Preemptive Job-Shop Scheduling using Stopwatch Automata*

Yasmina Abdeddaïm and **Oded Maler**
Verimag,Centre Equation, 2, av. de Vignate 38610 Gières, France
Yasmina.Abdeddaim@imag.fr Oded.Maler@imag.fr

Abstract

In this paper we show how the problem of job-shop scheduling where the jobs are preemptible can be modeled naturally as a shortest path problem defined on an extension of timed automata, namely stopwatch automata where some of the clocks might be freezed at certain states. Although general verification problems on stopwatch automata are known to be undecidable, we show that due to particular properties of optimal schedules, the shortest path in the automaton belongs to a finite subset of the set of acyclic paths and hence the problem is solvable. We present several algorithms and heuristics for finding the shortest paths in such automata and test their implementation on numerous benchmark examples.

Introduction

In (AM01) we have described a first step in a research programme intended to re-formulate scheduling problems using (timed) automata-based formalisms. Apart from the undeniable joy of re-inventing the wheel, this work is motivated by the belief that such automata provide timing problems with faithful state-based dynamic models on which a systematic study of semantic and computational problems can be done — the reader is referred to (AM01) for some of the motivation and background and to (AM99; AGP99; NTY00; NY01; BFH$^+$01) for other recent results in this spirit. In this framework the runs of the timed automaton correspond to feasible schedules and finding a time-optimal schedule amounts to finding the shortest path (in terms of elapsed time) in the automaton. In (AM01) we have shown how this works nicely for the job-shop scheduling problem which can be modeled by a certain class of *acyclic* timed automata, having finitely many qualitative[1] runs. Each such qualitative run is an equivalence class of a non-countable number of quantitative runs, but as we have shown, one of those (a "non-lazy" run which makes transitions as soon as possible) is sufficient to find the optimum over the whole class. These observations allowed us to apply efficient search algorithms over single configurations of clocks rather than work with zones.

In this work we extend these results to preemptible jobs, i.e. jobs that can use a machine for some time, stop for a while and then resume from where they stopped. Such situations are common, for example, when the machines are computers. While extending the framework of (AM01) to treat this situation we encounter two problems:

1. The corresponding class of automata goes beyond timed automata because clocks are stopped but not reset to zero when a job is preempted. General reachability problems for such stopwatch automata (also known as *integration graphs*) are known to be undecidable (C92; KPSY99).
2. Due to preemption and resumption, which corresponds to a loop in the underlying transition graph, the obtained automata are cyclic (unlike the non-preemptive case) and they have an *infinite* number of qualitative runs.

We will show however that these problems can be overcome for the class of stopwatch automata that correspond to preemtible job shop problems, and that efficient algorithms can be constructed.

The rest of the paper is organized as follows. In section 2 we give a short introduction to the preemptive job-shop scheduling problem including a fundamental property of optimal schedules. In section 3 we recall the definition of stopwatch automata and show how to transform a job-shop specification into such an automaton whose runs correspond to feasible schedules. In section 4 we describe efficient algorithms for solving the shortest-path problem for these automata (either exactly or approximately) and report the performance results of their prototype implementation on numerous benchmark examples.

Preemptive Job-Shop Scheduling

The Job-shop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given durations) by different tasks. The goal is to find a way to allocate the resources such that all the tasks terminate as soon as possible. We consider throughout the paper a fixed set M of resources. A *step* is a pair (m, d) where $m \in M$ and $d \in \mathbb{N}$, indicating the required utilization of resource m for time duration d. A *job specification* is a finite sequence

$$J = (m_1, d_1), (m_2, d_2), \ldots, (m_k, d_k) \quad (1)$$

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

[1] By a qualitative run of a timed automaton we mean a sequence of states and transitions without metric timing information.

of steps, stating that in order to accomplish job J, one needs to use machine m_1 for d_1 time, then use machine m_2 for d_2 time, etc.

Definition 1 (Job-Shop Specification) *Let M be a finite set of resources (machines). A job specification over M is a triple $J = (k, \mu, d)$ where $k \in \mathbb{N}$ is the number of steps in J, $\mu : \{1..k\} \to M$ indicates which resource is used at each step, and $d : \{1..k\} \to \mathbb{N}$ specifies the length of each step. A job-shop specification is a set $\mathcal{J} = \{J^1, \ldots, J^n\}$ of jobs with $J^i = (k^i, \mu^i, d^i)$.*

In order to simplify notations we assume that each machine is used exactly once by every job. We denote $\mathbb{R}_+$ by T, abuse $\mathcal{J}$ for $\{1, \ldots, n\}$ and let $K = \{1, \ldots, k\}$.

Definition 2 (Feasible Schedules) *Let $\mathcal{J} = \{J^1, \ldots, J^n\}$ be a job-shop specification. A feasible schedule for $\mathcal{J}$ is a relation $S \subseteq \mathcal{J} \times K \times T$ so that $(i, j, t) \in S$ indicates that job J^i is busy doing its j^{th} step at time t and, hence, occupies machine $\mu^i(j)$. We let T^i_j be the set of time instants where job $i \in \mathcal{J}$ is executing its j^{th} step, i.e. $T^i_j = \{t : (i, j, t) \in S\}$.[2] A feasible schedule should satisfy the following conditions:*

1. Ordering*: if $(i, j, t) \in S$ and $(i, j', t') \in S$ then $j < j'$ implies $t < t'$ (steps of the same job are executed in order).*
2. Covering*: For every $i \in \mathcal{J}$ and $j \in K$*
$$\int_{t \in T^i_j} dt \geq d^i(j)$$
(every step is executed).
3. Mutual Exclusion*: For every $i \neq i' \in \mathcal{J}$, $j, j' \in K$ and $t \in T$, if $(i, j, t) \in S$ and $(i', j', t) \in S$ then $\mu^i(j) \neq \mu^{i'}(j')$ (two steps of different jobs which execute at the same time do not use the same machine).*

Note that we allow a job to occupy the machine *after* the step has terminated. The *length* $|S|$ of a schedule is the supremal t over all $(i, j, t) \in S$. We say that a step j of job i is *enabled* in time t if $t \in \mathcal{E}^i_j = (\max T^i_{j-1}, \max T^i_j]$. The *optimal job-shop scheduling problem* is to find a schedule of a minimal length. This problem is known to be NP-hard (GJ79). From the relational definition of schedules one can derive the following commonly used definitions:

1. The *machine allocation function* $\alpha : M \times T \to \mathcal{J}$ stating which job occupies a machine at any time, defined as $\alpha(m, t) = i$ if $(i, j, t) \in S$ and $\mu^i(j) = m$.
2. The *task progress function* $\beta : \mathcal{J} \times T \to M$ stating what machine is used by a job is at a given time, defined as $\beta(i, t) = m$ if $(i, j, t) \in S$ and $\mu^i(j) = m$.

These functions are partial — a machine or a job might be idle at certain times.

Example 1: Consider $M = \{m_1, m_2, m_3\}$ and two jobs $J^1 = (m_1, 3), (m_2, 2), (m_3, 4)$ and $J^2 = (m_2, 5)$. Two schedules S_1 and S_2 appear in Figure 1. The length of S_1 is 9 and it is the optimal schedule. As one can see, at $t = 3$, J^1 preempts J^2 and takes machine m_2.

[2] We may assume further that T^i_j is can be decomposed into a countable number of left-closed right-open intervals.

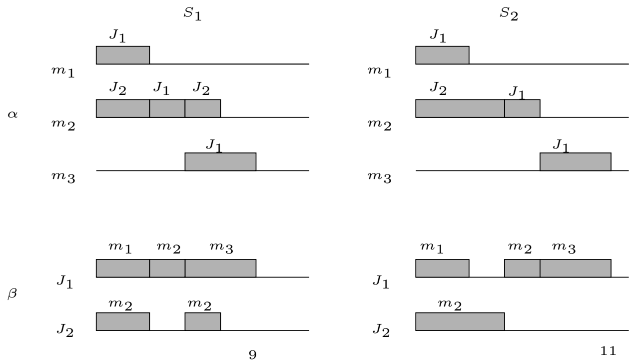

Figure 1: Two schedule S_1 and S_2 visualized as the machine allocation function α and the task progress function β.

We conclude this section with a reformulation of a well-known result concerning optimal preemptive schedules which will be used later. In essence this result formalizes the following two intuitive observations: *1) When jobs can be preempted and resumed at no cost, there is no reason to delay a step not being in a conflict with another. 2) Two jobs that keep on preempting each other do not contribute to the general progress.*

Definition 3 (Conflicts and Priorities) *Let S be a feasible schedule. We say that job i is in conflict with job i' on machine m in S (denoted by $i \nparallel_m i'$) when there are two respective steps j and j' such that $\mu^i(j) = \mu^{i'}(j') = m$ and $\mathcal{E}^i_j \cap \mathcal{E}^{i'}_{j'} \neq \emptyset$. We say that i has priority on m over a conflicting job i' (denoted by $i \prec_m i'$) if it finishes using m before i' does, i.e. $\sup T^i_j < \sup T^{i'}_{j'}$.*

Note that conflicts and priorities are always induced by a schedule S although S is omitted from the notation.

Definition 4 (Efficient Schedules) *A schedule S is efficient if for every job i and a step j such that $\mu^i(j) = m$, job i uses m during all the time interval $\mathcal{E}^i_j$ except for times when another job i' such that $i' \prec_m i$ uses it.*

The following is a folk theorem, whose roots go back at least to (J55) with some reformulation and proofs in, for example, (CP90; PB96).

Theorem 0.1 (Efficiency is Good) *Every preemptive job-shop specification admits an efficient optimal schedule.*

Sketch of Proof: The proof is by showing that every inefficient schedule S can be transformed into an efficient schedule S' with $|S'| \leq |S|$. Let I be the first interval when inefficiency occurs for job i and machine m. We modify the schedule by shifting some of the later use of m by i into I. If m was occupied during I by another job i' such that $i \prec_m i'$, we give it the time slot liberated by i. The termination of the step by i' is not delayed by this modification because it happens anyway after i terminates its step. ■

As an illustration consider the schedules appearing in Figure 2 with $J_1 \prec_m J_2 \prec_m J_3$ and where J_2 is enabled in the interval $[t_1, t_2]$. The first inefficiency in S_1 is eliminated in S_2 by letting J_2 use the free time slot before the arrival of

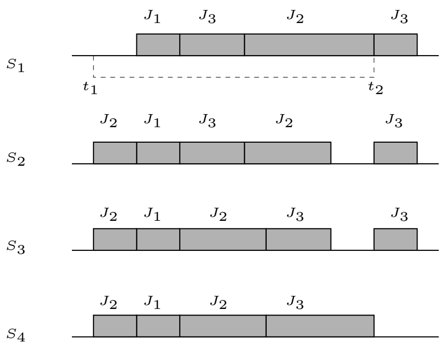

Figure 2: Removal of inefficiency, $J_1 \prec J_2 \prec J_3$.

J_1. The second inefficiency occurs when J_3 uses the machine while J_2 is waiting, and it is removed in S_3. The last inefficiency where J_3 is waiting while m is idle is removed in S_4.

This result reduces the set of candidates for optimality from the non-countable set of feasible schedules to the finite set of efficient schedules, each of which corresponds to a fixed priority relation.[3] There are potentially $kn!$ priority relations but only a fraction of those needs to be considered because when i and i' are never in conflict concerning m, the priority $i \prec_m i'$ has no influence on the schedule.

Stopwatch Automata

Timed automata (AD94) are automata augmented with continuous clock variables whose values grow uniformly at every state. Clocks are reset to zero at certain transitions and tests on their values are used as pre-conditions for transitions. Hence they are ideal for describing concurrent time-dependent behaviors. There are however situations, preemptive scheduling being among those, in which we need to measure the overall accumulated time that a systems spends in some state. This motivated the extension of the model to have clocks with derivative zero at certain states. Unlike timed automata, the reachability problem for these automata is undecidable (C92). Some sub-classes, *integration graphs*, were investigated in (KPSY99), where a decision procedure based on reducing the problem into linear constraint satisfaction was reported. Similar automata were studied in (MV94) and in (CL00) where an implementation of an approximate verification algorithm was described.

Definition 5 (Stopwatch Automaton)
A stopwatch automaton is a tuple $\mathcal{A} = (Q, C, s, f, \mathbf{u}, \Delta)$ where Q is a finite set of states, C is a finite set of n clocks, $\mathbf{u} : Q \rightarrow \{0,1\}^n$ assigns a constant slope to every state 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 boolean combination of formulae of the form $(c \in I)$ for some clock c and some integer-bounded interval I. States s and f are the initial and final states, respectively.

[3] This might explain the popularity of priority-based approach in computer scheduling.

A *clock valuation* is a function $\mathbf{v} : C \rightarrow \mathbb{R}_+ \cup \{0\}$, or equivalently a $|C|$-dimensional vector over $\mathbb{R}_+$. We denote the set of all clock valuations by $\mathcal{H}$. A configuration of the automaton is hence a pair $(q, \mathbf{v}) \in Q \times \mathcal{H}$ consisting of a discrete state (sometimes called "location") and a clock valuation. Every subset $\rho \subseteq C$ induces a reset function $\mathrm{Reset}_\rho : \mathcal{H} \rightarrow \mathcal{H}$ 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 others unchanged. We use $\mathbf{1}$ to denote the unit vector $(1, \ldots, 1)$, $\mathbf{0}$ for the zero vector and $\mathbf{u}_q$ for $\mathbf{u}(q)$, the derivative of the clocks at q.

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: $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{u}_q)$, $t \in \mathbb{R}_+$.

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

$$\xi : \quad (q_0, \mathbf{v}_0) \xrightarrow{t_1} (q_1, \mathbf{v}_1) \xrightarrow{t_2} \cdots \xrightarrow{t_l} (q_l, \mathbf{v}_l).$$

The *logical length* of such a run is l and its *metric length* is $|\xi| = t_1 + t_2 + \cdots + t_l$. Note that discrete transitions take no time.

Next we construct for every job $J = (k, \mu, d)$ a timed automaton with one clock such that for every step j with $\mu(j) = m$ there are three states: a state $\overline{m}$ which indicates that the job is waiting to start the step, a state m indicating that the job is executing the step and a state $\tilde{m}$ indicating that the job is preempted after having started. Upon entering m the clock is reset to zero, and measures the time spent in m. Preemption and resumption are modeled by transitions to and from state $\tilde{m}$ in which the clock does not progress. When the clock value reaches $d(j)$ the automaton can leave m to the next waiting state. Let $\overline{M} = \{\overline{m} : m \in M\}$, $\tilde{M} = \{\tilde{m} : m \in M\}$ and let $\overline{\mu} : K \rightarrow \overline{M}$ and $\tilde{\mu} : K \rightarrow \tilde{M}$ be auxiliary functions such that $\overline{\mu}(j) = \overline{m}$ and $\tilde{\mu}(j) = \tilde{m}$ whenever $\mu(j) = m$.

Definition 6 (Stopwatch Automaton for a Job) *Let $J = (k, \mu, d)$ be a job. Its associated automaton is $\mathcal{A} = (Q, \{c\}, u, \Delta, s, f)$ with $Q = P \cup \overline{P} \cup \tilde{P} \cup \{f\}$ where $P = \{\mu(1), \ldots \mu(k)\}$, $\overline{P} = \{\overline{\mu}(1), \ldots, \overline{\mu}(n)\}$ and $\tilde{P} = \{\tilde{\mu}(1), \ldots, \tilde{\mu}(n)\}$. The slope is defined as $u_q = 1$ when $q \in P$ and $u_q = 0$ otherwise.*[4] *The transition relation Δ consists of the following types of tuples*

type	q	ϕ	ρ	q'	
1) begin	$\overline{\mu}(j)$	*true*	$\{c\}$	$\mu(j)$	$j = 1..k$
2) pause	$\mu(j)$	*true*	$\emptyset$	$\tilde{\mu}(j)$	$j = 1..k$
3) resume	$\tilde{\mu}(j)$	*true*	$\emptyset$	$\mu(j)$	$j = 1..k$
4) end	$\mu(j)$	$c \geq d(j)$	$\emptyset$	$\overline{\mu}(j+1)$	$j = 1..k-1$
end	$\mu(k)$	$c \geq d(k)$	$\emptyset$	f	

[4] Note that the slope at state $\overline{m}$ can be arbitrary because clock c is *inactive* in this state: it is reset to zero without being tested upon leaving $\overline{m}$.

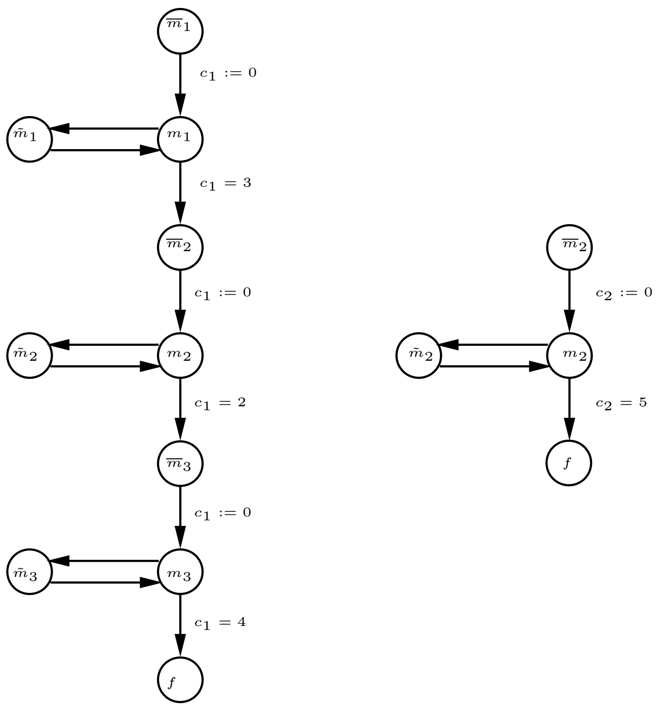

Figure 3: The automata corresponding to the jobs $J^1 = (m_1, 3), (m_2, 2), (m_3, 4)$ and $J^2 = (m_2, 5)$.

The initial state is $\overline{\mu}(1)$.

The automata for the two jobs in Example 1 are depicted in Figure 3.

For every automaton $\mathcal{A}$ we define a *ranking function* $g : Q \times \mathbb{R}_+ \to \mathbb{R}_+$ such that $g(q, v)$ is a lower-bound on the time remaining until f is reached from (q, v):

$$\begin{aligned}
&g(f, v) = 0 \\
&g(\overline{\mu}(j), v) = \textstyle\sum_{l=j}^{k} d(l) \\
&g(\mu(j), v) = g(\overline{\mu}(j), v) - \min\{v, d(j)\} \\
&g(\tilde{\mu}(j), v) = g(\overline{\mu}(j), v) - \min\{v, d(j)\}
\end{aligned}$$

In order to obtain the timed automaton representing the whole job-shop specification we need to compose the automata for the individual tasks. The composition is rather standard, the only particular feature is the enforcement of mutual exclusion constraints by forbidding global states in which two or more automata are in a state corresponding to the same resource m. An n-tuple $q = (q^1, \ldots, q^n) \in (M \cup \overline{M} \cup \tilde{M} \cup \{f\})^n$ is said to be *conflicting* if it contains two distinct components q^i and $q^{i'}$ such that $q^i = q^{i'} = m \in M$.

Definition 7 (Mutual Exclusion Composition) *Let $\mathcal{J} = \{J^1, \ldots, J^n\}$ be a job-shop specification and let $\mathcal{A}^i = (Q^i, C^i, u^i, \Delta^i, s^i, f^i)$ be the automaton corresponding to each J^i. Their mutual exclusion composition is the automaton $\mathcal{A} = (Q, C, \mathbf{u}, \Delta, s, f)$ such that Q is the restriction of $Q^1 \times \ldots Q^n$ to non-conflicting states, $C = C^1 \cup \ldots \cup C^n$, $s = (s^1, \ldots, s^n)$, $f = (f^1, \ldots, f^n)$. The slope $\mathbf{u}_q$ for a global state $q = (q^1, \ldots q^n)$ is $(u_{q^1}, \ldots, u_{q^n})$ 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 the global states $(q^1, \ldots, q^i, \ldots, q^n)$ and $(q^1, \ldots, p^i, \ldots, q^n)$ are non-conflicting.

Part of the automaton obtained by composing the two automata of Figure 3 appears in Figure 4. We have omitted the *pause/resume* transitions for m_1 and m_3 as well as some other non-interesting paths.

A run of $\mathcal{A}$ is *complete* if it starts at $(s, \mathbf{0})$ and the last step is a transition to f. From every complete run ξ one can derive in an obvious way a schedule S_ξ such that $(i, j, t) \in S_\xi$ if at time t the i^{th} component of the automaton is at state $\mu(j)$. The length of S_ξ coincides with the metric length of ξ.

Claim 1 (Runs and Schedules) *Let $\mathcal{A}$ be the automaton generated for the preemptive job-shop specification $\mathcal{J}$ according to Definitions 6 and 7. Then:*

1. *For every complete run ξ of $\mathcal{A}$, its associated schedule S_ξ is feasible for $\mathcal{J}$.*
2. *For every feasible schedule S for $\mathcal{J}$ there is a run ξ of $\mathcal{A}$ such that $S_\xi = S$.*

Corollary 1 (Preemptive Scheduling and Stopwatch Automata)

The optimal preemptive job-shop scheduling problem can be reduced to the problem of finding the shortest path in a stopwatch automaton.

The two schedules of Figure 1 correspond to the following two runs (we use the notation $\perp$ to indicate inactive clocks):

S_1 :
$(\overline{m}_1, \overline{m}_2, \perp, \perp) \xrightarrow{0} (m_1, \overline{m}_2, 0, \perp) \xrightarrow{0} (m_1, m_2, 0, 0) \xrightarrow{3}$
$(m_1, m_2, 3, 3) \xrightarrow{0} (\overline{m}_2, m_2, \perp, 3) \xrightarrow{0} (\overline{m}_2, \tilde{m}_2, \perp, 3) \xrightarrow{0}$
$(m_2, \tilde{m}_2, 0, 3) \xrightarrow{2} (m_2, \tilde{m}_2, 2, 3) \xrightarrow{0} (\overline{m}_3, \tilde{m}_2, \perp, 3) \xrightarrow{0}$
$(\overline{m}_3, m_2, \perp, 3) \xrightarrow{0} (m_3, m_2, 0, 3) \xrightarrow{2} (m_3, m_2, 2, 5) \xrightarrow{0}$
$(m_3, f, 2, \perp) \xrightarrow{2} (m_3, f, 4, \perp) \xrightarrow{0} (f, f, \perp, \perp)$

S_2 :
$(\overline{m}_1, \overline{m}_2, \perp, \perp) \xrightarrow{0} (m_1, \overline{m}_2, 0, \perp) \xrightarrow{0} (m_1, m_2, 0, 0) \xrightarrow{3}$
$(m_1, m_2, 3, 3) \xrightarrow{0} (\overline{m}_2, m_2, \perp, 3) \xrightarrow{2} (\overline{m}_2, m_2, \perp, 5) \xrightarrow{0}$
$(\overline{m}_2, f, \perp, \perp) \xrightarrow{0} (m_2, f, 0, \perp) \xrightarrow{2} (m_2, f, 2, \perp) \xrightarrow{0}$
$(\overline{m}_3, f, \perp, \perp) \xrightarrow{0} (m_3, f, 0, \perp) \xrightarrow{4} (m_3, f, 4, \perp) \xrightarrow{0}$
$(f, f, \perp, \perp)$

The job-shop automaton admits a special structure: ignoring the *pause* and *resume* transitions, the automaton is *acyclic* and its state-space admits a natural partial-order. It can be partitioned into levels according to the number of *begin* and *end* transitions from s to the state. There are no staying conditions (invariants) and the automaton can stay forever in any given state. Recall that in any automaton extended with auxiliary variables the transition graph might be misleading, because two or more transitions entering *the*

	state	action	new state	remark
1	$(\overline{m}, \overline{m})$	start 1	$(m, \overline{m})$	
2	$(\overline{m}, \tilde{m})$	start 1	$(m, \tilde{m})$	
3	$(\overline{m}, m)$	preempt 2	$(\overline{m}, \tilde{m})$	
4	$(\tilde{m}, \overline{m})$	resume 1	$(m, \overline{m})$	
5	$(\tilde{m}, \tilde{m})$	resume 1	$(m, \tilde{m})$	
6	$(\tilde{m}, m)$			(impossible)
7	$(m, \overline{m})$	(continue)	$(m, \overline{m})$	
8	$(m, \tilde{m})$	(continue)	$(m, \tilde{m})$	
9	(m, m)			(impossible)

Table 1: Resolving conflicts when $J_1 \preceq_m J_2$.

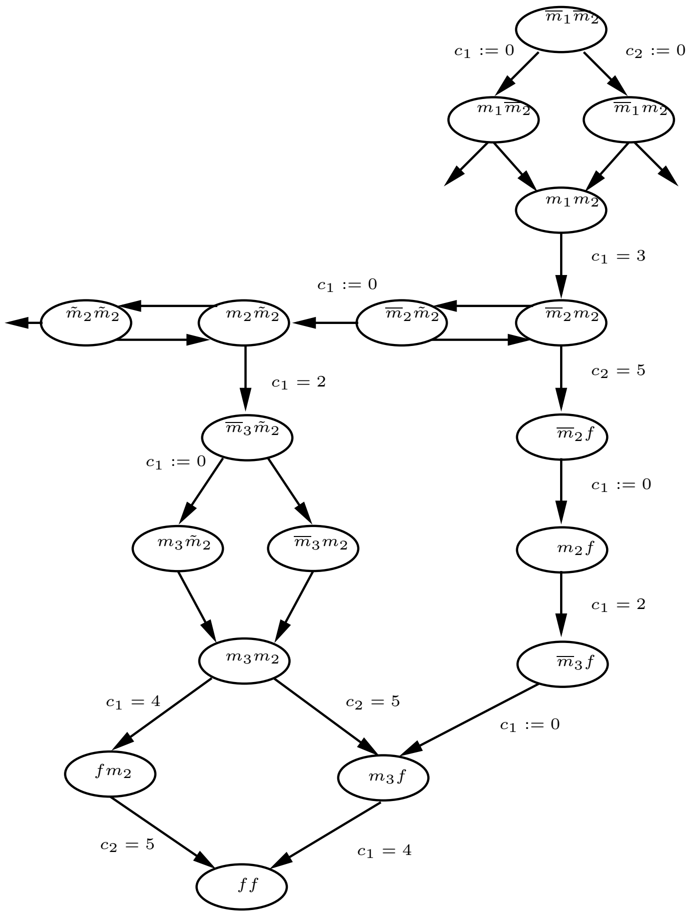

Figure 4: The global stopwatch automaton for the two jobs.

same discrete state, e.g. transitions to (m_3, f) in Figure 4, might enter it with different clock valuations, and hence lead to different continuations. Consequently, algorithms for verification and quantitative analysis might need to explore all the nodes in the unfolding of the automaton into a tree. Two transitions outgoing from the same state might represent a choice of the scheduler, for example, the two transitions outgoing from $(\overline{m}_2, m_2)$ represent the choice of whether or not to preempt J^2 and give machine m_2 to J^1. On the other hand some duplication of paths are just artifacts due to interleaving, for example, the two paths leading from $(\overline{m}_1, \overline{m}_2)$ to (m_1, m_2) are practically equivalent.

Another useful observation is that from every (preemptive or non-preemptive) job-shop specification $\mathcal{J}$ one can construct its reverse problem $\mathcal{J}'$ where the order of every individual job is reversed. Every feasible schedule for $\mathcal{J}'$ can be transformed easily into a feasible schedule for $\mathcal{J}$ having the same length. Doing a forward search on the automaton for $\mathcal{J}'$ is thus equivalent to doing a backward search on the automaton for $\mathcal{J}$.

Shortest Paths in Stopwatch Automata

In order to find shortest paths in stopwatch automata we will take advantage of Theorem 0.1 to restrict the search to runs whose corresponding schedules are efficient.

Definition 8 (Efficient Runs) *A run of a stopwatch automaton constructed according to Definitions 6 and 7 is efficient if all discrete transitions are taken as soon as they are enabled, and all conflicts are resolved according to a fixed priority relation.*

To be more precise, let J_1 and J_2 be two jobs which are in conflict concerning machine m and let J_1 be the one with the highest priority on m. Table depicts all the potential conflict situations and how they are resolved.

In situations 1, 2, 4, and 5 J_1 is waiting for the machine which is not occupied and so it takes it. Such situations could have been reached, for example, by a third job of higher priority releasing m or by J_1 finishing its prior step and entering $\overline{m}$. Situation 3 is similar but with J_2 occupying m and hence has to be preempted to reach situation 2. Situation 6, where J_1 is preempted and J_1 is executing, contradicts the priority and is not reachable. In situations 7 and

8, J_1 is executing and no preemption action is taken. Finally situation 9 violates mutual exclusion.

The restriction to efficient runs makes the problem decidable: we can just enumerate all priority relations, derive the schedules implied by each of them and compare their lengths. The search algorithm that we employ on the unfolding of the automaton generates priorities *on the fly* whenever two jobs come into conflict. In the example of Figure the first conflict is encountered in state $(\overline{m}_2, m_2)$ and from there we may choose between two options, either to continue with time passage or preempt J_2. in the first case we fix the priority $J_2 \prec J_1$ and let J_2 finish without considering preemption anymore while in the second case the priority is $J_1 \prec J_2$, we move to $(\overline{m}_2, \tilde{m}_2)$ and the transition back to $(\overline{m}_2, m_2)$ becomes forbidden. From there we can only continue to $(m_2, \tilde{m}_2)$ and let the time pass until J_1 releases m_2.

To formalize this we define a *valid successors* relation over tuples of the form $(q, \mathbf{x}, \Pi, \theta)$ where $(q, \mathbf{x})$ is a global configuration of the automaton, Π is a (partial) priority relation and θ is the total elapsed time for reaching $(q, \mathbf{x})$ from the initial state. When there are no immediate transitions enabled in $(q, \mathbf{x})$ we have

$$Succ(q, \mathbf{x}, \Pi, \theta) = \{(q, \mathbf{x} + t \cdot \mathbf{u}_q, \Pi, \theta + t)\}$$

where t is the minimal time until a transition becomes enabled, that is, the least t such that a guard on a transition from q is satisfied at $\mathbf{x} + t \cdot \mathbf{u}_q$.

When there are immediate transition enabled in $(q, \mathbf{x})$ we have

$$Succ(q, \mathbf{x}, \Pi, \theta) = L_1 \cup L_2 \cup L_3$$

where

$$L_1 = \{(q', \mathbf{x}', \Pi, \theta) : (q, \mathbf{x}) \xrightarrow{\tau} (q', \mathbf{x}')\}$$

for every immediate transition τ such that τ is non-conflicting or belongs to the job whose priority on the respective machine is higher than those of all competing jobs. In addition, if there is a conflict on m involving a new job i whose priority compared to job i^*, having the highest priority so far, has not yet been determined, we have

$$L_2 = \{(q, \mathbf{x}, \Pi \cup \{i^* \prec i\}, \theta)\}$$

and

$$L_3 = \{(q, \mathbf{x}, \Pi \cup \bigcup_{\{i' : i' \nparallel_m i\}} \{i \prec i'\}, \theta)\}.$$

The successor in L_2 represent the choice to prefer i^* over i (the priority of i relative to other waiting jobs will be determined only after i^* terminates), while S_3 represents the choice of preferring i over all other jobs.

Using this definition we can construct a search algorithm that explores all the efficient runs of $\mathcal{A}$.

Algorithm 1 (Forward Reachability for Stopwatch Automata)

Waiting:=$\{(s, \mathbf{0}, \emptyset, 0)\}$*;*
while *Waiting* $\neq \emptyset$*;* **do**
 Pick $(q, \mathbf{x}, \Pi, \theta) \in$ *Waiting;*
 For every $(q', \mathbf{x}', \Pi', \theta') \in Succ(q, \mathbf{x}, \Pi, \theta)$*;*
  Insert $(q', \mathbf{x}', \Pi', \theta')$ *into Waiting;*
 Remove $(q, \mathbf{x}, \Pi, \theta)$ *from Waiting*
end

The length of the shortest path is the least θ such that $(f, \mathbf{x}, \Pi, \theta)$ is explored by the algorithm.

This exhaustive search algorithm can be improved into a best-first search as follows (similar ideas were investigated in (BFH+01)). We define an evaluation function for estimating the quality of configurations.

$$E((q_1, \ldots, q_n), (v_1, \ldots, v_n), \Pi, \theta) = \theta + \max\{g^i(q_i, v_i)\}_{i=1}^n$$

where g^i is the previously-defined ranking function associated with each automaton $\mathcal{A}^i$. Note that $\max\{g^i\}$ gives the most optimistic estimation of the *remaining* time, assuming that no job will have to wait. It is not hard to see that $E(q, \mathbf{x}, \Pi, \theta)$ gives a lower bound on the length of every complete run which passes through $(q, \mathbf{x})$ at time θ.

The following algorithm orders the waiting list of configurations according to their evaluation. It is guaranteed to produce the optimal path because it stops the exploration only when it is clear that the unexplored states cannot lead to schedules better than those found so far.

Algorithm 2 (Best-first Forward Reachability)

Waiting:=$\{(s, \mathbf{0}, \emptyset, 0)\}$*;*
Best:=∞
$(q, \mathbf{x}, F, \theta)$*:= first in Waiting;*
while *Best* $> E(q, \mathbf{x}, F, \theta)$
do
 $(q, \mathbf{x}, \Pi, \theta)$*:= first in Waiting;*
 For every $(q', \mathbf{x}', \Pi', \theta') \in Succ(q, \mathbf{x}, \Pi, \theta)$*;*
  if $q' = f$ **then**
   Best:=$\min\{$*Best,*$E((q', \mathbf{x}', \Pi', \theta'))\}$
  else
   Insert $(q', \mathbf{x}', \Pi', \theta')$ *into Waiting;*
 Remove $(q, \mathbf{x}, \Pi, \theta)$ *from Waiting*
end

Using this algorithm we were able the find optimal schedules of systems with up to 8 jobs and 4 machines (12^8 discrete states and 8 clocks). In order to treat larger problems we abandon optimality and use a heuristic algorithm which can quickly generate sub-optimal solutions. The algorithm is a mixture of breadth-first and best-first search with a fixed number w of explored nodes at any level of the automaton. For every level we take the w best (according to E) nodes, generate their successors but explore only the best w among them, and so on.

In order to test this heuristics we took 16 problems among the most notorious job-shop scheduling problems.[5] For each of these problems we have applied our algorithms for different choices of w, both forward and backward (it takes, on the average few minutes for each problem). In Table we compare our best results on these problems to the most recent results reported by Le Pape and Baptiste (PB96; PB97) where the problem was solved using state-of-the-art constraint satisfaction techniques. As the table shows, the results our first prototype are very close to the optimum.

[5] The problems are taken from `ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt`

problem			non preempt	preemptive			
name	#j	#m	optimum	optimum	(PB96; PB97)	stopwatch	deviation
LA02	10	5	655	655	655	655	0.00 %
FT10	10	10	930	900	900	911	1.21 %
ABZ5	10	10	1234	1203	1206	1250	3.76 %
ABZ6	10	10	943	924	924	936	1.28 %
ORB1	10	10	1059	1035	1035	1093	5.31 %
ORB2	10	10	888	864	864	884	2.26 %
ORB3	10	10	1005	973	994	1013	3.95 %
ORB4	10	10	1005	980	980	1004	2.39 %
ORB5	10	10	887	849	849	887	4.28 %
LA19	10	10	842	812	812	843	3.68 %
LA20	10	15	902	871	871	904	3.65 %
LA21	10	15	1046	1033	1033	1086	4.88 %
LA24	10	15	936	909	915	972	6.48 %
LA27	10	20	1235	1235	1235	1312	5.87 %
LA37	15	15	1397	1397	1397	1466	4.71 %
LA39	15	15	1233	1221	1221	1283	4.83 %

Table 2: The results of our implementation on the benchmarks. Columns #j and #m indicated the number of jobs and machines, followed by the best known results for non-preemptive scheduling, the known optimum for the preemptive case, the results of Le Pape and Baptiste, followed by our results and their deviation from the optimum.

Conclusion

We have demonstrated that the automata-theoretic approach to scheduling can be extended to preemptive scheduling and can be applied successfully to very large systems. Future work will investigate the applicability of this approach to scheduling of periodic tasks in real-time systems. In retrospect, it looks as if the undecidability results for *arbitrary* stopwatch automata have been taken too seriously. Timed and stopwatch automata arising from specific application domains have additional structure and their analysis might turn out to be feasible (see also recent results in (FPY02)).

Acknowledgment: We thank Eugene Asarin and Stavros Tripakis for numerous useful comments.

References

Y. Abdeddaïm 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.

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.

R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183-235, 1994.

E. Asarin and O. Maler, As Soon as Possible: Time Optimal Control for Timed Automata, *Proc. HSCC'99*, 19-30, LNCS 1569, Springer, 1999.

G. Behrmann, A. Fehnker T.S. Hune, K.G. Larsen, P. Pettersson and J. Romijn, Efficient Guiding Towards Cost-Optimality in UPPAAL, *Proc. TACAS 2001*, 174-188, LNCS 2031, Springer, 2001.

J. Carlier and E. Pinson, A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem, *Annals of Operations Research* 26, 1990.

F. Cassez and K.G. Larsen, On the Impressive Power of Stopwatches, in C. Palamidessi (Ed.) *Proc. CONCUR'2000*, 138-152, LNCS 1877, Springer, 2000.

K. Cerans, *Algorithmic Problems in Analysis of Real Time System Specifications*, Ph.D. thesis, University of Latvia, Riga, 1992.

E. Fersman, P. Pettersson and W. Yi, Timed Automata with Asynchronous Processes: Schedulability and Decidability, *TACAS 2002*, this volume, 2002.

M. R. Garey and D. S Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

J. R. Jackson, Scheduling a Production Line to Minimize Maximum Tardiness, Research Report 43, *Management Sciences Research Project*, UCLA, 1955.

A.S. Jain and S. Meeran, Deterministic Job-Shop Scheduling: Past, Present and Future, *European Journal of Operational Research* 113, 390-434, 1999.

Y. Kesten, A. Pnueli, J. Sifakis and S. Yovine, Decidable Integration Graphs, *Information and Computation* 150, 209–243, 1999.

J. McManis and P. Varaiya, Suspension Automata: A Decidable Class of Hybrid Automata, in D.L Dill (Ed.), *Proc. CAV'94*, 105-117, LNCS 818, Springer, 1994.

P. Niebert, S. Tripakis S. Yovine, Minimum-Time Reachability for Timed Automata, *IEEE Mediteranean Control Conference*, 2000.

P. Niebert and S. Yovine, Computing Efficient Operation Schemes for Chemical Plants in Multi-batch Mode, *European Journal of Control* 7, 440-453, 2001.

C. Le Pape and P. Baptiste, A Constraint-Based Branch-and-Bound Algorithm for Preemptive Job-Shop Scheduling, *Proc. of Int. Workshop on Production Planning and Control*, Mons, Belgium, 1996.

C. Le Pape and P. Baptiste, An Experimental Comparaison of Constraint-Based Algorithms for the Preemptive Job-shop Sheduling Problem, *CP97 Workshop on Industrial Constraint-Directed Scheduling*, 1997.