Optimal Scheduling under Uncertainty[*]

Yasmina Abdeddaïm, Eugene Asarin and Oded Maler[†]

March 29, 2002

1 Introduction

The problem of evaluating or optimizing the performance of an open system, that is, a system that interacts with an uncontrollable external environment, raises some serious conceptual problems. Such a system will typically have many behaviors, and the question is how to take all of them into account while evaluating the system. There are two basic approaches which can roughly be characterized as "hard" and "soft":

- Worst-case analysis: the system is evaluated according to its worst behavior. In verification, where the performance measure is discrete and consists of a binary classification into "correct" and "incorrect", this means that a system is correct only if *all* its behaviors satisfy the property in question.
- Average-case analysis: the set of all environmental inputs is considered as a probability space and this induces a probability over all system behaviors. The system is then evaluated according to the expected value (over all its behaviors) of the performance measure.

Each of these approaches has its advantages and shortcomings. The first approach is often used in safety-critical systems where the cost associated with bad behaviors is too high to tolerate, even if they constitute a negligible fraction of the admissible behaviors. On the negative side, this approach might lead to an over-pessimistic allocation of resources which can be very inefficient during most of the system lifetime.[1] The probabilistic approach is more appropriate where the performance measure is more "continuous" in nature, e.g. the waiting time in a queue, and we can tolerate graceful degradation in moments of extreme environmental pressure. The computational cost, however, of probabilistic evaluation or optimization problems is usually much larger, except for simple cases when they admit analytical solutions.

In this work we propose an alternative framework for the problem of job-shop scheduling under temporal uncertainty. The system to be designed is a scheduler, i.e. a mechanism that controls the allocation of resources to competing tasks. The environment consists of tasks, all known in advance, that need to be executed in on certain machines while satisfying some ordering constraints. The only source of uncertainty is the duration of the tasks which is known to be bounded within an interval of the form $[l, u]$. Each *instance* of the environment consists of selecting a number $d \in [l, u]$ for every

[*]This work was partially supported by the European Community Project IST-2001-35304 AMETIST `http://ametist.cs.utwente.nl`

[†]VERIMAG, 2, av. de Vignate, 38610 Gières, France. {`abdeddai | asarin | maler`}`@imag.fr`

[1]A good analogy is to live all your life wearing a helmet fearing a meteorite rain.

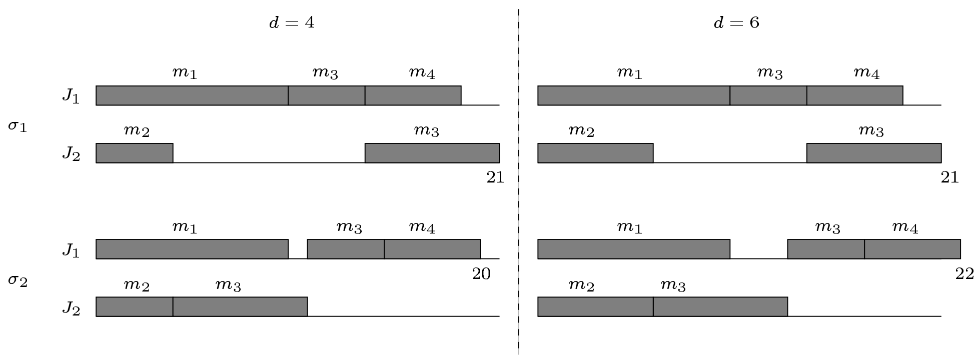

Figure 1: The schedules generated by σ_1 and σ_2 for environment instances $d = 4$ and $d = 6$.

task. The behavior induced by the scheduler on this instance is evaluated according to the length of the schedule, i.e. the termination time of the last task executed.

As a running example consider two jobs

$$J_1 = (m_1, 10) \prec (m_3, 4) \prec (m_4, 5) \qquad J_2 = (m_2, [4, 6]) \prec (m_3, 7)$$

with the intended meaning that J_1 has to use m_1 for 10 time, then m_3 for 4 time, etc. The environment is deterministic except for the uncertainty in the duration of the first task of J_2 which ranges between 4 and 6. In this example the only resource for which there is a conflict between the two jobs is m_3 and this is the only decision the scheduler needs to take. Let us consider first two static priority schedulers, σ_1 which gives priority to J_1 over J_2 on m_3 and σ_2 which prefers J_2. Figure 1 compares the performance of the two schedulers on two instances of the environment, one where the duration of m_2 is 4 and the other where the duration is 6. As one can see σ_1 is better when $d = 6$ while σ_2 is better when $d = 4$.

Applying a worst-case optimality criterion we would prefer σ_1 because it has better performance for the worst-case[2] where $d = 6$. According to this criterion there is no reason to prefer to a smarter dynamic scheduler that observes the termination time of task m_2 in J_2 and decides accordingly whether or not to sneak in task m_3 as well. Intuitively, however, we will prefer such an adaptive scheduler and the major contribution of this paper is the formalization this intuition and the development and implementation of an algorithm for finding adaptive schedulers that are optimal in this sense.

Consider the following approach: initially we assume that all future tasks will terminate as late as possible and find an optimal schedule for that instance. During the execution, whenever a good surprise occurs, i.e. a job terminates before expected, we reschedule the "residual" problem, assuming worst-case for the tasks that has not yet terminated. In our example, if task m_2 in J_2 terminated after 4 time we have:

$$J_1' = (m_1, 6, !) \prec (m_3, 4) \prec (m_4, 5) \qquad J_2' = (m_3, 7)$$

where the ! indicates that m_1 must be scheduled immediately (we assume no preemption). For this problem the optimal solution will behave like σ_2. Likewise if m_2 terminates at 6 we have

$$J_1' = (m_1, 4, !) \prec (m_3, 4) \prec (m_4, 5) \qquad J_2' = (m_3, 7)$$

[2]In fact, we are considering a special class of problems in which the performance is *monotone* with respect to the instance. That is if we have two instances such that $d \leq d'$ than the performance of any static priority scheduler will be better on d than on d' and, in addition, if d is given in advance the optimal schedule on d will be better than that of d'.

and the optimal schedule will behave like σ_1. The property of the schedules obtained this way is stronger than simple worst-case optimality: at any moment in the execution they are optimal with respect to the worst-case assumption concerning the future.[3]

This approach involves a lot of *online* computation, namely solving a new scheduling problem each time a task terminates prematurely. The alternative approach that we propose in this paper is based on expressing the scheduling problem using timed automata and synthesizing a controller *offline*. In this framework [AMPS98, AM99, AGP99] a strategy is a function from states and clock valuations to controller actions (in this case starting tasks). After computing such a strategy and representing it properly, the execution of the schedule may proceed while keeping track of the state of the corresponding automaton. Whenever a task terminates, the optimal action is quickly computed from the strategy look-up table and the results are identical to those obtained via online re-scheduling.

The rest of the paper is organized as follows. In Section 2 we describe the model and characterize the properties of the dynamic schedulers we want to compute. In section 3 we define timed automata and show how they can model scheduling situations. The algorithm for synthesizing optimal strategies is described in Section 4 along with its implementation using the zone library of Kronos. Section 5 presents experimental results demonstrating the improved performance of the computed strategy over static schedules. Finally we discuss further improvements to increase the size of the problems that can be treated.

2 The Model

We will use a formulation which is slightly more general than the standard job-shop problem by allowing a partial-order relation between tasks. We still abuse the name job-shop for a while.

Definition 1 (Uncertain Job-Shop Specification)
An uncertain job-shop specification is $\mathcal{J} = (P, M, \prec, \mu, D, U)$ where P is a finite number of tasks, M is a finite set of machines, $\prec$ is a partial-order precedence relation on tasks, $\mu : P \to M$ assigns tasks to machines, $D : P \to \mathbb{N} \times \mathbb{N}$ assigns an integer-bounded interval to each task and U is a subset of immediate tasks consisting of some $\prec$-minimal elements.

The set U is typically empty in the intial definition of the problem and we need it later to define residual problems. We use D^l and D^u to denote the projection of D on the lower- and upper-bounds of the interval, respectively. The set $\Pi(p) = \{p' : p' \prec p\}$ denotes all the predecessors of p, namely the tasks that need to terminate before p starts. In the standard job-shop scheduling problem, $\prec$ decomposes into a disjoint union of chains (linear orders) called jobs.

An *instance* of the environment is any function $d : P \to \mathbb{R}_+$, such that $d(p) \in D(p)$ for every $p \in P$. The set of instances admits a natural partial-order relation: $d \leq d'$ if $d(p) \leq d'(p)$ for every $p \in P$. Any environment instance induces naturally a deterministic instance of $\mathcal{J}$, denoted by $\mathcal{J}(d)$, which is a classical job-shop scheduling problem.

Definition 2 (Schedule) *Let $\mathcal{J} = (P, M, \prec, \mu, D, U)$ be an uncertain job-shop specification and let $\mathcal{J}(d)$ be a deterministic instance. A feasible schedule for $\mathcal{J}(d)$ is a function $s : P \to \mathbb{R}_+$, where $s(p)$ defines the start time of task p, satisfying:*

1. *Precedence: for every p, $s(p) \geq \max_{p' \in \Pi(p)} s(p') + d(p')$.*

[3]A similar idea is used in model-predictive control where at each time actions at the current "real" state are re-optimized while assuming some "nominal" prediction of the future.

2. *Mutual exclusion: for every p, p' such that $\mu(p) = \mu(p')$*

$$[s(p), s(p) + d(p)] \cap [s(p'), s(p') + d(p')] = \emptyset.$$

3. *Immediacy: For every $p \in U$, $s(p) = 0$.*

The length of the schedule is the termination time of the last task, i.e. $\max_{p \in P} s(p) + d(p)$. An *optimal schedule* for $\mathcal{J}(d)$ is a feasible schedule having a minimal length. If we care only about the worst-case, we let $\overline{d}$ be the maximal instance ($\overline{d}(p) = D^u(p)$ for every p), find an optimal schedule s for the deterministic problem $\mathcal{J}(\overline{d})$ and stick to s regardless of the actual instance. It can be easily shown that if $d' < d$ then every feasible schedule for d is also feasible for d'.

If we want to be adaptive we need a *scheduling strategy*, i.e. a rule that may induce a different schedule for every d. However, this definition is not simple because we need to restrict ourselves to *causal* strategies, strategies that can base their decisions only on information available at the time they are made. In our case, the value of $d(p)$ is revealed only when p terminates.

Definition 3 (State of Schedule) *A state of a schedule is $S = (P^f, P^a, c, P^e)$ such that P^f is a downward-closed subset of $(P, \prec)$ indicating the tasks that have terminated, P^a is a set of active tasks currently being executed, $c : P^a \to \mathbb{R}_+$ is a function such that $c(p)$ indicates the time elapsed since the activation of p and P^e is the set of enabled tasks consisting of those whose predecessors are in P^f. The set of all possible states is denoted by $\mathcal{S}$.*

Definition 4 (Scheduling Strategy) *A (state-based) scheduling strategy is a function $\sigma : \mathcal{S} \to P \cup \{\bot\}$ such that for every $S = (P^f, P^a, c, P^e)$, $\sigma(S) = p \in P^e \cup \{\bot\}$ and for every $p' \in P^a$, $\mu(p) \neq \mu(p')$.*

In other words the strategy decides at each state whether to do nothing and let time pass ($\bot$) or to choose an enabled task, not being in conflict with any active task, and start executing it. An operational definition of the interaction between a strategy and an instance will be given later using timed automata, but intuitively one can see that the evolution of the state of a schedule consists of two types of transitions: uncontrolled transitions where an active task p terminates after $d(p)$ time and moves from P^a to P^f, leading possibly to adding new tasks to P^e, and a decision of the scheduler to start an enabled task. The combination of a strategy and an instance yields a unique schedule $s(d, \sigma)$ and we say that a state is (d, σ)-reachable if it occurs in $s(d, \sigma)$.

Next we formalize the notion of a residual problem, namely a specification of what remains to be done in an intermediate state of the execution.

Definition 5 (Residual Problem) *Let $\mathcal{J} = (P, M, \prec, \mu, D, U)$ and let $S = (P^f, P^a, c, P^e)$ be a state. The residual problem starting from S is $\mathcal{J}_S = (P - P^f, M, \prec', \mu', D', P^a)$ where $\prec'$ and μ' are, respectively, the restrictions of $\prec$ and μ, to $P - P^f$ and D' is constructed from D by letting*

$$D'(p) = \begin{cases} D(p) \dot{-} c(p) & \text{if } p \in P^a \\ D(p) & \text{otherwise} \end{cases}$$

Let $\overline{d}$ be an instance. A strategy σ is *$\overline{d}$-future-optimal* if for every instance d and from every (σ, d)-reachable state S, it produces the optimal schedule for $\mathcal{J}_S(\overline{d})$. If we take $\overline{d}$ to be the maximal instance, this is exactly the property of the online re-scheduling approach described informally in the previous section.

3 Timed Automata for Scheduling Problems

Timed automata are automata augmented with continuous clock variables whose values grow uniformly at every state. Clocks can be reset to zero at certain transitions and tests on their values can be used in conditions for enabling transitions. Hence they are ideal for describing concurrent time-dependent behaviors.

Definition 6 (Timed Automaton) *A* timed automaton *is a tuple $\mathcal{A} = (Q, C, s, f, I, \Delta)$ where Q is a finite set of states, C is a finite set of clocks, I is the staying condition (invariant), assigning to every $q \in Q$ a conjunction I_q of inequalities of the form $c \leq u$, for some clock c and integer u, and Δ is a transition relation consisting of elements of the form (q, ϕ, ρ, q') where q and q' are states, $\rho \subseteq C$ and ϕ (the transition guard) is a conjunction of formulae of the form $(c \geq l)$ for some clock c and integer l. States s and f are the initial and final states, respectively.*

A *clock valuation* is a function $\mathbf{v} : C \to \mathbb{R}_+ \cup \{0\}$, or equivalently a $|C|$-dimensional vector over $\mathbb{R}_+$. We denote the set of all clock valuations by V. A configuration of the automaton is hence a pair $(q, \mathbf{v}) \in Q \times V$ consisting of a discrete state (sometimes called "location") and a clock valuation. Every subset $\rho \subseteq C$ induces a reset function $\mathrm{Reset}_\rho : V \to V$ defined for every clock valuation $\mathbf{v}$ and every clock variable $c \in C$ as

$$\mathrm{Reset}_\rho\, \mathbf{v}(c) = \begin{cases} 0 & \text{if} \quad c \in \rho \\ \mathbf{v}(c) & \text{if} \quad c \notin \rho \end{cases}$$

That is, Reset_ρ resets to zero all the clocks in ρ and leaves the other clocks unchanged. We use **1** to denote the unit vector $(1, \ldots, 1)$ and **0** for the zero vector.

A *step* of the automaton is one of the following:

- A discrete step: $(q, \mathbf{v}) \xrightarrow{0} (q', \mathbf{v}')$, where there exists $\delta = (q, \phi, \rho, q') \in \Delta$, such that $\mathbf{v}$ satisfies ϕ and $\mathbf{v}' = \mathrm{Reset}_\rho(\mathbf{v})$.
- A time step: $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1})$, $t \in \mathbb{R}_+$ such that $\mathbf{v} + t\mathbf{1}$ satisfies I_q.

A *run* of the automaton starting from a configuration $(q_0, \mathbf{v}_0)$ is a finite sequence of steps

$$\xi : \quad (q_0, \mathbf{v}_0) \xrightarrow{t_1} (q_1, \mathbf{v}_1) \xrightarrow{t_2} \cdots \xrightarrow{t_n} (q_n, \mathbf{v}_n).$$

The *logical length* of such a run is n and its *metric length* is $t_1 + t_2 + \cdots + t_n$. Note that discrete transitions take no time.

Next we construct for every task p with $d(p) = [l, u]$ a 3-state timed automaton $\mathcal{A}_D$ (Figure 2-(a)) with a waiting state $\overline{p}$, an active state p where the task is executing and a final state $\underline{p}$. The automaton has one clock which is reset to zero upon entering p ("start") and its value determines when a transition to $\underline{p}$ ("end") is taken. This automaton captures all instances: it can stay in p as long as $c \leq u$ and can leave p as soon as as $c \geq l$. This automaton represents the possible behaviors of the task *in isolation*, i.e. ignoring precedence and resource constraints. The transition from $\overline{p}$ to p is triggered by a decision of the scheduler, respecting those constraints, while the time of the transition from p to $\underline{p}$ depends on the instance. For a given instance d we have the automaton $\mathcal{A}_d$ of Figure 2-(b) where this transition happens after exactly d time. The automaton $\mathcal{A}_{D,d}$ of Figure 2-(c) will be used later for computing d-future optimal strategies: it can terminate as soon as $c \geq d$ but can stay in p until $c = u$.

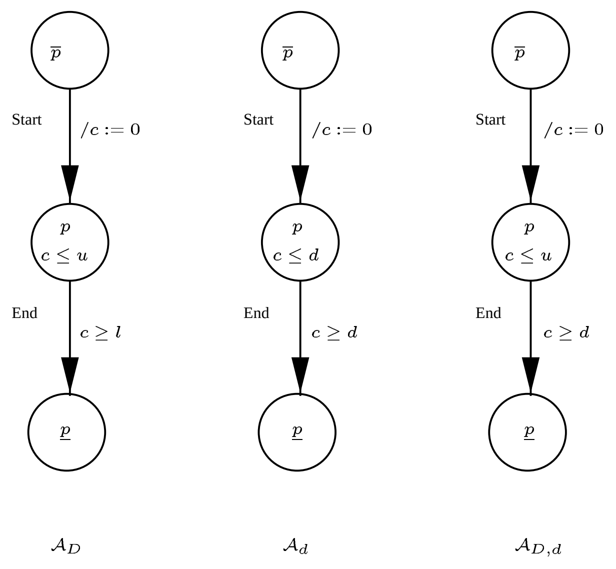

Figure 2: (a) The generic automaton $\mathcal{A}_D$ for a task p such that $D(p) = [l, u]$. (b) The automaton $\mathcal{A}_d$ for a deterministic instance d. (c) The automaton $\mathcal{A}_{D,d}$ for computing d-future optimal strategies. Staying conditions for $\overline{p}$ and $\underline{p}$ are **true** and omitted from the figure.

The timed automaton for the whole job-shop specification is the composition of the automata for the individual tasks.[4] The composition is rather standard, the only particular feature is the enforcement of precedence and mutual exclusion constraints. This is achieved by forbidding global states in which a task is active before all its predecessor have terminated or in which two or more tasks that use the same resource are active.

Let $Q^i = \{\overline{p}_i, p_i, \underline{p}_i\}$ be the state-space of the automaton for task p_i. We say that a global state $q = (q^1, \ldots, q^n) \in Q^1 \times \ldots Q^n$ is *valid* if:

1. Precedence: For every i, if $q^i \neq \overline{p}$ then $q^j = \underline{p}$ for every $p_j \in \Pi(p_i)$.
2. Mutual exclusion: For every i, if $q^i = p$ then for every p_j such that $\mu(p_i) = \mu(p_j)$, $q^j \neq p$.

Definition 7 (Composition) *Let $\mathcal{J} = (P, M, \prec, \mu, D, U)$ be a job-shop specification and let $\mathcal{A}^i = (Q^i, C^i, \Delta^i, I^i, s^i, f^i)$ be the automaton corresponding to each task p_i. The composition of these automata is $\mathcal{A} = (Q, C, \Delta, I, s, f)$ such that Q is the restriction of $Q^1 \times \ldots Q^n$ to valid states, $C = C^1 \cup \ldots \cup C^n$, $s = (s^1, \ldots, s^n)$, $f = (f^1, \ldots, f^n)$, $I_q = \bigwedge_i I_{q^i}$ and the transition relation Δ contains all the tuples of the form*

$$((q^1, \ldots, q^i, \ldots, q^n), \phi, \rho, (q^1, \ldots, p^i, \ldots, q^n))$$

such that $(q^i, \phi, \rho, p^i) \in \Delta^i$ for some i and both $(q^1, \ldots, q^i, \ldots, q^n)$ and $(q^1, \ldots, p^i, \ldots, q^n)$ are valid.

The result of applying this composition to the automata corresponding to the example appears in Figure 3. Since in this example $\prec$ decomposes into two disjoint chains, we can annotate global

[4]In the following we will not distinguish between $\mathcal{A}_D$, $\mathcal{A}_d$ and $\mathcal{A}_{D,d}$ — the definitions are the same for all of them.

discrete states with tuples of the form (α^1, α^2) where α^j is either $\overline{m}$ or m where $m = \mu(p)$ and p is the maximal enabled or active task in the j^{th} chain (or $*$ when the last task in the chain has terminated). For example $(\underline{p}_1, \overline{p}_2, \overline{p}_3, p_4, \overline{p}_5)$ is written as $(\overline{m}_3, m_2)$ and $(\underline{p}_1, \overline{p}_2, \overline{p}_3, \underline{p}_4, \underline{p}_5)$ as $(\overline{m}_3, *)$. For the same reason we can re-use the same clock for all tasks that share the same chain. Note that the automaton is acyclic.

The relation between the automaton and the scheduling problem is rather straightforward. A configuration of the automaton is a state of the schedule where each clock measures the time since the initiation of an active task. When time passes without transitions, the evolution of clock values is the only change in the state of the schedule. We say that a run of $\mathcal{A}$ is *complete* if it starts at $(s, \mathbf{0})$ and the last step is a transition to f. The following is an evident non-deterministic generalization of the observation from [AM01] concerning the relation between runs and schedules:

Claim 1 (Runs and Schedules) *Let $\mathcal{J}$ be an uncertain job-shop specification and let $\mathcal{A}_D$ be its associated timed automaton.*

1. *Every complete run of $\mathcal{A}_D$ corresponds to an environment instance d and to a schedule s, feasible for $\mathcal{J}(d)$.*

2. *For every instance d and every feasible schedule of s for $\mathcal{J}(d)$ there is a corresponding complete run of $\mathcal{A}_D$.*

The length of the run and the length of its corresponding schedule coincide.

The correspondence is simple: for every task p, $s(p)$ is the time (since the beginning of the run) in which the automaton entered a global state in which p is active. The residual problem associated with a state of the shcedule is represented by the sub-automaton rooted in the corresponding configuration.

The automaton can be viewed as specifying a *game* between the scheduler and the environment. The envirnment can decide whether or not to take an "end" transition and terminate an active task, and the scheduler can decide whether or not to take some enabled "start" transition. A strategy is a function that maps any configuration of the automaton into either one of its transition successors or to the waiting "action". For example, at $(m_1, \overline{m}_3)$ there is a choice between moving to (m_1, m_3) by giving m_3 to J_2 or waiting until J_1 terminates m_1 and letting the environment take the automaton to $(\overline{m}_3, \overline{m}_3)$, from where the conflict concerning m_3 can be resolved in either of the two possibile ways.

A strategy is d-future optimal if from every configuration reachable in $\mathcal{A}_{D,d}$ it gives the shortest path to the final state. In the next section we use a simplified form of the definitions and the algorithm of [AM99] to find such strategies.

4 Optimal Strategies for Timed Automata

Let $\mathcal{J}$ be a job-shop specification and let $\mathcal{A}_{D,d} = (Q, C, s, f, I, \Delta)$ be the automaton corresponding to an instance d, that is, end transitions are guarded by conditions of the form $c_i \geq d(p_i)$. Let $h : Q \times V \to \mathbb{R}_+$ be a function such that $h(q, \mathbf{v})$ is the length of the minimal run from $(q, \mathbf{v})$ to f, assuming that all uncontrolled future transitions are taken according to d. This function admits the following recursive backward definition:

$$h(f, \mathbf{v}) = 0 \qquad h(q, \mathbf{v}) = \min\{t + h(q', \mathbf{v}') : (q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1}) \xrightarrow{0} (q', \mathbf{v}')\}.$$

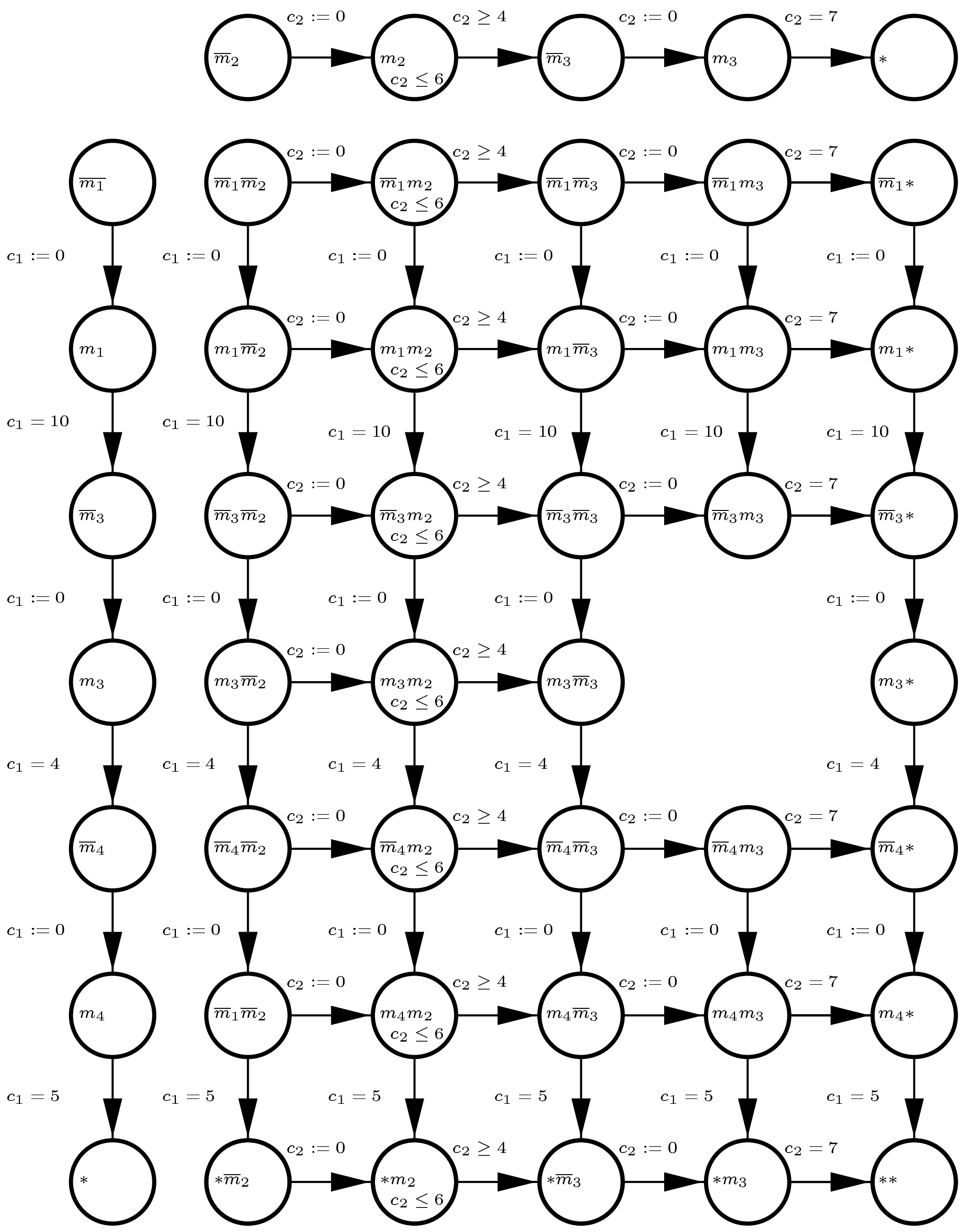

Figure 3: The global automaton for the job-shop specification. The automata on the upper and left parts of the figure are the partial compositions of the automata for the two chains.

Roughly speaking, $h(q, \mathbf{v})$ is the minimum over all immediate succssors q' of q of the time it takes from $(q, \mathbf{v})$ to satisfy the transition guard to q' plus the time to reach f from the resulting configuration $(q', \mathbf{v}')$. In [AM99] we have shown that h ranges over a class of "nice" functions closely related to the zones used in the verification of timed automata. There we have shown that this class is well-founded and hence the computation of h converges even for automata with cycles, a fact that we do not need here as h is computed in one sweep through all paths from the final to the initial state. Let us illustrate the computation of h on our example. We write the function in the form $h(\alpha^1, \alpha^2, c_1, c_2)$ and use $\perp$ to denote cases where the value of the corresponding clock is irrelevant (its task is not active). We start with:

$$h(*,*,\perp,\perp) = 0 \qquad h(m_4,*,c_1,\perp) = 5 \dot{-} c_1 \qquad h(*,m_3,\perp,c_2) = 7 \dot{-} c_2$$

The value of h at (m_4, m_3) depends on the values of both clocks which determine what will finish before, m_4 or m_3 and whether the shorter path goes via $(m_4, *)$ or $(*, m_3)$. Note that the corresponding transitions are both uncontroled and no decision of the scheduler is involved:

$$\begin{aligned} h(m_4, m_3, c_1, c_2) &= \min \left\{ \begin{array}{l} 7 \dot{-} c_2 + h(m_4, *, c_1 + 7 \dot{-} c_2, \perp), \\ 5 \dot{-} c_1 + h(*, m_3, \perp, c_2 + 5 \dot{-} x1) \end{array} \right\} \\ &= \min\{5 \dot{-} c_1, 7 \dot{-} c_2\} \\ &= \left\{ \begin{array}{ll} 5 \dot{-} c_1 & \text{if } c_2 \dot{-} c_1 \geq 2 \\ 7 \dot{-} c_2 & \text{if } c_2 \dot{-} c_1 \leq 2 \end{array} \right. \end{aligned}$$

This procedure goes higher and higher in the graph, computing h for the whole state-space $Q \times V$.

The extraction of a strategy from h is straightforward: if the optimum of h at $(q, \mathbf{v})$ is obtained via a controlled transition to q' we let $\sigma(q, \mathbf{v}) = q'$ otherwise if it is obtained via an uncontrolled transition we let $\sigma(q, \mathbf{v}) = \perp$. For example,

$$\begin{aligned} h(m_1, \overline{m}_3, c_1, \perp) &= \min\{16, 21 \dot{-} c_1\} \\ &= \left\{ \begin{array}{ll} 16 & \text{if } c_1 \leq 5 \\ 21 \dot{-} c_1 & \text{if } c_1 \geq 5 \end{array} \right. \end{aligned}$$

Here the choice of whether to give m_3 immediately to J_2 (the first line) or to wait until J_1 terminates m_1 (the second line) depends on the value of c_1 which indicates how far the step m_1 has progressed. If it is less than 5 it is better to move to (m_1, m_3) while if it is greater it is better to wait in $(m_1, \overline{m}_3)$ until $(\overline{m}_3, \overline{m}_3)$ and then move to $(m_3, \overline{m}_3)$. Note that if we assume that J_1 and J_2 started their first task simultaneously, the value of c_1 upon entering $(m_1, \overline{m}_3)$ is exactly the duration of m_2 in the instance.

The results of [AM01] concerning "non-lazy" schedules imply that optimal strategies have the additional property that if $\sigma(q, \mathbf{v}) = \perp$ then $\sigma(q, \mathbf{v}') = \perp$ for every $\mathbf{v}' \geq \mathbf{v}$. In other words, if an enabled controllable transition gives the optimum it should be taken as soon as possible. This fact will be used later in the implementation of the strategy.

Existing algorithms for timed automata work on sets, not on functions, and in order to apply them to the computation of h we use the following construction. Let $\mathcal{A}'$ be an auxiliary automaton obtained from $\mathcal{A}$ by adding a clock T which is never reset to zero. Clearly, if $(q, \mathbf{v}, T)$ is reachable in $\mathcal{A}'$ from the initial state $(s, \mathbf{0}, 0)$ then $(q, \mathbf{v})$ is reachable in $\mathcal{A}$ in time T. Let Θ be a positive integer larger then the longest path in the automaton. Starting from $(f, \perp, \Theta)$ and doing backward reachability we can construct a relational represntation of h. More precisely, if $(q, \mathbf{v}, T)$ is backward reachable in $\mathcal{A}'$ from $(f, \{\bot\}, \Theta)$ then f is forward reachable in $\mathcal{A}$ from $(q, \mathbf{v})$ within $\Theta - T$ time.

We recall some commonly-used definitions in the verification of timed automata [HNSY94]. A *zone* is a subset of V consisting of points satisfying a conjunction of inequalities of the form $c_i - c_j \geq d$ or $c_i \geq d$. A *symbolic state* is a pair (q, Z) where q is a discrete state and Z is a zone. It denotes the set of configurations $\{(q, \mathbf{v}) : \mathbf{v} \in Z\}$. Zones and symbolic states are closed under various operations including the following:

- The *time predecessors* of (q, Z) is the set of configurations from which (q, Z) can be reached by letting time progress:
$$Pre^t(q, Z) = \{(q, \mathbf{v}) : \mathbf{v} + r\mathbf{1} \in Z, r \geq 0\}.$$
- The *δ-transition predecessor* of (q, Z) is the set of configurations from which (q, Z) is reachable by taking the transition $\delta = (q', \phi, \rho, q) \in \Delta$:
$$Pre^\delta(q, Z) = \{(q', \mathbf{v}') : \mathbf{v}' \in \mathrm{Reset}_\rho^{-1}(Z) \cap \phi\}.$$
- The *predecessors* of (q, Z) is the set of all configuration from which (q, Z) is reachable by any transiton δ followed by passage of time:
$$Pre(q, Z) = \bigcup_{\delta \in \Delta} Pre^t(Pre^\delta(q, Z)).$$
  The result can be represented as a set of symbolic states.

Algorithm 1 is based on the standard backaward reachability algorithm for timed automata. It starts with the final state of $\mathcal{A}'$ in a waiting list and outputs the set R of all backward-reachable symbolic states. In order to be able to extract strategies we store tuples of the form (q, Z, q') such that Z is a zone of $\mathcal{A}'$ and q' is the successor of q from which q was reached backwards.

Algorithm 1 (Backward Reachability for Timed Automata)

```
Waiting:={$(f, (\{\bot\}, \Theta), \emptyset)$};
Explored:=$\emptyset$;
while Waiting $\neq \emptyset$ do
  Pick $(q, Z, q'') \in$ Waiting;
  For every $(q', Z') \in Pre(q, Z)$;
    Insert $(q', Z', q)$ into Waiting;
  Move $(q, Z, q'')$ from Waiting to Explored
end
R:=Explored;
```

The set R gives sufficient information for implementing the strategy. Whenever a transition to $(q, \mathbf{v})$ is done during the execution we look at all the symbolic states with q and find
$$\max\{T : (\mathbf{v}, T) \in Z \wedge (q, Z, q') \in R\}.$$
If q' is a successor via a controlled transition, we move to q', otherwise we wait until a task terminates and an uncontrolled transition is taken. Non-laziness guarantees that we need not revise a decision to wait until the next transition. This concludes the major contribution of this paper, an algorithm for computing d-future optimal strategies for the problem of job-shop scheduling under uncertainty.

5 Experimental Results

5.1 Schedule Quality

We have implemented the algorithm using the zone library of Kronos [BDM+98]. As a benchmark we took the following problem with 5 jobs and 4 machines:

$$\begin{array}{l} J_1: (m_1,[4,25]) \prec (m_4,[49,60]) \prec (m_2,[41,55]) \prec (m_3,[4,17]) \\ J_2: (m_1,[8,20]) \prec (m_3,[14,35]) \prec (m_2,[13,25]) \prec (m_4,[62,89]) \\ J_3: (m_4,[12,36]) \prec (m_3,[7,15]) \prec (m_2,[35,54]) \prec (m_1,[49,61]) \\ J_4: (m_1,[35,78]) \prec (m_2,[21,30]) \prec (m_3,[1,10]) \prec (m_4,[12,26]) \\ J_5: (m_2,[1,30]) \prec (m_3,[26,40]) \prec (m_1,[32,56]) \prec (m_4,[14,21]) \end{array}$$

The static worst-case schedule for this problem is 241. We have applied Algorithm 1 to find d-future optimal strategies based on three instances that correspond, respectively, to "optimisitc", "realistic" and "pessimistic" predictions. For every p such that $D(p) = [l, u]$ they are defined as

$$d_{min}(p) = l \quad d_{avg}(p) = (l+u)/2 \quad d_{max}(p) = u.$$

Intuitively the performance of a a pessimistic strategy should be better on large instances than that of an optimisitic strategy, etc. To confirm this intuition we generated randomly three sets of instances: *Small*, *Medium* and *Large* where $d(p)$ is drawn uniformly from the intervals $[l,(l+u)/2]$, $[(3l+u)/4,(l+3u)/4]$ and $[(l+u)/2,u]$, respectively. For each instance d we have compared the results of 5 schedulers: an optimal clairvoyant scheduler[5] that knows d in advance, a static worst-case scheduler and the 3 adaptive schedulers derived using d_{min}, d_{avg} and d_{max}. The results are depicted in in the appendix.

It turns out that the pessimistic adaptive strategy is very good and robust. It gives schedules that on the average are only between 1% and 3% longer than those produced by a clairvoyant scheduler. For comparison, the static worst-case strategy deviates from the optimum by 9% on large instances and 37% on large instances. The pessimistic strategy is also better on medium instances than the "realistic" strategy and its quality on small instances is not siginificantly inferior to that of the two other strategies which seem to be less robust. The performance of the optimistic strategy on large instances is even inferior to that of the static worst-case scheduler. This can be explained by the fact that schedules that rely on the minimal prediction are almost always not executed as planned. Without ascribing a deep statistical significance to our experiments so far, they seem to clearly demonstrate the advanatges of being adaptive. Further experiments are still to be conducted.

5.2 Performance

Having demonstrated that adaptive strategies can lead to more efficient schedules, the question of scaling-up the results to larger problems remains. Currently the computation of a strategy for the 5×4 example takes around 20 minutes and there is not much hope to go beyond this size using exhaustive backward reachability. For the deterministic case, we have shown in [AM01] that rather large problems can be solved using forward reachability algorithms that need not use zones (only points in the clock space) and that can use intelligent search strategies combined with heuristics to prune the search space (heuristic search was first introduced for timed automata in [BFH+01]). Apparently this is not the case for uncertain problems where backward computations on zones is inevitable. The

[5] In the domain called *online algorithms* it is common to compare the performance of algorithms that receive their inputs progressively to a clairvoyant algorithm and the relation between their performances is called the *competitive ratio*.

reason is that, unlike deterministic problems where the scheduler can determine the set of reachable states, under uncertainty the environment can lead the automaton to a large portion of the discrete state-space and to uncountably-many clock valuations. The strategy needs to be defined for all of them.

We are investigating some ideas to reduce the set of states for which a strategy needs to be computed. The first idea is to start with a preliminary forward search to eliminate discrete states which are not reachable under any reasonable strategy, whatever the instance is. Intuitively, we eliminate "lazy" paths where enabled tasks are not activated even though they cannot block other tasks under any instance. This procedure leads to a sub-sutomaton to which algorithm 1 can be applied. The potential performance improvement of this idea seems, however, limited. Novel ideas that combine (parametrized) forward search with some approximations need to be developed.

6 Conclusions

We have developed a conceptual framework that allows to formulate and solve optimal scheduling problems under uncertainty. This framework goes beyond worst-case reasoning without resorting to costly probabilistic computations. This work also sheds some light on the applicability of forward and backward algorithms to scheduling problems and controller synthesis problems in general.

References

[AM01] Y. Abdedaïm and O. Maler, Job-Shop Scheduling using Timed Automata in G. Berry, H. Comon and A. Finkel (Eds.), *Proc. CAV'01*, 478-492, LNCS 2102, Springer 2001.

[AGP99] K. Altisen, G. Goessler, A. Pnueli, J. Sifakis, S. Tripakis and S. Yovine, A Framework for Scheduler Synthesis, *Proc. RTSS'99*, 154-163, IEEE, 1999.

[AM99] E. Asarin and O. Maler, As Soon as Possible: Time Optimal Control for Timed Automata, in F. Vaandrager and J. van Schuppen (Eds.), *Hybrid Systems: Computation and Control* 19-30, LNCS 1569, Springer, 1999.

[AMPS98] E. Asarin, O. Maler, A. Pnueli and J. Sifakis, Controller Synthesis for Timed Automata, *Proc. IFAC Symposium on System Structure and Control*, 469-474, Elsevier, 1998.

[BFH$^+$01] G. Behrmann, A. Fehnker T.S. Hune, K.G. Larsen, P. Pettersson and J. Romijn, Efficient Guiding Towards Cost-Optimality in UPPAAL, in T. Margaria and W. Yi (Eds.), *Proc. TACAS 2001*, 174-188, LNCS 2031, Springer, 2001.

[BDM$^+$98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, Kronos: a Model-Checking Tool for Real-Time Systems, *Proc. CAV'98*, LNCS 1427, Springer, 1998.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, Symbolic Model-checking for Real-time Systems, *Information and Computation* 111, 193–244, 1994.

Appendix: Schedule Quality Benchmark

The follwing tables show schedule quality for the small, medium and large benchmarks. Column “opt” denotes the clarivoyant optimal scheduler, “static” — the worst-case non-adaptive scheduler (the variations in length are due to the termination of the last task) and then the optimisitc, realistic and pessimistic schedulers. The % coulmns specify the deviation from the optimum.

inst	opt	static	%	min	%	avg	%	max	%
1	169	237	40	175	4	180	7	182	8
2	175	236	35	187	7	179	2	175	0
3	177	235	33	178	1	183	3	183	3
4	171	237	39	196	15	171	0	177	4
5	173	237	37	173	0	181	5	181	5
6	173	236	36	176	2	174	1	177	2
7	167	236	41	167	0	175	5	175	5
8	176	237	35	192	9	176	0	176	0
9	166	235	42	166	0	170	2	173	4
10	167	236	41	168	1	168	1	170	2
11	178	237	33	178	0	180	1	180	1
12	172	235	37	172	0	181	5	181	5
13	176	237	35	181	3	176	0	176	0
14	175	235	34	175	0	180	3	180	3
15	176	235	34	182	3	177	1	180	2
16	173	237	37	173	0	176	2	179	3
17	165	236	43	165	0	170	3	170	3
18	164	234	43	165	1	166	1	166	1
19	173	237	37	173	0	184	6	184	6
20	172	234	36	172	0	176	2	175	2
avg	171.90	235.95	37.40	175.70	2.30	176.15	2.50	177	2.95

inst	opt	static	%	min	%	avg	%	max	%
1	205	239	17	205	0	205	0	205	0
2	199	235	18	218	10	210	6	201	1
3	195	236	21	203	4	203	4	205	5
4	201	238	18	201	0	201	0	201	0
5	200	239	20	223	12	203	2	203	2
6	198	237	20	199	1	198	0	200	1
7	195	236	21	209	7	202	4	197	1
8	197	236	20	219	11	197	0	199	1
9	190	237	25	196	3	195	3	197	4
10	181	236	30	198	9	190	5	190	5
11	200	238	19	200	0	204	2	204	2
12	202	237	17	217	7	202	0	202	0
13	188	239	27	214	14	203	8	195	4
14	196	237	21	216	10	205	5	197	1
15	202	237	17	221	9	202	0	202	0
16	196	238	21	212	8	206	5	206	5
17	191	235	23	191	0	194	2	194	2
18	195	237	22	203	4	196	1	195	0
19	201	239	19	218	8	206	2	206	2
20	194	236	22	203	5	200	3	196	1
avg	196.30	237.10	20.90	208.30	6.10	201.10	2.60	199.75	1.85

inst	opt	static	%	min	%	avg	%	max	%
1	213	239	12	236	11	223	5	214	0
2	239	241	1	264	10	264	10	239	0
3	213	237	11	227	7	216	1	220	3
4	195	237	22	205	5	198	2	195	0
5	227	239	5	249	10	242	7	227	0
6	201	237	18	231	15	208	3	205	2
7	206	238	16	235	14	214	4	206	0
8	215	239	11	245	14	225	5	215	0
9	234	240	3	275	18	247	6	234	0
10	229	241	5	245	7	245	7	229	0
11	223	239	7	253	13	252	13	223	0
12	233	238	2	260	12	243	4	233	0
13	220	240	9	251	14	234	6	230	5
14	219	240	10	252	15	235	7	233	6
15	218	239	10	265	22	223	2	221	1
16	223	238	7	245	10	235	5	223	0
17	227	241	6	251	11	230	1	227	0
18	218	238	9	236	8	223	2	219	0
19	214	240	12	244	14	217	1	214	0
20	228	240	5	251	10	229	0	228	0
avg	219.75	239.05	9.05	246	12	230.15	4.50	221.75	0.85