Scheduling in the Face of Uncertain Resource Consumption And Utility

Jeremy Frank and Richard Dearden
NASA Ames Research Center
Mail Stop N269-3
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 show that the problems are $\mathcal{NP}$-complete, and present some results that characterize the behavior of some simple heuristics over a variety of problem classes.

Introduction

In this paper we discuss scheduling problems in which the resource consumption and the utility of the tasks to be scheduled are given only as probability distributions. 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. We can then find a schedule that maximizes the expected utility, or find a schedule whose expected utility exceeds a lower bound.

The treatment of events during execution can lead to different formalizations of the expected utility of a schedule. In one scenario, the execution system can decide whether to reject an event based on pre-dispatch knowledge of resource requirements. Thus, event failure does not result in any resource modification. This is appropriate in data acquisition scenarios; the execution system may determine that poor quality image data will result, and reject an image acquisition task without storing any data. In the other scenario, the event is dispatched, the resource is impacted, and then the execution system is informed of success or failure. This model is appropriate in power system modeling, where unpredictable spikes in power requirement or generation can cause faults. **Try and figure out language to describe "failures" like downlink operations where the data buffer is "over-emptied" or re-charge operations where a battery is "over charged" but where no utility accrues to these tasks and where execution of the schedule can continue.**

Traditionally, constraint reasoning approaches have been applied to scheduling problems with known resource consumption and temporal constraints; this has led to "global" resource constraints such as those described in (Mus02; BC02). These techniques must be extended to handle problems with uncertain resource consumption and utility, where the goal is to find schedules that exceed a utility bound. While these problems are similar in spirit to bin-packing problems, numerical integration is required to convolve arbitrary probability distributions over resource availability. This introduces challenges in the application of constraint reasoning approaches to solve the problems.

elaborate: difference between ucrsp and urrsp
When computing the expected utility of a schedule, we must compute the probability that tasks fail because they attempt to consume or produce too much resource. Examples of resources that may behave this way are batteries; draining too much energy or attempting to over-charge a battery can destroy it. Under this model, the failure of one job no longer leads to unconditional failure of the rest of the jobs in the schedule. For example, a resource replenishment event that fails can be followed by a sequence of successful resource consumption events.

The paper is organized as follows. In § we present theoretical results on the problem of scheduling tasks that have uncertain resource consumption, utility, or both. In § we first describe some likely heuristics, and discuss drawbacks to them. We then briefly discuss some empirical results.

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 $r \in R$; thus, 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. 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. Finally, let $T = \tau_i(x, y)$ be a set of binary metric temporal constraints over pairs of events x, y.

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. Our task is to build a schedule π in advance 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.

clean up a little Initially, we will assume that there is only one resource r with maximum capacity r^{max}. We will also assume without loss of generality that the probability that there is less than 0 resource initially available is 0. 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 non-zero 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.

Closed-Loop Model

Our first 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. 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.

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:

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

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 event j successfully executes, conditioned on the execution of the previous $j-1$ events:

$$S(\pi_j, r) = \int_0^{r^{max}} T_{\pi_j,r}(z) dz \quad (2)$$

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 is permitted, only if it is known that the resource bounds will not be exceeded, which happens with probability $S(\pi_j, r)$. In this case, we must chop $T_{\pi_j,r}(z)$ so that it is non-zero only between 0 and r^{max}, and normalize by dividing by $S(\pi_j, r)$. The rest of the time the resource distribution remains unchanged, which means we add this to the previous distribution of resource availability, $A_{\pi_{j-1},r}(z)$, after normalizing by $1 - S(\pi_j, r)$. We can now write the following recurrence for $A_{\pi_j,r}(z)$:

$$A_{\pi_j,r}(z) = \frac{T_{\pi_j,r}(z)}{S(\pi_j, r}|_0^{r^{max}} + \frac{A_{\pi_{j-1},r}(z)}{1 - S(\pi_j, r)} \quad (3)$$

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:

$$E(\pi) = \sum_{i=1}^{n} S(\pi_j, r) E(U(\pi_j)) \quad (4)$$

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

Open Loop Model

Our second 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 use and event success or failure. We call this the *Open Loop Model.*

We preserve the intuitive definitions of $T_{\pi_j,r}(z), A_{\pi_j,r}(z)$ and $S(\pi_j, r)$. In this case, however, we modify the recurrence of $A_{\pi_j,r}(z)$; this now must acknowledge that the resource is modified unconditionally by event execution. This means we need to accumulate the probability of resource exhaustion, which was not a problem in the Closed Loop case. 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 consumed, and so the recurrence for $A_{\pi_j,r}(z)$ is as follows[1]:

$$A_{\pi_j,r}(z) = T_{\pi_j,r}(z)|_0^{r^{max}} \quad (5)$$

$$A_{\pi_j,r}(0) = \int_{-\infty}^{0} T_{\pi_j,r}(z) dz \quad (6)$$

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

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

$$X(\pi_i) = (1 - S(\pi, r, i+1)) \prod_{j=1}^{i} S(\pi, r, j) \quad (7)$$

(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

$$E(\pi) = \sum_{i=1}^{n} X(\pi, i) \left(\sum_{j=1}^{i} E(U(\pi_j)) \right) \quad (8)$$

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

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 $S_i(\pi)$ be these subsequences. In linear time, we can identify all $S_i(\pi)$, and there are $m \leq n$ of them. Let $S_i(\pi)$ be a* consuming subsequence *if $\forall j \in S_i(\pi) C_{r,j}(z) > 0$ only when $z < 0$, and a* producing subsequence *if $\forall j \in S_i(\pi) C_{r,j}(z) > 0$ only when $z > 0$.*

We can now compute the expected value $E(\pi)$ for URRSP as follows. First, observe that each $S_i(\pi)$ defines a UCRSP problem. If $S_i(\pi)$ is a consuming subsequence, this is obvious. If $S_i(\pi)$ is a producing subsequence, we can treat productions as consumptions and invert the resource bounds. Equation 8 shows how to compute $E(S_i(\pi))$ But $E(\pi) = \sum_{i=1}^{m} E(S_i(\pi))$; this is because the contribution of each $S_i(\pi)$'s utility is solely dependent on the resource availability distribution when the subsequence begins. **check rationale of this and clean it up some.** Another way of thinking about it 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 $S_i(\pi)$; 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 $I(z)$, which we have shown that we can do above. From elementary probability, we know $E(\prod_i X_i) = \sum_i E(X_i)$ where all of the X_i are independent.

clean this up generalize to all problem classes In closing, we observe that scaling up to multiple resources does not increase the difficulty of the problem. Suppose there are q resources. For all cases except UCRSP in the Open Loop model, we define $S(\pi_j) = \prod_{r \in R} S(\pi_j, r)$ and then generalize Equation 4:

$$E(\pi) = \sum_{i=1}^{n} S(\pi_j) E(U(\pi_j)) \quad (9)$$

For UCRSP in the Open Loop model, the probability of successfully executing only the first i events of schedule π is now given by [2]

$$X(\pi_i) = \left(1 - \left(\prod_{r=1}^{q} S(\pi, r, i+1)\right)\right) \left(\prod_{r=1}^{q} \prod_{j=1}^{i} S(\pi, r, j)\right) \quad (10)$$

The expected value equation remains unchanged.

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 passing both B and ϵ the allowed numerical error tolerance. We then compute the lower bound on the expected value of the schedule using the error analysis, and returning "Yes" if we find a schedule for which the lower bound is $\geq B$.

We can perform an analysis on equations 4 and ?? 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. Suppose $\mathcal{S}$ is the set of all $S(\pi_j, r)$. For a problem under the Closed Loop model, we can use Equation 4 to get the following equation for the lower bound of the expected value:

$$\begin{aligned} E(\pi)_{lb} = (-\epsilon)^n \quad (11) \\ +(-\epsilon)^{n-1}(S(\pi_1, r) + S(\pi_2, r)) \ldots + S(\pi_n, r)) \\ +(-\epsilon)^{n-2}(S(\pi_1, r)S(\pi_2, r) \ldots + S(\pi_{n-1}, r)S(\pi_n, r)) \\ \ldots \\ -\epsilon(\prod_{j \neq 1} S(\pi_j, r) + \prod_{j \neq 2} S(\pi_j, r)) \ldots \prod_{j \neq n} S(\pi_j, r)) \\ + \prod_{j} S(\pi_j, r) \end{aligned}$$

(where it is understood that the sums in this equation are sums of all products of $kS(\pi_{n-1}, r)$values.) Thus, if we receive a "Yes" answer to the decision problem given B and ϵ we know that $E(\pi)_{lb} \geq B$. **I don't see the need in doing this error analysis for Equation ?? but I will if we have to.**

serious hand waving going on here A subtle but important argument involves some of the numerical integration steps, particularly those that produce probability distributions instead of point probabilities. Since these distributions are carried throughout the computation, we must be certain that the error doesn't propagate by means of repeated numerical integration. The

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

crux of the argument is that we can ensure that the error at *any* point in the process is constant. This may require adapting the error bound passed to the numerical integrator to ensure that the error on the final probabilities is bounded by the input error bound, but this can all still be done in constant time.

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 B. 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 11; 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 Monte Carlo Integration, which takes constant time for a fixed error (HH64). We can add temporal constraints back to the UCRSP and preserve $\mathcal{NP}$-completeness. 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 $C_{r,j}(s) = 1$ and $U_j(u) = 1$). 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. 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, pi_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. Note that there is an additional complication, which is determining the probability of any permutation

of the unordered events when computing the expected utility.

The third relaxation is the limitation on the probability of 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 required only polynomial work to build, because we still only need to perform numerical integrals like those described in Equations 2 and 3.

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 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.

Modeling

Uncertain this belongs here In some cases, we might want to model a scenario in which an event "fails" due to a resource violation but utility is still deirved from the job. An example of this is downlinking data from a spacecraft with an uncertain transmission rate. If there is sufficient time and bandwidth to transmit more data than necessary to empty the onboard data buffer, it should be possible to model this. Doing so requires modifying the update formula for URRSP under the Open Loop model but should pose no serious problems.

Problem	Resource	Ordering	Complexity	Reducti
UCRSP	Consumable	Total	$\mathcal{NP}$-complete	Knapsa
UCRSP	Consumable	Partial	$\mathcal{NP}$-Hard	Knapsa
URRSP	Replenishable	Total	$\mathcal{NP}$-complete	UCRSI
URRSP	Replenishable	Partial	$\mathcal{NP}$-Hard	UCRSI

Figure 1: Complexity of various problems of scheduling with uncertain resource consumption.

Summary

might want to fix this up to handle closed/open loop model 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 [3].

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. We have also omitted results on the different event failure models since these do not change the complexity.

Practice

Empirical Results

beef this section up. We only experimented on UCRSPs. (URRSPs were too expensive to import and our budget for this project was limited.)

We devised three heuristics to choose among unscheduled events: maximize the expected partial schedule utility (E), minimize the expected resource consumption of the job (R), and minimize the probability of job failure given the current partial schedule (S). 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,

[3] 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

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

Job Variance	Heuristic	Mean	Variance	Problem Type
0.1–1.0	E	20.35	26.20	Uncorr.
	R	18.52	33.92	
	S	17.53	32.45	
0.1–0.2	E	13.88	32.88	Corr.
	R	11.71	45.39	
	S	11.18	26.55	
0.8–1.0	E	13.91	37.71	AntiCor.
	R	11.72	44.91	
	S	11.63	43.91	

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.

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 4. 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) 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. We expected the E heuristic to perform poorly when most job's resource consumption and utility are positively correlated. We performed additional experiments on such problems, but it still outperforms R and S. When job resource consumption and utility are anti-correlated 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. 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 simply the sum of the usages for the jobs, which is itself a Gaussian. However, it overestimates the value of each schedule. The right-hand columns of Table 1 shows results on the same set of problems using this approximation, again only for the low variance case.

Heuristic	Mean	Variance
E	13.88	32.88
E'	16.07	26.75
R	11.71	45.39
E	13.91	37.71
E'	15.64	51.16
R	11.72	44.91

The approximation actually beats E for the uncorrelated problems by a statistically significant amount. Our intuition is jobs that use few resources gain more from the approximation than large jobs, so the approximation favours small jobs at the beginning of the schedule, which is good for cases such as this with tight resource bounds. 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 problems.

Heuristics for Different Failure Models

How do the existing heuristics perform on the two rejection models? Can we come up with a sensible alternative heuristic for the event failure model?

References

N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. *Proceedings of the 8th International Conference on the Principles and Practices of Constraint Programming*, 2002.

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

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

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.