Scheduling with Uncertain Release Dates

Christine Wei Wu[1], Kenneth N. Brown[1], and J. Christopher Beck[2]

[1] Cork Constraint Computation Center,
Computer Science, University College Cork, Cork, Ireland
{cww1, k.brown}@cs.ucc.ie
[2] Toronto Intelligent Decision Engineering Laboratory,
Department of Mechanical and Industrial Engineering,
University of Toronto, Canada.
jcb@mie.utoronto.ca

Abstract. The aim of the paper is to examine whether we can adapt Bent and van Hentenryck's combination methods to our problem, and to determine how effective and significant the methods are. In particular, we develop four ways of applying Bent and Van Hentenryck's consensus approach[3] to our problem. In addition, since we assume the uncertainty distribution is known, we propose a probabilistic sampling method to handle lead-time uncertainty. That is we use this knowledge to select samples, and associate with them weights corresponding to their probability. This method allows us to generate fewer samples but have a more accurate model of future scenarios.

1 Introduction

The question of how to schedule tasks efficiently is an important problem for many applications. There are many off-line optimization algorithms for generating good schedules to satisfy different optimization criteria. However, these algorithms almost always assume a fully specified problem that doesn't change. The real world is rarely that well behaved. Many problems have uncertain specifications, and are subject to change, but still an immediate solution is needed at the outset. In particular, when the company doing the scheduling is part of a bigger supply chain, one of the main causes of uncertainty is the late arrival of raw material or components from a supplier (known as lead-time uncertainty). If the supplies have not arrived, then the job cannot be released for execution. The question then becomes how to construct an initial schedule that is robust to this variability in the release dates.

A number of approaches have been proposed to handle uncertain scheduling problems. In particular, Bent and van Hentenryck [2–4] have studied on-line packet scheduling and vehicle routing, where packet values and delivery requests are uncertain at the start. They show that sampling from a black-box distribution, solving each sample and combining the results using one of three methods (i.e. expectation, consensus or regret), does improve decision making. In this paper, we consider a different problem - job shop scheduling, where the jobs are sequences of tasks with constraints between them, where the jobs are known but have uncertain release dates, and where we have knowledge of the uncertainty distribution. The aim of the paper is to examine whether we can adapt Bent and van Hentenryck's combination methods to our problem, and to determine how effective and significant the methods are. Since the uncertainty distribution is known, we propose a probabilistic sampling method to handle lead-time uncertainty. That is we use this knowledge to select samples, and associate with them weights corresponding to their probability. This method allows us to generate fewer samples but have a more accurate model of future scenarios. Also, we consider four ways of applying Bent and Van Hentenryck's consensus approach[3] to our problem.

The paper is organized as follows. Section 2 briefly describes Bent and Van Hentenryck's online stochastic algorithms and the problems (i.e. packet scheduling and multiple vehicle routing) they experimented on. Section 3 presents the uncertain release date job-scheduling problem, and highlights the differences from vehicle routing. Section 4 defines probabilistic sampling in general. Section 5 and 6 present various ways of adapting the combination methods to our problem. Section 7 presents experimental results on instances of a job-scheduling problem. Finally, section 8 concludes the paper.

2 Background

Bent and van Hentenryck [2–4] considered two online resource allocation problems (packet scheduling and vehicle routing), where new tasks arrive every time step, and an immediate decision must be made whether to accept or reject them. They assume that they can obtain samples of future arrival patterns, and they propose a number of methods for using these samples to compute the best immediate decision. *Expectation* considers each possible decision, computes the reward over all the samples, and chooses the decision with the highest reward. *Consensus* considers each sample in turn, computes the optimal schedule with respect to that sample, and then selects the decision that appears most often in the optimal schedules of the samples. *Regret* is similar to consensus, but as well as computing the optimal schedules also computes the loss of reward for each other possible decision, and then chooses the decision that has the lowest total loss. Regret relies on having an accurate estimate of the cost of completing a schedule, to avoid the overhead of re-optimizing. They show that although expectation may give the best results, it is infeasible since it requires too much time to make its decisions. Consensus is much faster, since it has significantly fewer optimizations to consider. Regret is a more robust method, approaching expectation when there is sufficient computation time, and approaching consensus when time is limited. For the vehicle routing problems, the optimization time is significant, so to try to reduce the number of optimizations, they maintain a set of possible plans. Each decision is then restricted to extending the possible plans - i.e. when a new task arrives, it must be slotted into an existing delivery schedule, rather than completely re-optimizing the schedule. Some of the delivery plans will become infeasible, and are replaced by new variants of the feasible plans. Similarly, they treat each vehicle in turn, and do not attempt to reallocate the customer requests between vehicles.

There has been some previous research reasoning directly about uncertainty in combinatorial problems like scheduling [1, 5–9], but this approach is not considered further here.

3 The problem

We consider the problem of scheduling jobs where some of the jobs have uncertain release dates. We have N jobs, each with K tasks. There are M machines, and each machine can only process one task at a time. The tasks of a job have to be processed in a given order, and each task must be processed on a specified machine for a known duration. Once a machine has started processing a task, it must complete it. We assume P of the jobs have an uncertain release date (arrival time) - i.e. there is uncertainty about the date on which the first task of the job can be started. We assume that the actual start date is characterized by a known probability distribution. The remaining $N - P$ jobs have a release date of 0 - i.e. they can be started immediately. The aim is to construct and execute an initial schedule for the $N - P$ jobs, so that online during execution, the schedule can be adapted to include the arriving P jobs, while minimizing *makespan* (i.e. the total time until the last job is completed). For this paper, as an initial test of our methods, we set $P = 1$, $M = N - 1$, and we assume each job must visit each machine. The uncertain job arrives no earlier than t_0, and no later than t_n. That is, we have a job-shop scheduling problem, with the addition of one extra job with an uncertain release date.

Note that there are some significant differences from the packet scheduling and vehicle routing problems considered by Bent and van Hentenryck. In their problems, each job consisted of a single task, whereas in the JSP each job has multiple tasks with precedence constraints between them. That means for the JSP that there are more constraints between the tasks and the resources, so each decision will have more consequences further down the line (i.e. a decision to delay the start of one job will have an impact on all resources). Secondly, their problems are focused on maximizing the reward obtained for tasks completed (e.g. number of customers served or packets scheduled) - at each time step, they have a probability distribution for what tasks will arrive, and they must choose immediately whether to accept or reject the tasks. In the JSP, we know the jobs that we will be required to process, and must process them all, but the aim is to minimize the finishing time of the last task. We also consider an initial lead time before time 0, which we may spend trying to optimize the schedules.

4 Probabilistic sampling

Bent and van Hentenryck assumed that the arrival process for their tasks could be modelled by a black-box simulator. They had no knowledge of the distribution inside the simulator, but could ask for samples to be generated, and they used those samples to make their decisions. In this work, we assume that we do have access to the probability distribution from some historical data analysis. We can then use that distribution directly to select samples, and we can give them a weight corresponding to their probability. This should allow us to generate fewer samples, and to have a more accurate picture of the consequences of our decisions. Further, if we have limited time, we can generate samples in the order of probability, and thus do anytime reasoning. For example, suppose the arrival time distribution of a job is $Prob(\text{arrive at }(5, 6, 7, 8, 9)) = (0.1, 0.2, 0.4, 0.2, 0.1)$. There are five possible arrival times, so we only need to generate 5 samples - one for each arrival time. We can then compute the best decision for each sample by whatever method we choose, but then we combine them by weighting the samples by their probability. Thus when combining the decisions, a decision that is made for the sample where the job arrives at time 7 is given twice the weight of the decision made for the sample with arrival time 8. In the sections that follow, we discuss how the expectation and consensus methods can be adapted to the JSP with uncertain release dates, for use with probabilistic sampling.

5 Probabilistic Expectation

The idea of expectation is to find the decision that produces the lowest expected makespan. By assumption, nothing is going to change in the problem until time t_0. Therefore, we can consider a decision to be a schedule of tasks which start before t_0. For every feasible schedule up to t_n of the released jobs, for each possible arrival time, fix all the activities that start before that time, and run the optimizer on all the other activities, including the new job. This gives us a makespan for each possible partial schedule for each arrival time. We then obtain the expected makespan for the partial schedule by taking the weighted average over the arrival times of the makespans, using the probabilities as weights.

```
for each feasible solution s up to tn
   for each possible arrival time a
      fix all tasks starting before a
      optimize schedule for remaining tasks to get s'
      obtain the makespan
      f(s) += makespan(s') * prob(a)
return s with minimum f(s)
```

The optimization is called $|S| \times |A|$ times, where $|S|$ is the number of feasible initial schedules and $|A|$ is the number of possible arrival times. It also requires us to generate the $|S|$ initial feasible schedules.

6 Probabilistic Consensus:

The aim of the consensus approach is to reduce the number of optimizations required, so rather than starting with all possible decisions, we start with all possible arrival times. We compute the optimal schedule with respect to each possible arrival time, then construct 'consensus' early decision from those optimums. There are a number of different ways of defining consensus and early decisions, and we examine four below.

6.1 *C-schedule*: Consensus feasible schedule to t_0

The simplest method is the dual of expectation. For each arrival time, compute the optimal schedule up to t_0. Weight each schedule by the probability of the arrival time that generated it, and select the schedule with the highest weighted occurrence. For problems with a long gap from 0 to t_0, it is likely that each schedule will be different, and so this may default to selecting the optimal schedule for the most probable arrival time. This algorithm requires $|A|$ optimizations.

```
for each arrived time a
   compute optimal schedule s wrt. a
   f(s) += prob(a)
return s with maximum f(s)
```

6.2 *C-resource*: Consensus independent resource schedules up to t_0

Since the algorithm in 6.1 may not result in any schedule appearing twice, we could consider treating each resource as independent. For each arrival time, we will thus compute the optimal schedule as before, but now we will maintain a weighted count of the different tasks that start on a resource at each time step. To get the consensus decision, we will then examine each resource independently, and get a series of consensus schedules for each resource. This may introduce further problems, though, since the resource-independent consensus decisions may not be consistent - therefore, at each time step when building the schedule, we have to select the decision with the highest weighted count that is consistent with previous decisions. This algorithm also requires $|A|$ optimizations.

```
for each possible arrival time a
   compute optimal schedule wrt a
   for each resource M
      for each time step t
          if act starts at t
             M[act][t] += prob(a)
for each time step t
   for each resource M
       find consistent act with highest value in M
       schedule act on M at t
```

6.3 *C-point*: Consensus independent-resource schedules by time step

Although the algorithm above is likely to make more consensual decisions, those decisions are getting steadily less informed as the time steps increase (since the schedules on which they were based may have had different task allocations). Therefore we developed an approach which breaks decisions down, and considers each resource and each time step independently. That is, the consensus decision for each resource independently is computed for time 0 and allocated, the optimal schedules are then recomputed from time 1 onwards to produce new consensus decisions for time 1, and so on. This algorithm requires up to $|t_0| * |A| + |A - 1| + ... + 1$ optimizations.

```
for each time step t before to
   for each possible arrival time a
      compute optimal schedule s from t wrt a
      for each resource M
         if act starts at t
            M[act] +=  prob(a)
   for each resource M
      find consistent act with highest value in M
      schedule act on M
compute and execute optimal completion when job arrives
```

6.4 *C-tuple*: Consensus schedules by time step

We can also consider an alternative to *C-point* (6.3) in which we do not treat the resources independently - that is, each decision is a tuple of allocations of tasks to machines at a single time-step (i.e. the size of a tuple is equal to the number of machines). This algorithm requires the same number of optimizations as *C-point*.

```
for each time step t before to
   for each possible arrival time a
       compute optimal schedule s from t onwards
       find highest weighted tuple of task allocations
       schedule the tuple of tasks
compute and execute optimal schedule when job arrives
```

6.5 Discussion

The effect of the different definitions of consensus is shown in Figure 1. We have three resources (M_0, M_1, and M_2), three jobs ready to start (A, B and C), and another job due to arrive at times 5, 6, 7 or 8, with probabilities 0.4, 0.3, 0.15 and 0.15 respectively. We have four possible arrival times, so we generate four possible samples, and compute the optimal schedule for each one. These optimal schedules (up to time 4) are shown in the figure. Each schedule is different, so *C-schedule* selects the one generated by the highest weighted sample. *C-resource* sums the weights for each new task over each sample independently by resource, and generates a schedule that is different from any of the others. *C-point* treats the resources independently, and chooses a task with the highest weight (on each resource) to execute at time step 1 (and re-optimize for subsequent time steps). Finally, *C-tuple* selects the tuple of task allocations at time step 1 with the highest weight (and will also re-optimize).

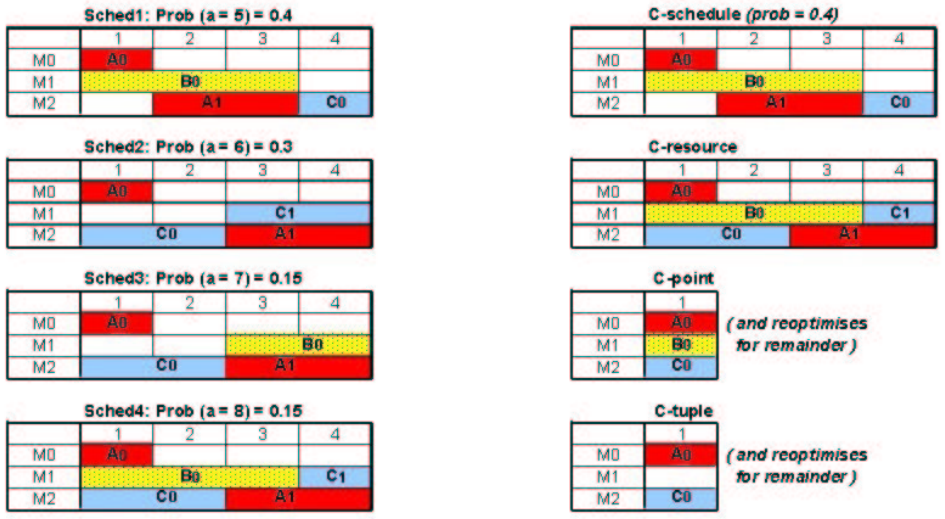
Sched1: Prob (a = 5) = 0.4
Sched2: Prob (a = 6) = 0.3
Sched3: Prob (a = 7) = 0.15
Sched4: Prob (a = 8) = 0.15
C-schedule (prob = 0.4)
C-resource
C-point
C-tuple
(and reoptimises for remainder)
M0
M1
M2
A0
A1
B0
C0
C1

Fig. 1. LHS is four optimal schedules for each sample (up to time 4). RHS is the decisions made by four kinds of consensus at time 1.

All four methods conduct off-line scheduling to compute the best partial schedule up to the arrival of the new job, and then react to the arrival on-line. The off-line parts of *C-schedule* and *C-resource* both return a single partial schedule, and require all their off-line computation to be carried out before the first task is executed. *C-point* and *C-tuple* produce their partial schedule iteratively, updating their decisions each time step, even though nothing has changed in the problem. They require more optimizations, but spread them over the time period from 0 to t_0.

7 Experimental Results

We have implemented *expectation*, *C-schedule*, *C-point* and *C-tuple* in ILOG Scheduler 6.0, a C++ library for constraint-based scheduling.

Since *C-point* and *C-tuple* update their decisions at each time step, after constructing an off-line schedule up to the earliest arrival t_0, they can also act as online methods to react to the new job's arrival. However, *expectation* and *C-schedule* are only off-line optimization methods and we choose *C-tuple* as the online reacting method for both. We also implemented two pure reaction methods (i.e. no lookahead) - *re-optimize* and *right-shift*. Both of the methods find a best schedule for the released jobs and execute the schedule. When a new job arrives at t, *re-optimize* optimizes the unscheduled tasks and the new job after t, while *right-shift* inserts the new job into the original schedule by shifting relevant tasks to the right.

We have tested the algorithms on JSP benchmark problems [10]. At first, we tried to use a 10 by 5 benchmark with the last job having an uncertain release date, but their results showed little difference with different approaches. Then, we adjusted the problems to 6 by 5 (with the last job having the uncertain lead-time) and scaled the durations of tasks, in order to make the problems more tightly constrained (i.e. more sensitive to the uncertainty).

Table 1 shows the results for one typical problem (la03) using two different distributions for the arrival time. *Absolute optimal* is an optimized solution knowing the release date of the new job (i.e. no uncertainty). The figures beside each method are the makespan of schedules generated by that method. The figures in bold indicate the schedules with same makespan as absolute optimal solutions. For example, in distribution 1 our four methods all produce very good schedules at the most probable (probability 0.6) arrival time 30, unlike those two pure reaction methods. Table 2 is the mean relative error (weighted by arrival probability) of each method with respect to absolute optimal.

Table 1. The experimental results on scheduling 6 jobs with one job having uncertain arrival time. It shows the makespan on all possible arrivals wrt. two probability distributions.

	Distribution 1			Distribution 2		
Arrival time	18	24	30	54	60	66
Probability	0.1	0.3	0.6	0.6	0.3	0.1
Abs Optimal	**93**	**96**	**100**	**121**	**127**	**133**
C-point	99	100	**100**	124	**127**	**133**
C-tuple	99	100	**100**	**121**	**127**	**133**
C-schedule + C-tuple	99	100	**100**	**121**	**127**	**133**
Expectation + C-tuple	94	**96**	**100**	**121**	**127**	**133**
Re-Opt	94	**96**	102	**121**	135	135
R-Shift	98	100	102	135	135	135

From Table 2, we can see that the four adapted methods all provide good results, not being more than 2% over the optimal. Because *expectation* must examine all possible initial schedules, it takes much longer time than the other methods - approximately 5 minutes as opposed to just over 1 second. However, when we look at the reactive methods, we see that the performance is not that

Table 2. This table shows the mean weighted relative errors (i.e. difference from absolute optimal) of using different methods in the two experiments in Table 1.

Method	Mean weighted relative error	
	Experiment 1	Experiment 2
C-point	1.02	1.01
C-tuple	1.02	1
C-schedule + C-tuple	1.02	1
Expectation + C-tuple	1.001	1
Re-Opt	1.02	1.02
R-Shift	1.03	1.09

significant, since it is possible to get very close to the optimal without reasoning about the uncertain events in advance. We believe that this is a feature of the problems we have looked at, in that the resource contention is generally not high, and so they are not sensitive to changes in the release date of a single job.

8 Conclusion and Future work

We considered various ways of adapting Bent and Van Hentenryck methods to the uncertain release date job-scheduling problem. We proposed the probabilistic sampling idea that allows us to use probability distribution to select samples, and give them a weight corresponding to their probability. Then we use the samples and their weights to make decisions. This gives us a more accurate picture of the consequences of decisions by generating fewer samples. Also, we devised probabilistic *expectation*, *C-schedule*, *C-point* and *C-tuple* methods. We experimented with these approaches on scaled benchmark problems, and showed the methods can solve uncertain release date job shop scheduling problems effectively. However, the benchmark problems appear to be too loosely constrained for us to draw any strong conclusions about the methods, since a reactive optimizer with no model of the future is also able to perform well on these problems.

For future work, we intend to investigate what properties of the problem make it sensitive to the uncertainty. Further, we intend to adapt the *regret*[2] method to the job-scheduling problem and also to consider breaking the expectation method down by time step as we have done for consensus.

9 Acknowledgments

This work is funded by Enterprise Ireland (SC/2003/81), and has received support from Science Foundation Ireland (00/PI.1/C075) and ILOG, S.A.

References

1. C. J. Beck, N. Wilson: Job Shop Scheduling with Probabilistic Durations. Proceedings of the Sixteenth European Conference on Artificial Intelligence (2004).
2. R. Bent, P. Van Hentenryck: Regret Only! Online Stochastic Optimization under Time Constraints. American Association for Artificial Intelligence (2004).
3. R. Bent, P. Van Hentenryck: The Value of Consensus in Online Stochastic Scheduling. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (2004).
4. R. Bent, P. Van Hentenryck: Online Stochastic and Robust Optimization. Ninth Asian Computing Science Conference Chiang Mai University (2004)
5. A. J. Davenport, C. Gefflot, J. C. Beck: Slack-based Techniques for Robust Schedules. Proceedings of the Sixth European Conference on Planning (2001).
6. Hélène Fargier, Jérŏme Lang, Thomas Schiex: Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. Proceedings American Association for Artificial Intelligence, pp175-180 (1996).
7. D. W. Fowler, K. N. Brown: Branching constraint satisfaction problems and Markov Decision Problems compared. Annals of Operations Research, Volume 118, Issue 1-4, pp85-100 (2003).
8. N. Yorke-Smith: Reliable Constraint Reasoning with Uncertain Data. PhD thesis, IC-Parc, Imperial College London (2004).
9. T. Walsh: Stochastic Constraint Programming. Proceedings of 15th European Conference on Artificial Intelligence (2002).
10. S. Lawrence: Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1984).