Scheduling with uncertain release dates

Student: Christine Wei Wu[1]
Supervisors: Kenneth N. Brown[1] and J. Christopher Beck[2]

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

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 [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 from a black-box, and they propose a number of methods (i.e. expectation, consensus and regret) for using these samples to compute the best immediate decision. There also has been some previous research reasoning directly about uncertainty in combinatorial problems like scheduling [1, 2, 5, 6, 8, 7], but this approach is not considered further here.

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 generate the samples deterministically, and associate weights with them, allowing us to generate fewer samples but have a more accurate model of future scenarios. We also develop four ways of adapting Bent and Van Hentenryck's methods to our problem.

2 The problem

We consider job-shop scheduling problems, but where P of the jobs have an uncertain release date - 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. As an initial test of our methods, we set $P = 1$. The uncertain job arrives no earlier than t_0, and no later than t_n.

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. For the JSP there are more constraints between the tasks and the resources, so each decision will have more consequences (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 all the jobs, and must process them all, but the aim is to minimize the finishing time of the last task.

3 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, so we can select samples, and 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.

4 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 a partial schedule by taking the weighted average over the makespans, using the probabilities as weights. 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.

5 Probabilistic Consensus

The aim of the consensus approach is to reduce the number of optimizations required. Rather than starting with all possible decisions, we start with all possible arrival times. For each one, we compute an optimal schedule. Then, we construct 'consensus' early decision from those optimums. We develop below three ways of defining consensus and early decisions for the JSP.

5.1 *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.

5.2 *C-point*: Consensus resource schedules by time step

Since the method above may not result in any schedule appearing twice, we could break the decisions down, and consider 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.

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

We can also consider an alternative to *C-point* 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. This algorithm requires the same number of optimizations as *C-point*.

5.4 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-point* treats the resources independently, and chooses a task with the highest weight (on each resource) to execute at time step 1 (and re-optimizes 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).

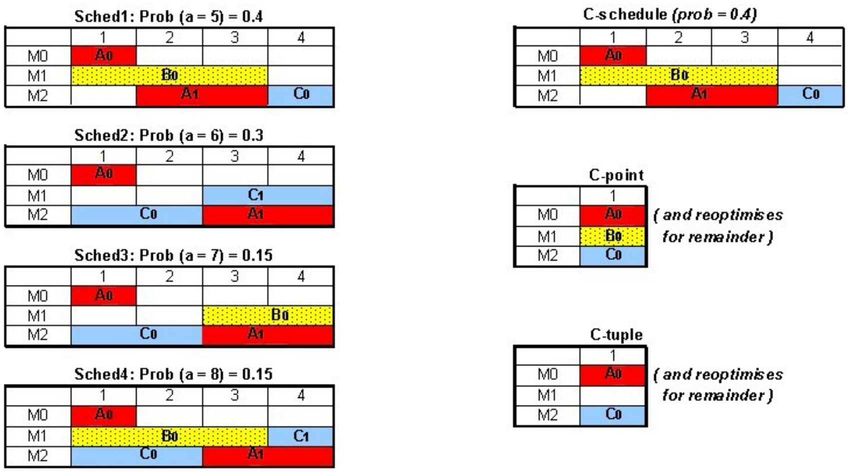

Fig. 1. LHS is four optimal schedules for each sample (up to time 4). RHS is the decisions made by three kinds of consensus at time 1.

All three methods conduct off-line scheduling to compute the best partial schedule up to the arrival of the new job, and then react online. The off-line part of *C-schedule* returns a single partial schedule, and requires all its 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.

6 Experimental Results

We have implemented *expectation*, *C-schedule*, *C-point* and *C-tuple* in ILOG Scheduler 6.0. 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 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 scaled benchmark problems [9]. Table 1 shows the results for one typical problem (la03) using two different distributions for the arrival time. It shows the weighted mean relative error of each method with respect to the absolute optimal (i.e. an optimized solution knowing the release date of the new job).

Table 1. mean weighted relative errors of different methods on one problem.

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

From Table 1 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 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 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.

7 Conclusion and Future work

We considered various ways of adapting Bent and Van Hentenryck's methods to the uncertain release date job-scheduling problem. We proposed a probabilistic sampling idea that allows us to use the 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 4, *C-schedule*, *C-point* and *C-tuple* methods. We experimented with these approaches on scaled benchmark problems, and showed the methods can solve uncertain lead-time job shop scheduling efficiently, but that the problems were too easy.

For future work, we intend to investigate what properties of the problem makes it sensitive to the uncertainty. Further, we intend to adapt the *regret*[3] method to the job-scheduling problem and also to consider breaking the expectation method down by time step as we have done for consensus.

8 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. A.J. Davenport, C. Gefflot, J. C. Beck: Slack-based Techniques for Robust Schedules. Proceedings of the Sixth European Conference on Planning (ECP-2001).
2. C. J. Beck, N. Wilson.: Job Shop Scheduling with Probabilistic Durations. Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI04).
3. R. Bent,P. Van Hentenryck.: Regret Only! Online Stochastic Optimization under Time Constraints. In American Association for Artificial Intelligence (AAAI-2004).
4. R. Bent,P. Van Hentenryck.: Online Stochastic and Robust Optimization. Ninth Asian Computing Science Conference (ASIAN'04) Chiang Mai University (December 2004)
5. Hélène Fargier, Jérǒme Lang, Thomas Schiex: Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. Proceedings AAAI'96: 175-180 (1996).
6. 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 (AOR-2003).
7. N. Yorke-Smith: Reliable Constraint Reasoning with Uncertain Data. PhD thesis, IC-Parc, Imperial College London, June 2004.
8. T. Walsh: Stochastic Constraint Programming. Proceedings of 15th European Conference on Artificial Intelligence (ECAI-2002).
9. 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).