, π_k for $k \geq j$ are not in the Knapsack. Thus, the set of Knapsack items obeys 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. It is clear from the simplicity of this mapping that the (expected) utility of the schedule is the value of a solution to the Knapsack. Thus, 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.*

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

Proof 3 *Suppose we are given a permutation π. Only linear work is required to identify the $S_i(\pi)$. From Theorem 1, only polynomial work is required to compute all of the $E(S_i(\pi))$, and only linear work is required to compute $E(\pi)$ from $E(S_i(\pi))$. Thus, the total work to compute the expectation $E(\pi)$ is polynomial, and thus URRSP is in $\mathcal{NP}$.*

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

Proof 4 *It is trivial to see that UCRSP can be reduced to URRSP in polynomial time, since URRSP is a generalization of UCRSP. Further, any solution to the resulting URRSP is trivially a solution to the original UCRSP. Thus, URRSP is $\mathcal{NP}$-Hard.*

Modifications

In this section we explore the impact of some more of the assumptions we have made above.

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.

The third relaxation is the limitation on the resource modification probabilities. As we have said previously, we have discounted the possibility that a job could either produce or consume resources. Relaxing this assumption requires revising the expectation calculation again. However, the crux of the argument still holds. We can produce n independent variables whose expectations we can sum; in this case, one for each job. Each of these still requires only polynomial work to build, because we still only need to perform numerical integrals like those described in Equations 3 and 8.

We now 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. Thus, when the ending event of an activity is executed, no numerical integration is required to update the resource availability distribution, but it may be necessary to perform a numerical integration step to determine the success probability.

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 is now 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.

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(Mus02) [5].

Practice

We only experimented on UCRSPs. (URRSPs were too expensive to import and our budget for this project was limited.) However we experiments with both closed and open loop models.

We devised four heuristics to choose among unscheduled events: maximize the expected partial schedule utility (E), minimize the expected resource consumption of the job (R), minimize the probability of job failure given the current partial schedule (S), and maximize the expected value of the job (V). One problem with using the E heuristic is that it takes approximately 15 times as long to find a schedule as the others, due to the complexity of the Monte Carlo estimate of the value of the whole schedule at each step. One approximation 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. We refer to this approximation to E as $E*$

[5]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

To test the performance of the heuristics 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 resource consumption means 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.

Open Loop Model

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 7. 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.

As the left-hand columns of Table 1 show, the E heuristic (choose the job that maximizes the expected utility of the schedule built so far) and its approximation $E*$ considerably outperform the others on all these problems, with the R heuristic (minimize the resource consumption of the job) performing next-best. One situation where we might expect E to perform less well is when the resource consumption of a job and its utility are positively correlated, since E gets relatively little information in this case. We performed additional experiments on such problems, but found that it still outperforms the other heuristics. Results are shown on the right-hand side of Table 1 for the low-variance ($0.1 - -0.2$) problems only. We also examined the case where job resource consumption and utility are anti-correlated, leading to very problems, and found as expected that most of the heuristics find high-quality schedules in this case. In fact, E, $E*$, R, and V all tend to produce very similar schedules for these problems, and the solutions found are usually very close to optimal (on small problems we have computed the optimal for).

One interesting point is the performance of the approximation $E*$. This is generally very close to E, but requires less than five percent of the computation time. Since $E*$ approximates E by ignoring the possibility that previous actions have already failed, this suggests

Table 1: (Left) Performance of the heuristics on "uncorrelated" problems with 20 jobs, 10 constraints, and a resource limit of 60. (Right) Performance of the heuristics on problems with correlated and anti-correlated resource usage and utility.

Job Variance	Heuristic	Mean	Variance	Problem Type	Heuristic	Mean	Variance
0.1–1.0	E	19.42	29.63	Uncorr.	E	19.25	29.62
	$E*$	19.40	29.72		$E*$	19.25	29.56
	R	18.52	33.97		R	18.50	34.10
	S	17.58	30.36		S	16.04	26.95
	V	15.38	29.39		V	15.36	31.13
0.1–0.2	E	19.25	29.62	Corr.	E	10.08	0.07
	$E*$	19.25	29.56		$E*$	10.08	0.07
	R	18.50	34.10		R	7.21	0.57
	S	16.04	26.95		S	8.02	0.78
	V	15.36	31.13		V	9.36	0.33
0.8–1.0	E	19.60	31.19	Anticor.	E	27.73	50.06
	$E*$	19.62	30.98		$E*$	27.73	50.03
	R	18.53	33.67		R	27.73	50.11
	S	18.31	34.14		S	23.64	51.17
	V	15.40	28.13		V	27.74	50.04

that these failure probabilities are relatively unimportant in determining the behaviour of a schedule on these problems—most schedules can be divided into jobs that almost certainly succeed, and jobs that almost certainly fail, and the set of jobs that fall into neither of these categories is very small, so computing their probabilities exactly has little effect on the schedule.

Heuristics for Closed Loop Models

We replicated the results using the closed loop model for UCRSP, computing the expected value of schedules as shown in Equation 10. For the different variance ranges of the problems we found the same patters of performance for all of them, so we present only the low-variance results in Table 2. The interesting thing to note here is that the V heuristic that performed very badly on the open-loop problems performs as well or better than E on closed-loop. V chooses jobs in order of their value, so for open-loop execution it frequently selects jobs with high resource usage very early in the schedule, so subsequent jobs, even if they are very cheap in terms of resources, never get executed. With closed-loop execution these jobs still get executed (as long as their constraints are satisfied) even after previous jobs have failed, so simply building a schedule with the jobs in order of their value is a pretty good heuristic in this case.

Table 2: (Performance of the heuristics on "uncorrelated," correlated and anti-correlated closed-loop problems with 20 jobs, 10 constraints, and a resource limit of 60.

Problem Type	Heuristic	Mean	Variance
Uncorrel.	E	19.42	29.35
	$E*$	19.41	29.34
	R	18.57	34.17
	S	16.09	27.02
	V	19.38	27.78
Corr.	E	10.15	0.08
	$E*$	10.15	0.08
	R	7.30	0.64
	S	8.09	0.80
	V	10.29	0.12
Anticor.	E	27.89	49.97
	$E*$	27.88	49.94
	R	27.89	49.96
	S	23.81	50.98
	V	27.89	49.95

Future work

Current assumptions make it hard to model spacecraft downlink problem where the utility depends on getting the data back to earth, not just getting it into the on-board memory. This model interacts with the assumption that failed events generate no utility, and also that the utilities of the jobs are independent.

Ther are also much richer constraint models to consider. At present we only have precedence constraints, but we could also consider interval constrants on when jobs must be done for example. More interestingly, the definition of τ currently ties precedence and conditional event execution. these two ideas can be separated, so that x and y can occur in either order, but failure of one implies failure of another. This leaves unresolved what happens to the utility if x occurs before y, x succeeds and y fails. Utility model can be fixed, and the expectation formulas fixed with little difficulty.

References

J. Boyan and M. Littman. Exact solutions to time-dependent MDPs. In *NIPS*, pages 1026–1032, 2000.

M. Drummond, J. Bresina, and K. Swanson. Just-In-Case scheduling. *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence*, pages 1098–1104, 1994.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–94, 1991.

J. Frank and R. Dearden. Scheduling in the face of uncertain resource consumption and utility (poster). *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*, 2003.

J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. J. Wiley, 1964.

L. Khatib, J. Frank, D. Smith, R. Morris, and J. Dungan. Interleaved observation execution and rescheduling on earth observing systems. In *To Appear, Proceedings of the ICAPS Workshop on Plan Execution*, 2003.

N. Meuleau, D. Smith, and R. Washington. Optimal limited contingency planning. In *Proceedings of Uncertainty in Artificial Intelligence*, 2003.

N. Muscettola. Computing the envelope for stepwise-constant resource allocations. *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*, 2002.

J. Schneider, J. Boyan, and A. Moore. Value function based production scheduling. In Jude Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 522–530. Morgan Kaufmann, San Francisco, CA, 1998.