Job-Shop Scheduling Using Timed Automata*

Yasmina Abdeddaïm and Oded Maler

Verimag, Centre Equation
2, av. de Vignate 38610 Gières, France
{Yasmina.Abdeddaim,Oded.Maler}@imag.fr

Abstract. In this paper we show how the classical job-shop scheduling problem can be modeled as a special class of acyclic timed automata. Finding an optimal schedule corresponds, then, to finding a shortest (in terms of elapsed time) path in the timed automaton. This representation provides new techniques for solving the optimization problem and, more importantly, it allows to model naturally more complex dynamic resource allocation problems which are not captured so easily in traditional models of operation research. We present several algorithms and heuristics for finding the shortest paths in timed automata and test their implementation in the tool Kronos on numerous benchmark examples.

1 Introduction

A significant part of verification consists in checking the *existence* of certain paths in very large transition graphs, given as a product (composition) of simpler graphs. Such paths correspond to bad behaviors of the system under consideration. On the other hand, in many application domains (optimal control, Markov decision processes, scheduling) we are interested in selecting, among the possible behaviors, one that *optimizes* some more sophisticated performance measure (note that in "classical" verification we use a very simple performance measure on behaviors, namely, they are either "good" or "bad"). Both verification and optimization suffer from the state-explosion problem, also known as "the curse of dimensionality", and various methods and heuristics have been developed in order to treat larger and larger problems. The main thrust of this work is to explore the possibility of exporting some of the ideas developed within the verification community, such as symbolic analysis of timed automata, to the domain of optimal scheduling, where most of the effort was directed toward a constrained optimization approach.

The observation underlying this paper is that classical scheduling and resource allocation problems can be modeled very naturally using timed automata whose runs correspond to feasible schedules. In this case, finding a time-optimal schedule amounts to finding the shortest path (in terms of elapsed time) in the automaton. This problem can be solved by some modifications in verification tools for timed automata. Posing the problem in automata-theoretic terms

* This work was partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems), and the AFIRST French-Israeli collaboration project 970maefut5 (Hybrid Models of Industrial Plants).

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 478–492, 2001.

might open the way to an alternative class of heuristics for intractable scheduling problems, coming from the experience of the verification community in analyzing large systems, and this might lead in the future to better algorithms for certain classes of scheduling problems. Even if they do not contribute to improving the performance, automata-based models have a clear *semantic* advantage over optimization-based models as they can model problems of scheduling under uncertainty (in arrival time and duration of tasks) and suggest solutions in terms of *dynamic* schedulers that observe the evolution of the plant.

Most of this work is devoted to establishing the link between the classical job-shop scheduling problem and timed automata and adapting the reachability algorithm of the tool Kronos to find shortest paths in timed automata. This is not a completely straightforward adaptation of standard graph-searching algorithms due to the density of the transition graph. We explore the performance limits of current timed automata technology, and although they cannot yet cope with the state-of-the-art in optimization, the results are rather encouraging.

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

2 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 (or "minimal makespan" in the scheduling jargon). We consider throughout the paper a fixed set M of resources. Intuitively, 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

$$J = (m_1, d_1), (m_2, d_2), \ldots, (m_k, d_k) \tag{1}$$

of steps, stating that in order the accomplish job J, one needs to use machine m_1 for d_1 time, then use machine m_2 for d_2 time, etc. The formal definition below tries to optimize the notations for the sequel.

Definition 1 (Job-Shop Specification). *Let M be a finite set of resources (machines). A job specification over a set M of resources 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 $J^i = (k^i, \mu^i, d^i)$.*

We make the following assumptions: 1) A job can wait an arbitrary amount of time between two steps. 2) Once a job starts to use a machine, it is not preempted until the step terminates. 3) Each machine is used exactly once by every job.[1]

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 $\mu^i(j)$. 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 and Non-Preemption*: For every $i\in\mathcal{J}$ and $j\in K$, the set $\{t:(i,j,t)\in S\}$ is a non-empty set of the form $[r,r+d]$ for some $r\in T$ and $d\geq d^i(j)$ (every step is executed continuously until completion).*[2]
3. Mutual Exclusion*: For every $i,i'\in\mathcal{J}$, $j,j'\in K$ 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).*

The *length* $|S|$ of a schedule is the maximal t over all $(i,j,t)\in S$. The *optimal job-shop scheduling problem* is to find a schedule of a minimal length. This problem is known to be NP-hard. From the relational definition of schedules one can derive the following commonly used definitions:

1. The *machine allocation function* $\alpha: M\times T\rightarrow\mathcal{J}$ 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* $\beta:\mathcal{J}\times T\rightarrow M$ 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 $M=\{m_1,m_2\}$ and two jobs $J^1=(m_1,4),(m_2,5)$ and $J^2=(m_1,3)$. Two schedules S_1 and S_2 appear in Figure 1. The length of S_1 is 9 and it is the optimal schedule.

We conclude this section with an observation concerning optimal schedules which will be used later. We say that a schedule S exhibits *laziness* at step j of job i if immediately before starting that step there is an interval in which both the job and the corresponding resource are idle. For example in the schedule S of Figure 2, there is a laziness at $(2,1)$. In the job-shop setting, where there are no *logical dependencies* among the jobs, such idling is of no use. Note that a waiting period which is not adjacent to the beginning of the step, e.g. step $(3,1)$ of the same schedule, is not considered as laziness.

Definition 3 (Lazy Schedules). *Let S be a schedule, let i be a job and j a step with $\mu^i(j)=m$ which starts at time t. We say that S exhibits laziness at (i,j) if there is a time $r<t$ such that for every $t'\in[r,t)$, $\beta(i,t')=\perp$ and for every $i'\neq i$, $\beta(i',t')\neq m$. A schedule S is non-lazy if it exhibits no laziness.*

[1] This assumption simplifies the presentation but maintains the inherent complexity.
[2] Note that we allow a job to occupy the machine *after* the step has terminated. This helps in simplifying the timed automata but has no effect on the *optimal* solution.

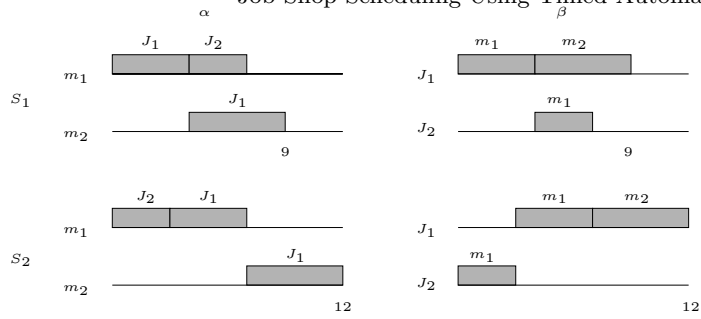

Fig. 1. Two schedule S_1 and S_2 visualized as the machine allocation function α and the task progress function β.

Claim 1 (Non-lazy Optimal Schedules) *Every lazy schedule S can be transformed into a non-lazy schedule $\hat{S}$ with $|\hat{S}| \leq |S|$. Hence every job-shop specification admits an optimal non-lazy schedule.*

Sketch of Proof: The proof is by taking a lazy schedule S and transforming it into a schedule S' were laziness occurs "later". A schedule defines a partial order relation $\prec$ on $\mathcal{J} \times K$ which is generated by the ordering constraints of each job, $(i,j) \prec (i,j+1)$, and by the choices made in the case of conflicts, $(i,j) \prec (i',j')$ if $\mu^i(j) = \mu^{i'}(j')$ and (i,j) precedes (i',j') in S. The laziness elimination procedure picks a lazy step (i,j) which is minimal with respect to $\prec$ and shifts its start time backward to t', to yield a new schedule S', such that $|S'| \leq |S|$. Moreover, the partial order associated with S' is identical to the one induced by S. The laziness at (i,j) is thus eliminated, and this might create new manifestations of laziness at later steps which are eliminated in the subsequent stages of the procedure (see illustration in Figure 2). Let $L(S) \subseteq \mathcal{J} \times K$ be the set of steps that are not preceded by laziness, namely $L(S) = \{(i,j) : \forall (i',j') \preceq (i,j)$ there is no laziness in $(i',j')\}$. Clearly the laziness removal procedure increases $L(S)$ and terminates due to finiteness. ∎

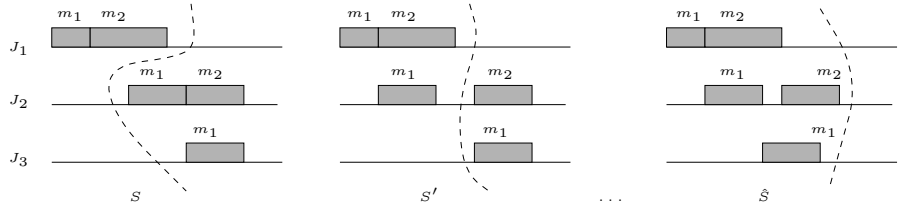

Fig. 2. Removing laziness from a schedule S: first we eliminate laziness at $(2,1)$ and create new ones at $(2,2)$ and $(3,1)$ in S', and those are further removed until a non-lazy schedule $\hat{S}$ is obtained. The dashed line indicates the frontier between $L(S)$ and the rest of the steps.

3 Timed 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.

Definition 4 (Timed Automaton). *A timed automaton is a tuple* $\mathcal{A} = (Q, C, s, f, \Delta)$ *where* Q *is a finite set of states,* C *is a finite set of clocks, 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.*

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 $\mathcal{H}$. 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 $\text{Reset}_\rho : \mathcal{H} \to \mathcal{H}$ defined for every clock valuation $\mathbf{v}$ and every clock variable $c \in C$ as

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

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 $(1, \ldots, 1)$ and $\mathbf{0}$ 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 $\mathbf{v}' = \text{Reset}_\rho(\mathbf{v})$.
- A time step: $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1})$, $t \in \mathbb{R}_+$.

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

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

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

A *lazy run* is a run containing a fragment

$$(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t) \xrightarrow{0} (q', \mathbf{v}')$$

where the transition taken at $(q, \mathbf{v} + t)$ is enabled already at $(q, \mathbf{v} + t')$ for some $t' < t$. In a non-lazy run whenever a transition is taken from a state, it is taken at the earliest possible time. Clearly, from any given configuration there are only finitely many non-lazy continuations and hence for every k there are only finitely many non-lazy runs with k steps.

Next we construct for every job $J = (k, \mu, d)$ a timed automaton with one clock such that for every step j such that $\mu(j) = m$ there will be two states: a

state $\overline{m}$ which indicates that the job is waiting to start the step and a state m indicating that the job is executing the step. Upon entering m the clock is reset to zero, and the automaton can leave the state only after time $d(j)$ has elapsed. Let $\overline{M} = \{\overline{m} : m \in M\}$ and let $\overline{\mu} : K \to \overline{M}$ be an auxiliary function such that $\overline{\mu}(j) = \overline{m}$ whenever $\mu(j) = m$. Note that the clock c is *inactive* at state $\overline{m}$ because it is reset to zero without being tested upon leaving $\overline{m}$.

Definition 5 (Timed Automaton for a Job). *Let $J = (k, \mu, d)$ be job. Its associated timed automaton is $\mathcal{A} = (Q, \{c\}, \Delta, s, f)$ with $Q = P \cup \overline{P} \cup \{f\}$ where $P = \{\mu(1), \ldots \mu(k)\}$, and $\overline{P} = \{\overline{\mu}(1), \ldots, \overline{\mu}(n)\}$. The transition relation Δ consists of the following tuples*

$$\begin{array}{ll} (\overline{\mu}(j), true, \{c\}, \mu(j)) & j = 1..k \\ (\mu(j), c \geq d(j), \emptyset, \overline{\mu}(j+1)) & j = 1..k-1 \\ (\mu(k), c \geq d(k), \emptyset, f) & \end{array}$$

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):

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

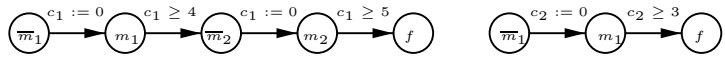

Fig. 3. The automata corresponding to the jobs $J^1 = (m_1, 4), (m_2, 5)$ and $J^2 = (m_1, 3)$.

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 $q = (q^1, \ldots, q^n) \in (M \cup \overline{M} \cup \{f\})^n$ is said to be *conflicting* if it contains two components q^a and q^b such that $q^a = q^b = m \in M$.

Definition 6 (Mutual Exclusion Composition). *Let $\mathcal{J} = \{J^1, \ldots, J^n\}$ be a job-shop specification and let $\mathcal{A}^i = (Q^i, C^i, \Delta^i, s^i, f^i)$ be the automaton corresponding to each J^i. Their mutual exclusion composition is the automaton $\mathcal{A} = (Q, C, \Delta, s, f)$ such that Q is the restriction of $Q^1 \times \ldots Q^n$ to non-conflicting states, $C = C^1 \cup \ldots \cup C^n$, $s = (s^1, \ldots, s^n)$, $f = (f^1, \ldots, f^n)$ and the transition relation Δ contains all the tuples of the form*

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

such that $(q^a, \phi, \rho, p^a) \in \Delta^a$ *for some* a *and the global states* $(q^1, \ldots, q^a, \ldots, q^n)$ *and* $(q^1, \ldots, p^a, \ldots, q^n)$ *are non-conflicting.*

The composition to the two automata of Figure 3 appears in Figure 4.

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 relation 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 2 (Runs and Schedules) *Let* $\mathcal{A}$ *be the automaton generated for the job-shop specification* $\mathcal{J}$ *according to Definitions 1 and 2. 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* $S_\xi = S$*. Moreover, if* S *is non-lazy so is* ξ*.*

Note that non-laziness of the run does not imply non-laziness of the schedule.

Corollary 1 (Job-Shop Scheduling and Timed Automata). *The optimal job-shop scheduling problem can be reduced to the problem of finding the shortest non-lazy path in a timed automaton.*

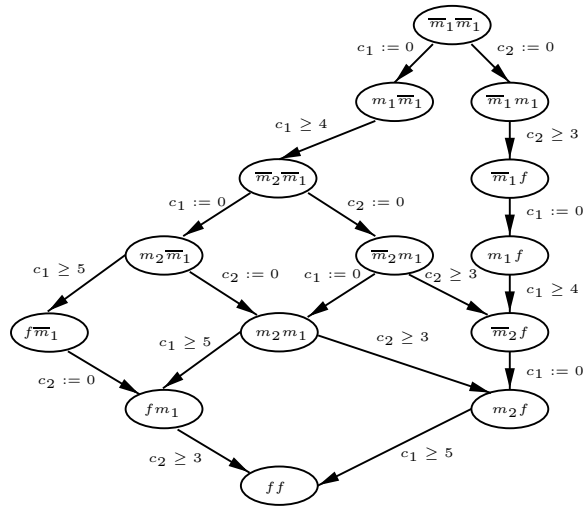

Fig. 4. The global timed automaton for the two jobs.

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

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

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

Some words are in order to describe the structure of the job-shop timed automaton. First, it is an *acyclic* automaton and its state-space admits a natural partial-order. It can be partitioned into levels according to the number of discrete transitions from s to the state. All transitions indicate either a component moving from an active to an inactive state (these are guarded by conditions of the form $c_i \geq d$), or a component moving into an active state (these are labeled by resets $c_i := 0$). There are no staying conditions (invariants) and the automaton can stay forever in any given state. Recall that in a timed automaton, the transition graph might be misleading, because two or more transitions entering *the same* discrete state, e.g. transitions to (m_2, 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 the initial state represent the decision to whom to give first the resource m_1. On the other hand some duplication of paths are just artifacts due to interleaving, for example, the two paths outgoing from $(\overline{m}_2, \overline{m}_1)$ to (m_2, m_1) are practically equivalent.

Another useful observation is that from every 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 $\mathcal{J}$.

4 Shortest Paths in Timed Automata

In this section we describe how the symbolic forward reachability algorithm of Kronos is adapted to find a shortest path in a job-shop timed automaton. Although Corollary 1 allows us to use enumerative methods in the case of deterministic job-shop problems, we start with algorithms that do not take advantage of non-laziness, both for the completeness of the presentation and as a preparation for more complex scheduling problems where non-laziness results do not hold. Standard shortest-path algorithms operate on *discrete* graphs with numerical weights assigned to their edges. The transition graphs of timed automata are non-countable and hence not amenable to enumerative algorithms.[3]

We recall some commonly-used definitions concerning timed automata. A *zone* is a subset of $\mathcal{H}$ 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 $\{(q, \mathbf{z}) : \mathbf{z} \in Z\}$. Symbolic states are closed under the following operations:

[3] One can, of course, discretize time into unit steps but this will cause an enormous increase in the state-space of the automaton.

- The *time successor* of (q, Z) is the set of configurations which are reachable from (q, Z) by letting time progress:

$$Post^t(q, Z) = \{(q, \mathbf{z} + r\mathbf{1}) : \mathbf{z} \in Z, r \geq 0\}.$$

  We say that (q, Z) is *time-closed* if $(q, Z) = Post^t(q, Z)$.
- The *δ-transition successor* of (q, Z) is the set of configurations reachable from (q, Z) by taking the transition $\delta = (q, \phi, \rho, q') \in \Delta$:

$$Post^\delta(q, Z) = \{(q', \mathrm{Reset}_\rho(\mathbf{z})) : \mathbf{z} \in Z \cap \phi\}.$$

- The *δ-successor* of a time-closed symbolic state (q, Z) is the set of configurations reachable by a δ-transition followed by passage of time:

$$Succ^\delta(q, Z) = Post^t(Post^\delta(q, Z)).$$

- The *successors* of (q, Z) is the set of all its δ-successors:

$$Succ(q, Z) = \bigcup_{\delta \in \Delta} (Succ^\delta(q, Z)).$$

To compute all the reachable configurations of the job-shop automaton we use a variant of the standard forward reachability algorithm for timed automata, specialized for acyclic graphs.

Algorithm 1 (Forward Reachability for Acyclic Timed Automata)
Waiting:=$\{Post^t(s, \mathbf{0})\}$*;*
while *Waiting* $\neq \emptyset$*;* **do**
  Pick $(q, Z) \in$ *Waiting;*
  For every $(q', Z') \in Succ(q, Z)$*;*
    Insert (q', Z') *into Waiting;*
  Remove (q, Z) *from Waiting*
end

This algorithm solves the reachability problem for timed automata — a trivial problem for job-shop automata since all complete runs lead to f. Its adaptation for finding *shortest paths* is rather straightforward. All we do is to use a clock-space $\mathcal{H}'$ which is the clock-space of $\mathcal{A}$ augmented with an additional clock c_{n+1} which is never reset. For any symbolic state (q, Z) reachable in the modified automaton $\mathcal{A}'$, if $(v_1, \ldots, v_n, v_{n+1}) \in Z$ then $(q, (v_1, \ldots, v_n))$ is reachable in $\mathcal{A}$ within any time $t \geq v_{n+1}$. Consequently, the length of the shortest run from the initial state to q via the (qualitative) path which generated (q, Z) is

$$G(q, Z) = \min\{v_{n+1} : (v_1, \ldots, v_n, v_{n+1}) \in Z\}$$

and the length of the optimal schedule is

$$\min\{G(f, Z) : (f, Z) \text{ is reachable in } \mathcal{A}'\}.$$

Hence, running Algorithm 1 on $\mathcal{A}'$ is guaranteed to find the minimal schedule.

The rest of the section is devoted to several improvements of this algorithm, whose naïve implementation will generate a symbolic state for almost every node in the unfolding of the automaton. Experimental results appear in Table 1.

Inclusion Test: This is a common method used in Kronos for reducing the number of symbolic states in verification. It is based on the fact that $Z \subseteq Z'$ implies $Succ^{\delta}(q, Z) \subseteq Succ^{\delta}(q, Z')$ for every $\delta \in \Delta$. Hence, whenever a new symbolic state (q, Z) is generated, it is compared with any other (q, Z') in the waiting list: if $Z \subseteq Z'$ then (q, Z) is not inserted and if $Z' \subseteq Z$, (q, Z') is removed from the list. Note that allowing the automaton to stay indefinitely in any state makes the explored zones "upward-closed" with respect to absolute time and increases significantly the effectiveness of the inclusion test.

Domination Test: The inclusion test removes a symbolic state only if *all* its successors are included in those of another symbolic state. Since we are interested only in optimal runs, we can apply stronger reductions that do not preserve all runs, but still preserve the optimal ones. As an example consider an automaton with two paths leading from the initial state to a state q, one by first resetting c_1 and then c_2 and one in the reverse order. The zones reachable via these paths are $Z_1 = c_3 \geq c_1 \geq c_2 \geq 0$ and $Z_2 = c_3 \geq c_2 \geq c_1 \geq 0$, where c_3 is the additional clock which measures absolute time. These zones are incomparable with respect to inclusion, however, for every t they share a "maximal" point (t, t, t) which corresponds to the respective non-lazy runs along each of the paths. Hence it is sufficient to explore only one of the symbolic states (q, Z_1) and (q, Z_2).

Let $(q, (\mathbf{v}, t))$ and $(q, (\mathbf{v}', t'))$ be two reachable configurations in $Q \times \mathcal{H}'$. We say that $(\mathbf{v}, t)$ *dominates* $(\mathbf{v}', t')$ if $t \leq t'$ and $\mathbf{v} \geq \mathbf{v}'$. Intuitively this means that $(q, \mathbf{v})$ was reached not later than $(q, \mathbf{v}')$ and with larger clock values, which implies that steps active at q started earlier along the run to $(q, \mathbf{v})$ and hence can terminate earlier. It can be shown that for every reachable symbolic state (q, Z), Z contains an optimal point $(\mathbf{v}^*, t^*)$ dominating every other point in Z. This point, which is reachable via a non-lazy run, can be computed by letting $t^* = G(q, Z)$ (earliest arrival time) and $\mathbf{v}^* = (v_1^*, \ldots, v_n^*)$ where for every i,

$$v_i^* = \max\{v_i : (v_1, \ldots, v_i, \ldots, v_n, t^*) \in Z\}.$$

We say that Z_1 dominates Z_2 if $(\mathbf{v}_1^*, t_1^*)$ dominates $(\mathbf{v}_2^*, t_2^*)$. We apply the domination test in the same manner as the inclusion test to obtain a further reduction of the number of symbolic states explored.

Best-First Search: The next improvement consists in using a more intelligent search order than breadth-first. To this end we define an evaluation function $E : Q \times \mathcal{H}' \to \mathbb{R}_+$ for estimating the quality of configurations and symbolic states:

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

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. The extension of this function to zones is $E(q, Z) = E(q, (\mathbf{v}^*, t^*))$. It is not hard to see that $E(q, Z)$ gives a lower bound on the length of every complete run which passes through (q, Z).

Table 1. The results for n jobs with 4 tasks. Columns #j, #ds and #tree show, respectively, the number of jobs, the number of discrete states in the automaton and the number of different reachable symbolic states (which is close to the number of nodes in the unfolding of the automaton into a tree). The rest of the table shows the performance, in terms of the number of explored symbolic states and time (in seconds), of algorithms employing, progressively, the inclusion test, the domination test, and the best-first search (m.o. indicates memory overflow).

Problem size			Inclusion		Domination		Best-first	
#j	#ds	#tree	#s	time	#s	time	#s	time
2	77	632	212	1	100	1	38	1
3	629	67298	5469	2	1143	1	384	1
4	4929	279146	159994	126	11383	2	1561	1
5	37225	m.o.	m.o.	m.o.	116975	88	2810	1
6	272125	m.o.	m.o.	m.o.	1105981	4791	32423	6

The modified algorithm now orders the waiting list of symbolic states according to their evaluation (and applies the inclusion and domination tests upon insertion to the list). This algorithm 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:=$\{Post^t(s, \mathbf{0})\}$*;*
Best:=∞
(q, Z)*:= first in Waiting;*
while *Best* $> E(q, Z)$
do
  For every $(q', Z') \in Succ(q, Z)$*;*
    if $q' = f$ **then**
      *Best:=*min$\{$*Best*,$E(q', Z')\}$
    else
      Insert (q', Z') *into Waiting;*
  Remove (q, Z) *from Waiting*
  (q, Z)*:= first in Waiting;*
end

We have implemented these techniques into Kronos and tested them first on a family of problems consisting of n jobs, $n = 2, \ldots, 6$, each with 4 steps.[4] We also make use of Kronos' capability to handle zones of varying dimensionality, were only active clocks are considered [DY96]. The results, obtained on a Pentium P3, 666 MHz under Linux, with memory restricted to 512MB, are depicted in Table 1. One can see that the number of symbolic states explored by the best-first

[4] The problems can be found in `http://www-verimag.imag.fr/~maler/jobshop`.

algorithm is smaller than the number of discrete states in the timed automaton. Nevertheless the combinatorial nature of the problem cannot be avoided.

Points instead of Zones: Following Corollary 1, an optimal run can be found among the non-lazy runs and the search can be restricted to explore only such runs. This search can be performed without using zones, but rather using single points in the clock space (which are exactly the dominating points of the reachable zones). This reduces significantly memory usage ($O(n)$ per symbolic state instead of $O(n^2)$) and simplifies the operations.

Sub-optimal Solutions: 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) symbolic states, generate their successors but explore only the best w among them, and so on. The number w is the main parameter of this technique, and although the number of explored states grows monotonically with w, the quality of the solution does not — sometimes the solution found with a smaller w is better than the one found with a larger w.

In order to test this heuristics we took 10 problems among the most notorious job-shop scheduling problems.[5] Note that these are pathological problems with a large variability in step durations, constructed to demonstrate the hardness of job-shop scheduling. For each of these problems we have applied our algorithms for different choices of w, both forward and backward. In Table 2 we compare our best results on these problems with the results reported in Table 15 of the recent survey [JM99], where the the 18 best-known methods were compared. In order to appreciate the difficulty, we also compare our results with the best results among 3000 randomly-generated solutions for each of the problems.

5 Related Work

This work can be viewed in the context of extending verification methodology in two orthogonal directions: from *verification* to *synthesis* and from *qualitative* to *quantitative* evaluation of behaviors. In verification we check the existence of certain paths in a *given* automaton, while in synthesis we have an automaton in which not all design choices have been made and we can remove transitions (and hence make the necessary choices) so that a property is satisfied. If we add a quantitative dimension (in this case, the duration of the path), verification is transformed to the evaluation of the worst performance measure over all paths, and synthesis into the restriction of the automaton to one or more optimal paths.

The idea of applying synthesis to timed automata was first explored in [WH92]. An algorithm for safety controller synthesis for timed automata, based on operation on zones was first reported in [MPS95] and later in [AMP95], where an example of a simple scheduler was given, and in [AMPS98]. This algorithm is a generalization of the verification algorithm for timed automata [HNSY94,ACD93] used in Kronos [Y97,BDM+98]. In these and other works on

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

Table 2. The results for 10 hard problems using the bounded width heuristic. The first three columns give the problem name, no. of jobs and no. of machines (and steps). Our results (time in seconds, the length of the best schedule found and its deviation from the optimum) appear next, followed by the best out of 3000 randomly-generated solutions and by the best known result for each problem.

problem			Kronos			Rand		Opt
name	#j	#m	time	length	deviation	length	deviation	length
FT10	10	10	13	982	5.59 %	1761	89.35 %	930
LA02	10	5	1	655	0.00 %	1059	61.68 %	655
LA19	10	10	12	885	5.11 %	1612	91.45 %	842
LA21	10	15	178	1114	6.50 %	2339	123.61 %	1046
LA24	10	15	186	992	5.98 %	2100	124.00 %	936
LA25	10	15	180	1041	6.55 %	2209	126.10 %	977
LA27	10	20	6	1343	8.74 %	2809	127.45 %	1235
LA29	10	20	193	1295	12.41 %	2713	135.50 %	1152
LA36	15	15	16	1391	9.70 %	2967	133.90 %	1268
LA37	15	15	72	1489	6.59 %	3188	128.20 %	1397

treating scheduling problems as synthesis problems for timed automata, such as [AGP99], the emphasis was on yes/no properties, such as the existence of a feasible schedule, in the presence of an uncontrolled adversary.

A transition toward quantitative evaluation criteria was made already in [CY91] where timed automata were used to compute bounds on delays in real-time systems and in [CCM$^+$94] where variants of shortest-path problems were solved on a timed model much weaker than timed automata. To our knowledge, the first quantitative synthesis work on timed automata was [AM99] in which the following problem has been solved: "given a timed automaton with both controlled and uncontrolled transitions, restrict the automaton in a way that from each configuration the worst-case time to reach a target state is minimal". If there is no adversary, this problem corresponds to finding the shortest path. Due to the presence of an adversary, the solution in [AM99] employs backward-computation (dynamic programming), i.e. an iterative computation of a function $h : Q \times \mathcal{H} \rightarrow \mathbb{R}_+$ such that $h(q, \mathbf{v})$ indicates the minimal time for reaching the target state from $(q, \mathbf{v})$. The implementation of the forward algorithm used in this paper can be viewed as iterating with a function h such that $h(q, \mathbf{v})$ indicates the minimal time to reach $(q, \mathbf{v})$ from the initial state. The reachable states in the augmented clock-space are nothing but a relational representation of h.

Around the same time, in the framework of the VHS (Verification of Hybrid systems) project, a simplified model of a steel plant was presented as a case-study [BS99]. The model had more features than the job-shop scheduling problem such as upper-bounds on the time between steps, transportation problems, etc. A. Fehnker proposed a timed automaton model of this plant from which feasible schedules could be extracted [F99]. This work inspired us to find a systematic connection between classical scheduling problems and timed automata [M99],

upon which this paper is based. Another work in this direction was concerned with another VHS case-study, a cyclic experimental batch plant at Dortmund for which an optimal dynamic scheduler was derived in [NY00].

The idea of using heuristic search is useful not only for shortest-path problems but for verification of timed automata (and verification in general) where some evaluation function can guide the search toward the target goal. These possibilities were investigated recently in [BFH+01a] on several classes of examples, including job-shop scheduling problems, where various search procedures and heuristics were explored and compared.

In [NTY00] it was shown that in order to find shortest paths in a timed automaton, it is sufficient to look at acyclic sequences of symbolic states (a fact that we do not need due to the acyclicity of job-shop automata) and an algorithms based on forward reachability was introduced. A recent generalization of the shortest path problem was investigated by [BFH+01b] and [ATP01]. In this model there is a *different* price for staying in any state and the total cost associated with the run progresses in different slopes along the path. It has been proved that the problem of finding the path with the minimal cost is computable.

6 Conclusions

We have suggested a novel application of timed automata, namely for solving job-shop scheduling problems. We believe that the insight gained from this point of view will contribute both to scheduling and to the study of timed automata. We have demonstrated that the performance of automata-based methods is not inferior to other methods developed within the last three decades. There are still many potential improvements to be explored such as the application of partial-order methods, more symbolic representation of the discrete states, new heuristics, etc. The most interesting challenge is to adapt these techniques for more complex scheduling situation such as those involving uncertainty or logical dependencies among tasks.

Acknowledgment

We are most grateful to Eugene Asarin and Marius Bozga for their help in the theoretical and practical aspects of this work. Thanks also to K. Larsen, S. Engell and anonymous referees for reading previous versions of this paper.

References

ATP01. R. Alur, S. La Torre and G.J. Pappas, Optimal Paths in Weighted Timed Automata, *Proc. HSCC'01*, 49-64, LNCS 2034, 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.

ACD93. R. Alur, C. Courcoubetis, and D.L. Dill, Model Checking in Dense Real Time, *Information and Computation* 104, 2-34, 1993.

AD94. R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183-235, 1994.

AM99. E. Asarin and O. Maler, As Soon as Possible: Time Optimal Control for Timed Automata, *Proc. HSCC'99*, 19-30, LNCS 1569, Springer, 1999.

AMP95. E. Asarin, O. Maler and A. Pnueli, Symbolic Controller Synthesis for Discrete and Timed Systems, *Hybrid Systems II*, LNCS 999, Springer, 1995.

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+01a. 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.

BFH+01b. G. Behrmann, A. Fehnker T.S. Hune, K.G. Larsen, P. Pettersson, J. Romijn and F.W. Vaandrager, Minimum-Cost Reachability for Linearly Priced Timed Automata, *Proc. HSCC'01*, 147-161, LNCS 2034, Springer 2001.

BS99. R. Boel and G. Stremersch, VHS case study 5: Modelling and Verification of Scheduling for Steel Plant at SIDMAR, Draft, 1999.

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.

CCM+94. S Campos, E. Clarke, W. Marrero, M. Minea and H. Hiraishi, Computing Quantitative Characteristics of Finite-state Real-time Systems, *Proc. RTSS'94*, IEEE, 1994.

CY91. C. Courcoubetis and M. Yannakakis, Minimum and Maximum Delay Problems in Real-time Systems, *Proc. CAV'91*, LNCS 575, 399-409, Springer, 1991.

DY96. C. Daws and S. Yovine, Reducing the Number of Clock Variables of Timed Automata, *Proc. RTSS'96*, 73-81, IEEE, 1996.

F99. A. Fehnker, Scheduling a Steel Plant with Timed Automata, *Proc. RTCSA'99*, 1999.

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

JM99. A.S. Jain and S. Meeran, Deterministic Job-Shop Scheduling: Past, Present and Future, *European Journal of Operational Research* 113, 390-434, 1999.

M99. O. Maler, On the Problem of Task Scheduling, Draft, February 1999.

MPS95. O. Maler, A. Pnueli and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems, *Proc. STACS'95,* 229-242, LNCS 900, Springer, 1995.

NTY00. P. Niebert, S. Tripakis S. Yovine, Minimum-Time Reachability for Timed Automata, *IEEE Mediteranean Control Conference*, 2000.

NY00. P. Niebert and S. Yovine, Computing Optimal Operation Schemes for Chemical Plants in Multi-batch Mode, *Proc. HSCC'2000*, 338-351, LNCS 1790, Springer, 2000.

WH92. H. Wong-Toi and G. Hoffmann, The Control of Dense Real-Time Discrete Event Systems, Technical report STAN-CS-92-1411, Stanford University, 1992.

Y97. S. Yovine, Kronos: A Verification Tool for Real-time Systems, *Int. J. of Software Tools for Technology Transfer* 1, 1997.