Handling Uncertainty in Job-Shop Scheduling

Srinivas Nedunuri
Dept. of Computer Sciences
University of Texas at Austin
Austin, Texas, 78712
nedunuri@cs.utexas.edu

Douglas R. Smith
Kestrel Institute
3260 Hillview Ave.
Palo Alto, California, 94304
smith@kestrel.edu

William R. Cook
Dept. of Computer Sciences
University of Texas at Austin
Austin, Texas, 78712
cook@cs.utexas.edu

ABSTRACT

We discuss our proposed approach to handling uncertainty and constraints in job-shop scheduling. Specifically we deal with uncertainty in the duration of a task, as well as constraints on task completion time, task separation, etc. We discuss both offline and adaptive scheduling strategies. Our approach is to synthesize fast schedulers that handle uncertainty by calculating them from formal specifications.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Algorithms, Design

Keywords

scheduling, job-shop scheduling, program synthesis, design by calculation, uncertainty

1. BACKGROUND AND MOTIVATION

We are working on the synthesis of algorithms that deal with uncertainty in their environments (that is, uncontrollable actions in the environment). One area of interest is scheduling, specifically job-shop scheduling.

The input to the job-shop scheduling problem is a set of machines and a set of jobs. A job consists of a totally ordered series of tasks. Each task has to run on a specific machine from the given set for a given duration. There are a number of constraints that must be satisfied. For example, a machine can execute only one task at a time. All tasks run to completion (ie. there is no pre-emption) and the order of tasks in a job must be preserved. The goal is to correctly schedule all the tasks while minimizing the makespan, that is the time taken for the longest job to execute. Scheduling a task means to state at what time it is to start executing.

A common kind of uncertainty is the exact duration of a task may not be known until the task completes. The only information known ahead of time are lower and upper bounds on the duration. Related to uncertainty is the presence of constraints such as availability and delivery dates or minimum/maximum separation between job tasks. The difference between uncertainty and constraints is that uncertainty refers to unknown values of parameters over which we have no control. Both controllable and uncontrollable parameters can, however, be subject to constraints.

2. OUR POSITION

Much work has been done on handing constraints and uncertainty. However, there does not appear to be an overall unified approach to the problem. For example, the basic job shop problem was solved by Carlier and Pinson using disjunctive graphs [3]. Uncertainty in task duration was tackled by [1] using timed automata, and constraints by [2] using Petri nets. Polyhedral methods, based on a geometric interpretation of the constraints, have been pursued by some. With the exception of Carlier and Pinson's approach, It is also not clear how well these approaches scale to practical size problems.

We propose to tackle these problems in a unified way using program synthesis. Our approach to program syn-

thesis is to instantiate generic algorithm classes (such as Global Search or Divide and Conquer). The instantiation is carried out by program calculation, which can sometimes be automated.

In the rest of this paper, we discuss how we intend to formalize the problem of dealing with uncertainty.

3. OFFLINE SCHEDULING

The goal is to generate a "best" schedule offline which is resilient in the face of uncertainty of various parameters (within given bounds), such as task duration. The schedule may also have to satisfy additional constraints.

In [2], Altisen et.al. present an approach to generating schedulers from models expressed as Petri nets. They translate the Petri net to an equivalent Timed Automaton with Deadlines (TAD) in which the constraints become deadlines on transitions. From this TAD they generate a further property-specific TAD that expresses a requirement (such as "never enter an error state"). They illustrate their technique on a simple example of scheduling two singleton jobs. Let t_i and d_i be the start time and duration resp. of job i. The example requires that a correct schedule additionally satisfy the following conditions:

$5 \leq d_1 \leq 7$
$3 \leq d_2 \leq 4$
$t_1 + d_1 \leq 12$
$t_2 + d_2 \leq 25$
$t_2 + d_2 - (t_1 + d_1) \leq 10$
$t_2 \geq 14$

Unfortunately, the TAD their procedure generates appears to be incorrect, as it violates one of the problem constraints. It is also difficult to assess the efficiency of their generator since they do not provide generation times. However, we expect that the multiple changes of problem representation would lead to inefficiencies.

Our Approach

We first eliminate the uncertainty by use of quantifier elimination laws and then consider how to solve the resulting set of inequalities.

Quantifier Elimination Laws

For any anti-monotone predicate P over a preorder $\langle A, \leq \rangle$, $(\forall a : A.\, a \leq \widehat{a} \Rightarrow P(a)) = P(\widehat{a})$. Similarly, for any monotone predicate P over a preorder $\langle A, \leq \rangle$, $(\forall a : A.\, \check{a} \leq a \Rightarrow P(a)) = P(\check{a})$

Example: job scheduling

Returning to the example, we can write the above constraints from the example more formally as

$\exists t_1, t_2 \forall d_1, d_2 : 5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4$
$\Rightarrow$
$t_1 + d_1 \leq 12 \wedge t_2 + d_2 \leq 25 \wedge t_2 + d_2 - (t_1 + d_1) \leq 10$
$\wedge\, t_2 \geq 14$

We can calculate as follows:

$\exists t_1, t_2 \forall d_1, d_2 : 5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4$
$\Rightarrow$
$t_1 + d_1 \leq 12 \wedge t_2 + d_2 \leq 25 \wedge t_2 + d_2 - (t_1 + d_1) \leq 10$
$\wedge\, t_2 \geq 14$
$=$ { distribute $\Rightarrow$ over $\wedge$}
$\exists t_1, t_2 \forall d_1, d_2 :$
$5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4 \Rightarrow t_1 + d_1 \leq 12$
$\wedge ...$
$5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4 \Rightarrow t_2 \geq 14$
$=$ { distribute $\forall$ over $\wedge$}
$\exists t_1, t_2 :$
$(\forall d_1, d_2 : 5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4 \Rightarrow t_1 + d_1 \leq 12)$
$\wedge ...$
$(\forall d_1, d_2 : 5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4 \Rightarrow t_2 \geq 14)$
$=$ { quantifier elimination + algebra on 1st conjunct}
$\exists t_1, t_2 :$
$t_1 \leq 5$
$\wedge ...$
$(\forall d_1, d_2 : 5 \leq d_1 \leq 7 \wedge 3 \leq d_2 \leq 4 \Rightarrow t_2 \geq 14)$

By similar application of the law to the other conjuncts, we can reduce the constraints to the following equivalent set:

$$\exists t_1, t_2 : t_1 \leq 5 \wedge t_2 \leq 21 \wedge t_2 - t_1 \leq 11 \wedge t_2 \geq 14$$

Example: multimedia document

[2] illustrate their technique on another example, which is determining a correct schedule for elements of a multimedia document:

> A multimedia document is composed of the following basic activities and their corresponding duration constraints, noted as [minimal duration, maximal duration]: music [30,40], video [15,20], audio [20,30], text [5,10], applet [20,30], picture [20,∞]. In the beginning, music, video, audio, and applet are launched in parallel. The basic activities are submitted to the following synchronization constraints: (1) video and audio terminate as soon as any of the them ends; their termination is immediately followed by the text to be displayed; (2) music and text must terminate at the same time; (3) the applet is followed by a picture; (4) the document terminates as soon as both the picture and the music (and text) have terminated; and (5) the execution times of both the audio and applet depend on the machine load and are therefore uncontrollable

Let $D_m, D_t, D_p, D_v, d_a, d_l$ be the duration of music, text, picture, video, audio and applet, resp. Using our ap-

proach, we have the following relevant set of constraints:

$\exists D_m, D_t, D_p, D_v, \forall d_a, d_l :$
$20 \leq d_a, d_l \leq 30$
$\Rightarrow$
$D_m = \min(d_a, D_v) + D_t$
$\wedge 30 \leq D_m \leq 40$
$\wedge 5 \leq D_t \leq 10$
$\wedge 20 \leq D_p$
$\wedge 15 \leq D_v \leq 20$

Replacing $\min(d_a, D_v)$ with a variable m, adding constraints $m \leq d_a$and $m \leq D_v$ and applying the laws above, we find that $20 \leq m$, ie. $D_v \geq 20$. Combined with the constraint $D_v \leq 20$ from the problem statement we find that D_v must be exactly 20. This corresponds with the result obtained from synthesizing a property-specific TAD in [2].

Constructively Solving a System of Inequalities

At this point we have a simple set of inequalities. Fourier-Motzkin elimination provides a decision procedure for linear inequalities. Hodes [4] also describes a similar algorithm. However, the complexity of these procedures is quite high ($O(c^{2^n})$ for c constraints over n variables). We would like to find more efficient procedures.

Rehof and Mogensen

Rehof and Mogensen [5] describe an efficient linear-time algorithm for finding the minimal solution to a set of definite inequalities over a finite lattice $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$, that is inequalities of the form $t \sqsubseteq a$ where t is a term made up of monotone functions applied to constants, variables and other terms and a is an atom (either a constant or a variable).

Since we seek interval solutions, we want to apply the algorithm over an interval lattice. This is just a product lattice of $\langle [0..T], \leq, \min, \max \rangle$ and $\langle [0..T], \geq, \max, \min \rangle$, one for increasing time, the other for decreasing time. Since $[p, q] \sqsubseteq [p', q']$ is defined as $p \sqsubseteq p' \wedge q \sqsubseteq q'$ this is just $p \leq p' \wedge q' \leq q$. Thus we can transform the given inequalities into inequalities over an interval lattice by calculation. Note that although the 3rd inequality ($t_2 - t_1 \leq 11$) contains the subtraction operator we can rewrite it as $t_2 - 11 \leq t_1$ which is monotonic in its variable argument. Letting $[l_i, u_i]$ be the interval of t_i, we end up with

$[l_1, u_1] \sqsubseteq [0, 5]$
$\wedge [l_2, u_2] \sqsubseteq [0, 21]$
$\wedge [l_2, u_2] \sqsubseteq [l_2, u_1 + 11]$
$\wedge [l_1, u_1] \sqsubseteq [l_2 - 11, u_1]$
$\wedge [l_2, u_2] \sqsubseteq [14, T]$

Applying algorithm RM to the above gives us a least solution of $[l_1, u_1] = [3, 5], [l_2, u_2] = [14, 16]$.

Limitations of RM

Algorithm RM finds only point solutions (minimal/maximal). In the case of job-shop scheduling this is not necessarily an issue as it can be shown that the best schedule (minimal makespan) that can be determined offline is obtained by selecting the lower bound on the start time for each task. However, more generally, we know from applying an actual decision procedure such as Fourier-Motzkin or Hodes that a complete characterization of the solution space is $t_1 \in [3, 5], t_2 \in [14, 16], t_2 \leq t_1 + 11$. In practice we can find such solutions by picking a value for t_1 in its allowable range and then propagating as suggested in [7], but this involves extra work and still does not provide us with a mathematical characterization of the solution space. There are also limitations introduced by the requirement that the inequalities be definite, which will not always be easy to work around.

For these reasons, we are currently investigating solution procedures based on a class of algorithms called Global Search with Optimality (GSO), [6]. The basic idea is to define a very high level generic parametrized algorithm which carries out a generate and test strategy. It contains various operators which must be instantiated to the specific problem. Domain knowledge is then applied to refine the algorithm into a very efficient one.

4. ADAPTIVE SCHEDULING

Introduction to the Problem and Existing solutions

So far we have relied only on information that is known in advance. This way a schedule can be computed offline and no run-time information is necessary. However, when there is uncertainty in task duration, there may be a radically different optimum schedule depending on when the task completes, as the following example from [1] shows. Let J and K be two jobs, where

$$J = (m1, 10) \prec (m_3, 4) \prec (m_4, 5)$$

and

$$K = (m2, [4, 8]) \prec (m3, 7)$$

Let the subscript i denote the i-th task in a job. The first element of the task is the machine which executes the task and the second is the duration, which may be a range. In the example, if K_1 finishes in no more than 5 units then K_2 should be scheduled immediately, otherwise the execution of K_2 should be delayed until J_2 has completed, which should be scheduled immediately

after J_1. If we don't take into account this *dynamic* or run-time information then we can end up with suboptimal solutions.

[1] present a solution to this problem based on timed automata. Each possible schedule (excluding machine uniqueness and task separation constraints) is considered as a path through a non-deterministic timed automaton. Starting at the final state of the automaton, and using a dynamic programming approach, it is possible to compute a function that determines a least cost path through the automaton. The function is then used to determine a scheduling strategy which examines the current value of a timer to determine which path to take through the automaton. The authors don't mention how long it takes for their approach to discover schedules in their paper, but they do mention that the 4×6 problem is at the upper limit of what they can compute. Given that the approach relies on a very generic algorithm (shortest path through an automaton), and uses very little problem specific knowledge it is difficult to see how it would be efficient.

Our Approach

Just as [1] do, we plan to incorporate uncertainty by pre-computing alternate feasible schedules. That is, as the GSO algorithm generates schedules (recall the basic paradigm of global search is generate and test), certain schedules that would otherwise get filtered out will not get filtered out because of uncertainties in the duration of some of the tasks. Associated with each such schedule will be a linear constraint (of the form $tk.d \leq T$) on the duration of the task. Then at runtime, once the the uncertain task has completed, we evaluate the constraint and if it succeeds, we take that schedule.

This is illustrated with a simplified version of the example above: $J = (m1, 10) \prec (m_3, 4) \prec (m_4, 5)$ and $K = (m2, [4, 8]) \prec (m3, 7)$ which generates the two following potentially optimal schedules:

for K1.d=8

```
m1: <---J1--->
m2: <--K1-->
m3:           <J2><-K2-->
m4:               <J3->
```

which has length 21, and for K1.d=4

```
m1: <---J1--->
m2: <K1>
m3:     <-K2--><J2>
m4:                <J3->
```

which has length 20. In order that the second schedule be shorter than the first we have an implied constraint:

$$K_1.d + 7 + 4 + 5 \leq 21$$

ie $K_1.d \leq 5$

We perform this test after K_1 has completed. If the constraint is not satisfied we switch to the first schedule. This is easier to carry out in a job completion view where the order of the tasks does not change, only the spacing between them. In this view, the schedules appear as follows:

```
J: <---J1---><J2><J3->
K: <--K1-->      <-K2-->
```

and

```
J: <---J1---> <J2><J3->
K: <K1><-K2-->
```

The basic idea is to index the times associated with each schedule to amalgamate them as:

```
J: [(m1,(0,0)), (m3,(10,11)), (m4,(14,15))]
K: [(m2,(0,0)), (m3,(14,4))]
```

where the first (second) value in the pair gives the start time in the first (second) schedule.

The difference between our approach and that of [1] is that generation of each of the potential schedules is not based on a generic procedure (shortest path through timed automata) but will be efficiently synthesized via GSO.

5. REFERENCES

[1] Y. Abdeddaim, E. Asarin, and O. Maler. On optimal scheduling under uncertainty. *TACAS*, 2003.

[2] K. Altisen, G. Gossler, A. Pnueli, and et.al. A framework for scheduler synthesis. *IEEE Real-Time Systems Symposium*, 1999.

[3] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2), Feb 1989.

[4] L. Hodes. Solving problems by formula manipulation in logic and linear inequalities. *Artificial Intelligence*, (3):165–174, 1972.

[5] J. Rehof and T. Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, 35(2–3):191–221, 1999.

[6] D. Smith. Structure and design of global search algorithms. *Kestrel Institute Tech Rep., KES.U,87.12*, December 1988.

[7] S. Westfold and D. Smith. Synthesis of efficient constraint satisfaction programs. *The Knowledge Engineering Review*, March 2001.