Task Graph Scheduling using Timed Automata*

Yasmina Abdeddaïm, Abdelkarim Kerbaa and Oded Maler

VERIMAG, Centre Equation, 2, av. de Vignate, 38610 Gières, France
Yasmina.Abdeddaim@imag.fr Abdelkarim.Kerbaa@imag.fr Oded.Maler@imag.fr

Abstract

In this paper we develop a methodology for treating the problem of scheduling partially-ordered tasks on parallel machines. Our framework is based on the timed automaton model, originally developed for verification of real-time programs and digital circuits and more recently adapted for solving time-optimal scheduling problems. In this framework, the scheduling problem admits a state-space representation and an optimal schedule corresponds to a shortest path in the timed automaton. We check our implementation on numerous benchmarks and show how release times and deadlines can be easily incorporated into the model.

1. Introduction

Timed Automata (TA) were introduced in [AD94] as a model for real-time systems, that is discrete systems whose behaviors are embedded in the real-time axis, where discrete transitions interact with the passage of time. Using this model one can express very naturally timing constraints such as the duration of certain processes or the separation time between two events. In addition to the expressive power, the reachability problem for timed automata is decidable and hence they can be subject to algorithmic verification (model-checking) [HNSY94], a fact exploited in TA verification tools such as Kronos [Y97] and Uppaal [LPW97b].

Although the initial motivation for TA models was *verification* of *qualitative* properties, for example, to show that all behaviors of a *given* timed automaton never reach an undesired state, there is a growing interest in recent years to extend the applicability of TA from verification to *synthesis* and from qualitative to *quantitative* evaluation of behaviors. In particular, the synthesis of schedulers that satisfy timing constraints or that are optimal in some sense has attracted a lot of attention [MPS95, AMP95, AM99, AGP99, NTY00, NY00, BFH+01].

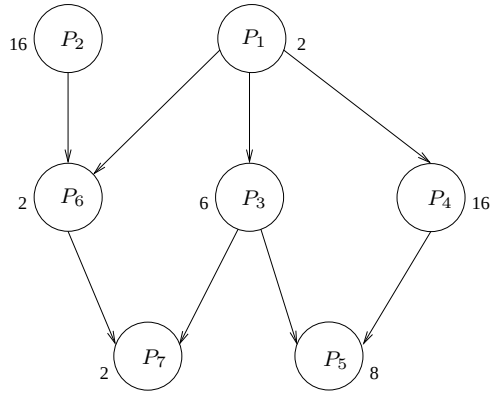
In [AM01] we have initiated a research programme intended to re-formulate in a systematic manner various scheduling problems using TA. In this framework the runs of the timed automaton correspond to feasible schedules and finding a time-optimal schedule amounts to finding the shortest path (in terms 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 the problem known as *task graph scheduling*, where one has to schedule tasks on a limited number of identical machines, while respecting some precedence constraints [C76]. A task graph is thus a set of partially-ordered tasks, with an integer number (the task duration) associated with every node (see Figure 1). A task can be executed only if all its predecessors in this graph have completed. This is a fundamental problem in parallel computation for which numerous algorithms have been proposed (see [KI99] for a comparison of these algorithms). The job shop problem can be viewed as a particular case where this graph is a set of linear chains, each chain representing the precedence relation in one job (however in that problem the machines are not identical).

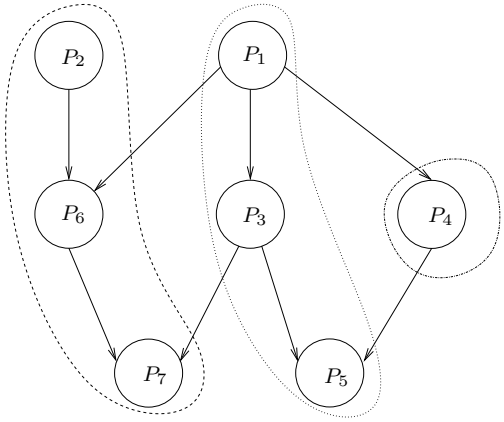
The rest of the paper is organized as follows. In section 2 we define the problem. Section 3 gives the basics of timed automata. In section 4 we give two translation schemes for transforming the scheduling problem into a timed automaton, and mention briefly the algorithms used for the shortest path problems. Experimental results with the implementa-

*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.



(a)



(b)

Figure 1. (a) A task graph. (b) A chain covering.

tion are described in section 5, followed by possible extensions of the model, some of which (deadlines and release times) already implemented.

2. The Problem

Definition 1 (Task Graph) *A task graph is a triple $G = (\mathcal{P}, \prec, d)$ such that $\mathcal{P} = \{P_1, \ldots, P_m\}$ is a set of m tasks, $\prec$ is a partial-order relation on $\mathcal{P}$ and $d : \mathcal{P} \rightarrow \mathbb{N}$ is a function which assigns a duration to each task.*

We denote by $\Pi(P)$ the set of immediate predecessors of P. Given a set $\{M_1, \ldots, M_n\}$ of n parallel identical machines, we need to find the schedule that minimizes the total execution time and respects the following conditions: 1) A task can be executed only if all its predecessors have completed; 2) Each machine can process at most one task at a time; 3) Tasks cannot be preempted.

Definition 2 (Feasible and Optimal Schedules) *A feasible schedule for a task graph $G = (\mathcal{P}, \prec, d)$ and n machines is a function $st : \mathcal{P} \rightarrow \mathbb{R}_+$ (indicating the start time of each task) satisfying:*

1. *For every $P \in \mathcal{P}$ $st(P) \geq \max_{P' \in \Pi(P)} st(P') + d(P')$.*
2. *Every $t \in \mathbb{R}_+$ belong to at most n of set of intervals of the form $\{[st(P), st(P) + d(P)] : P \in \mathcal{P}\}$.*

The length of the schedule is $\max\{st(P) + d(P) : P \in \mathcal{P}\}$. An optimal schedule is a schedule whose length is minimal.

Note that in this paper we treat problems without communication costs so that the identity of the machines on which a task is executed is not important.

If we have as many machines as we want, the optimal schedule is obtained by starting every task as soon as its predecessors terminate. In that case the length of the optimal schedule is the length of the maximal path from a minimal to a maximal element of $(\mathcal{P}, \prec)$. The schedule of Figure 2-(a) is an optimal schedule for the graph of Figure 1-(a) when the number of machines is unlimited. Notice that 3 machines are sufficient to construct this schedule, because no more than 3 tasks are enabled simultaneously, see Figure 2-(b).

On the other hand, if we have only 2 machines the number of enabled tasks may exceed the number of available machines and the conflict should be resolved by the scheduler. We can see in schedules S_1 and S_2 of Figure 3 that at $t = 2$, P_2 is already occupying M_1 where both P_3 and P_4 become enabled. In S_1 we give the remaining machine to P_3, and in S_2 we give it to P_4. Unlike the case of infinitely many machines, an optimal schedule may be obtained by

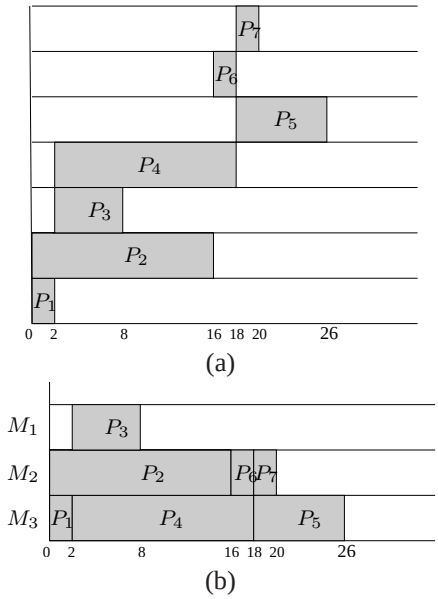


Figure 2. (a) An optimal schedule of the task graph of Figure 1 when the number of machine is unlimited. (b) An optimal schedule on 3 machines.

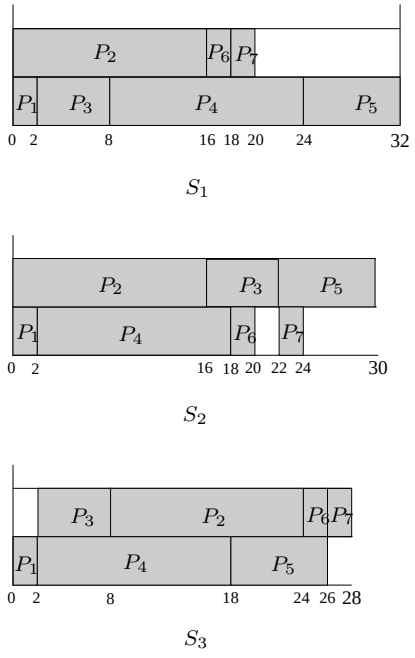


Figure 3. Three feasible schedules of the task graph of Figure 1 on 2 machines.

choosing at some point not to execute an enabled task. For example, schedule S_3 achieves the optimum while not starting task P_2 immediately although it is enabled at $t = 0$.

3. Timed Automata

Timed automata [AD94] 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 as conditions for the enabledness of transitions. Hence they are ideal for describing concurrent time-dependent behaviors.

Definition 3 (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 $\mathrm{Reset}_\rho : \mathcal{H} \to \mathcal{H}$ 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 other clocks 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 $\mathbf{v}' = \mathrm{Reset}_\rho(\mathbf{v})$.
- 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 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.

4. Modeling with Timed Automata

Our goal is to model the task graph scheduling problem using a timed automaton so that every run corresponds to

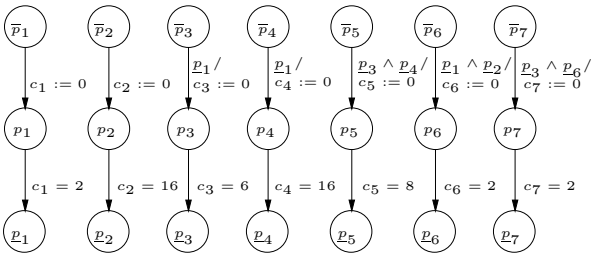


Figure 4. The automata for the task graph of Figure 1

a feasible schedule and the shortest run gives the optimal schedule. In order to compose timed automata we slightly modify the definition of the transition relation Δ to include tuples of the form (q, ϕ, ρ, q') where ϕ is either, as before, a combination of clock inequalities, or a formula specifying states of other automata.

For every task P we build a 3-state automaton with one clock c and a set of states $Q = \{\overline{p}, p, \underline{p}\}$ where $\overline{p}$ is the waiting state before the task starts, p is the active state where the task executes and $\underline{p}$ is a final state indicating that the task has terminated. The transition from $\overline{p}$ to p resets the clock and can be taken only if all the automata corresponding to the tasks in $\Pi(P)$ are in their final states. The transition from p to $\underline{p}$ is taken when $c = d(p)$. The automata for the task graph of Figure 1 appear in Figure 4.

Definition 4 (Timed Automaton for a Task)
Let $G = (\mathcal{P}, \prec, d)$ be a task graph. For every task $P \in \mathcal{P}$ its associated timed automaton is $\mathcal{A} = (Q, \{c\}, \Delta, s, f)$ with $Q = \{p, \overline{p}, \underline{p}\}$ where the initial state is $\overline{p}$ and the final state is $\underline{p}$. The transition relation Δ consists of the two transitions:

$$start: \quad (\overline{p}, \bigwedge_{P \in \Pi(P)} \underline{p}', \{c\}, p)$$

and

$$end: \quad (p, c = d(p), \emptyset, \underline{p})$$

The global automaton representing all the feasible schedules can be obtained as a composition of the individual task automata, a composition that takes care that the *start* transitions do not violate the precedence and resource constraints. This is achieved by allowing such a transition in a global state only if its guard is satisfied in that state and the number of active tasks (tasks in a p-state) in the resulting state does not exceed the number of machines.

Although in terms of the number of reachable global states this automaton is as good as we can get, it has some features which make its analysis impractical and which can be improved. The global states are m-tuples where m can be very large and the number of clocks is m as well. In reality, however, even when infinitely many machines are available, the number of tasks that can be active simultaneously is bounded by the *width* of the task graph, the maximal number of elements incomparable with respect to $\prec$.

Definition 5 (Chain) *A chain in a partially-ordered set $(\mathcal{P}, \prec)$ is a subset $\mathcal{P}'$ of $\mathcal{P}$ such that for every $P, P' \in \mathcal{P}'$ either $P \prec P'$ or $P' \prec P$.*

Definition 6 (Chain Cover) *A chain covering of a partially-ordered set $(\mathcal{P}, \prec)$ is a set of chains $\mathcal{H} = \{H_1, \ldots H_k\}$ satisfying*

1. *Each H_i is a linearly ordered subset of P.*
2. *$H_i \cap H_j = \emptyset$ for every $i \neq j$.*
3. *$\bigcup_{i \leq k} H_i = \mathcal{P}$*

An example of a chain cover for our task graph appears in Figure 1-(b). It is worth mentioning that chain covers are related to the width of a partial order via Dilworth's theorem [D50].

Theorem 1 (Dilworth) *The width of a partial order is equal to the minimal number of chains needed to cover it.*

The *external predecessors* of a task $P \in H_i$ are the predecessors of P outside its chain, i.e.

$$\Pi'(P) = \Pi(P) \cap (\mathcal{P} - H_i).$$

Given a chain $H = P_1 \prec P_2 \prec \cdots \prec P_k$ its automaton consists of a pair of state $\{\overline{p}_i, p_i\}$ for every P_i and a final state f. The start transition from $\overline{p_i}$ to p_i is enabled if for every $j \neq i$ and $P' \in \Pi'(P) \cap H_j$, the automaton for the chain H_j is in a state beyond p'. We denote this condition by:

$$\bigwedge_{P' \in \Pi'(P)} > p'.$$

After having constructed an automaton for every chain (see Figure 5) we compose these together, while avoiding global states when the number of active chains is larger than the number of machines. A global state $q = (q^1, \ldots, q^n)$ of the product automaton is said to be *conflicting* if it contains more than n active components.

Definition 7 (Mutual Exclusion Composition)

Let $\mathcal{H} = \{H_1, \ldots, H_n\}$ be a chain cover of a task graph and let $\mathcal{A}^i = (Q^i, C^i, \Delta^i, s^i, f^i)$ be the automaton corresponding to each H_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, $C = C^1 \cup \ldots \cup C^n$, $s = (s^1, \ldots, s^n)$, $f = (f^1, \ldots, f^n)$

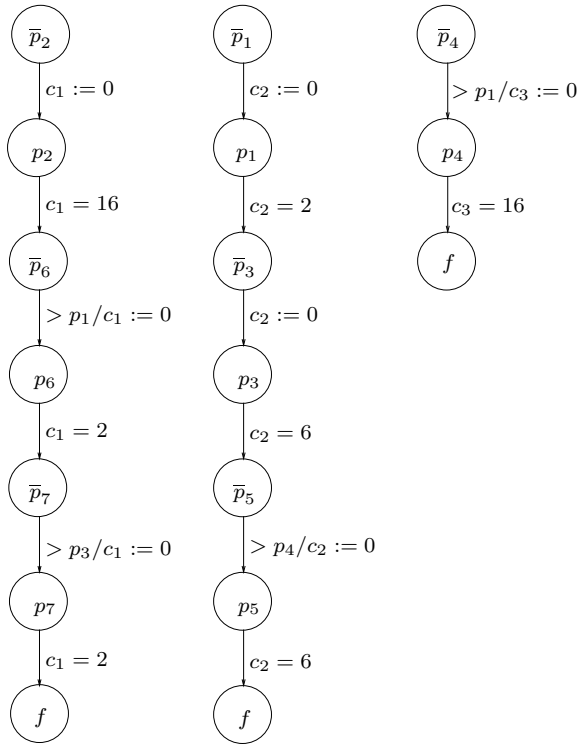


Figure 5. The automata for the chain cover of Figure 1-(b)

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, the guard ϕ is satisfied in $(q^1, \ldots, q^a, \ldots, q^n)$ and both $(q^1, \ldots, q^a, \ldots, q^n)$ and $(q^1, \ldots, p^a, \ldots, q^n)$ are non-conflicting.

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 correspondence between run and schedules is straightforward (see [AM01]): From every complete run ξ one can derive a schedule st_ξ such that $st_\xi(P)$ is the time where the automaton for the chain containing P makes a start transition to state p. Likewise one can generate from a feasible schedule st a complete run ξ_{st} of the same length.

Corollary 2 (Task Graph Scheduling and Timed Automata) *The optimal task graph scheduling problem can be reduced to the problem of finding the shortest path in a timed automaton.*

The timed automaton of Figure 6 represents a part of the timed automaton obtained by composing the automata of Figure 5 when there are 2 machines. This automaton has only 3 clocks (the number of chains in the cover). In the initial state, where tasks P_2, P_1 and P_4 are waiting, there are only two possible successors, to start P_2 (state $(p_2\ \overline{p}_1\ \overline{p}_4)$) or to start P_1 (state $(\overline{p}_2\ p_1\ \overline{p}_4)$). The transition to the state $(\overline{p}_2\ \overline{p}_1\ p_4)$ is disabled because task P_1 has not terminated. No start transition can be taken from $(p_2, p_3, \overline{p}_4)$ because all the machines are occupied in this state. Schedules S_2 and S_3 of Figure 3 correspond, respectively, to the following two runs of the automaton:

$$\begin{aligned}
&(\overline{p}_2, \overline{p}_1, \overline{p}_4, \bot, \bot, \bot) \xrightarrow{0} (\overline{p}_2, p_1, \overline{p}_4, \bot, 0, \bot) \xrightarrow{0} (p_2, p_1, \overline{p}_4, 0, 0, \bot)\\
&\xrightarrow{2} (p_2, p_1, \overline{p}_4, 2, 2, \bot) \xrightarrow{0} (p_2, \overline{p}_3, \overline{p}_4, 2, \bot, \bot) \xrightarrow{0} (p_2, \overline{p}_3, p_4, 2, \bot, 0)\\
&\xrightarrow{0} (p_2, \overline{p}_3, p_4, 2, \bot, 0) \xrightarrow{14} (p_2, \overline{p}_3, p_4, 16, \bot, 14) \xrightarrow{0} (\overline{p}_6, \overline{p}_3, p_4, \bot, \bot, 14)\\
&\xrightarrow{0} (\overline{p}_6, p_3, p_4, \bot, 0, 14) \xrightarrow{2} (\overline{p}_6, p_3, p_4, \bot, 2, 16) \xrightarrow{0} (\overline{p}_6, p_3, f, \bot, 2, \bot)\\
&\xrightarrow{0} (p_6, p_3, f, 0, 2, \bot) \xrightarrow{2} (p_6, p_3, f, 2, 4, \bot) \xrightarrow{0} (\overline{p}_7, p_3, f, \bot, 4, \bot)\\
&\xrightarrow{2} (\overline{p}_7, p_3, f, \bot, 6, \bot) \xrightarrow{0} (\overline{p}_7, \overline{p}_5, f, \bot, \bot, \bot) \xrightarrow{0} (p_7, \overline{p}_5, f, 0, \bot, \bot)\\
&\xrightarrow{0} (p_7, p_5, f, 0, 0, \bot) \xrightarrow{0} (p_7, p_5, f, 2, 0, \bot) \xrightarrow{2} (f, p_5, f, \bot, 2, \bot)\\
&\xrightarrow{6} (f, p_5, f, \bot, 8, \bot) \xrightarrow{0} (f, f, f, \bot, \bot, \bot)
\end{aligned}$$

$$\begin{aligned}
&(\overline{p}_2, \overline{p}_1, \overline{p}_4, \bot, \bot, \bot) \xrightarrow{0} (\overline{p}_2, p_1, \overline{p}_4, \bot, 0, \bot) \xrightarrow{2} (\overline{p}_2, p_1, \overline{p}_4, \bot, 2, \bot)\\
&\xrightarrow{0} (\overline{p}_2, \overline{p}_3, \overline{p}_4, \bot, \bot, \bot) \xrightarrow{0} (\overline{p}_2, p_3, \overline{p}_4, \bot, 0, \bot) \xrightarrow{0} (\overline{p}_2, p_3, p_4, \bot, 0, 0)\\
&\xrightarrow{6} (\overline{p}_2, p_3, p_4, \bot, 6, 6) \xrightarrow{0} (\overline{p}_2, \overline{p}_5, p_4, \bot, \bot, 6) \xrightarrow{0} (p_2, \overline{p}_5, p_4, 0, \bot, 6)\\
&\xrightarrow{10} (p_2, \overline{p}_5, p_4, 10, \bot, 16) \xrightarrow{0} (p_2, \overline{p}_5, f, 10, \bot, \bot) \xrightarrow{0} (p_2, p_5, f, 10, 0, \bot)\\
&\xrightarrow{6} (p_2, p_5, f, 16, 6, \bot) \xrightarrow{0} (\overline{p}_6, p_5, f, \bot, 6, \bot) \xrightarrow{0} (p_6, p_5, f, 0, 6, \bot)\\
&\xrightarrow{2} (p_6, p_5, f, 2, 8, \bot) \xrightarrow{0} (\overline{p}_7, p_5, f, \bot, 8, \bot) \xrightarrow{0} (\overline{p}_7, f, f, \bot, \bot, \bot)\\
&\xrightarrow{0} (p_7, f, f, 0, \bot, \bot) \xrightarrow{2} (p_7, f, f, 2, \bot, \bot) \xrightarrow{0} (f, f, f, \bot, \bot, \bot)
\end{aligned}$$

5. Implementation and Experimental Results

We have first implemented an algorithm for finding a chain covering for a given partial order. Although the computation of the width and its associated cover is known to be polynomial (via reduction to the max-flow problem), we do not compute it exactly but use a fast and simple algorithm to approximate it. Then we construct an automaton for each chain and apply a variant of the our shortest paths algorithm for finding algorithm for acyclic timed automata developed in [AM01]. This algorithm works on-the-fly and generates global states during the search. Since the scheduling problem is NP-hard, we cannot find the optimal solution for large problems and we use a sub-optimal algorithm, based on a combination of best-first search and breadth-first, where at each level of the search tree, a fixed number of successors is explored (see [AM01]).

To test our approach we took several benchmark problems from [TKK00] having up to few thousands of tasks.[2] For each of them we applied the above procedure for around 1 minute. As one can see from Table 5, our results are very close to the optimal results reported in [TKK00].

[2]The benchmarks can be found in `http://www.kasahara.elec.waseda.ac.jp/schedule/`

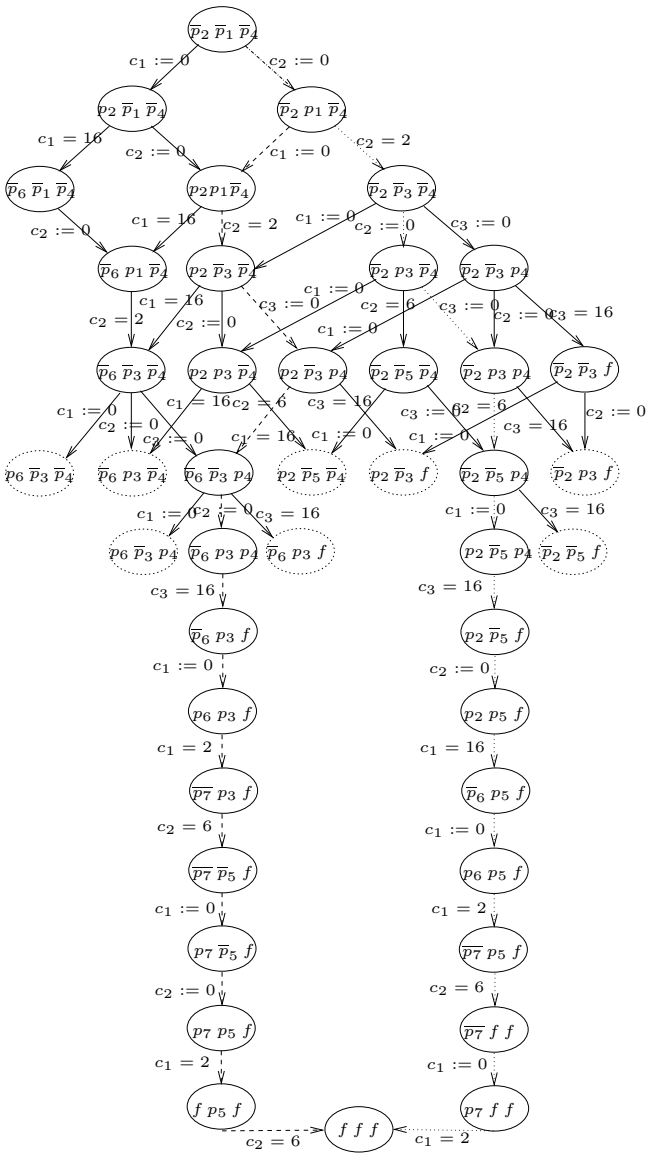


Figure 6. Part of the timed automaton obtained by composing the automata of Figure 5 for the case of 2 machines. The two runs corresponding to the schedules S_2 and S_3 are indicated by the dashed and dotted lines, respectively.

name	#tasks	#chains	# machines	optimal	TA
001	437	125	4	1178	1182
000	452	43	20	537	537
018	730	175	10	700	704
074	1007	66	12	891	894
021	1145	88	20	605	612
228	1187	293	8	1570	1574
071	1193	124	20	629	634
271	1348	127	12	1163	1164
237	1566	152	12	1340	1342
231	1664	101	16	t.o.	1137
235	1782	218	16	t.o.	1150
233	1980	207	19	1118	1121
294	2014	141	17	1257	1261
295	2168	965	18	1318	1322
292	2333	318	3	8009	8009
298	2399	303	10	2471	2473

Table 1. Computation of optimal schedules for benchmark problems. Our results appear in the TA column.

6. Extensions and Conclusions

We extend our model to include two additional feature that are often present in computer scheduling problems, deadlines and release times. For every task P_j a deadline $\lambda(j)$ indicates that the task must imperatively terminate before time $t = \lambda(j)$. The release time $r(j)$ indicates that the task can not be executed before time $t = r(j)$. Hence a feasible schedule st must respect, in addition, two new constraints:

- $\forall P_j \in \mathcal{P}\ st(P_j) + d(j) \leq \lambda(j)$.
- $\forall P_j \in \mathcal{P}\ st(P_j) \geq r(j)$.

These features are easily integrated into the model by making reference to an additional clock t which is never reset and hence it measures absolute time (Figure 7). This clock is used already by the shortest path algorithm. This way a complete run corresponds to a feasible schedule respecting the additional constraints (a run fragment violating a deadline cannot be completed). The results of [AM01] concerning non-lazy schedules hold in this setting as well. We have implemented these features.

There are some useful features that are not yet covered by our model and which are subject to ongoing work:

1. Relative deadlines: in some applications constraints of the form $st(P) - st(P') \leq \lambda$ should be satisfied. Although in terms of modeling such an extension is not difficult to implement, it makes the nature of the solutions more complicated. Roughly speaking, without

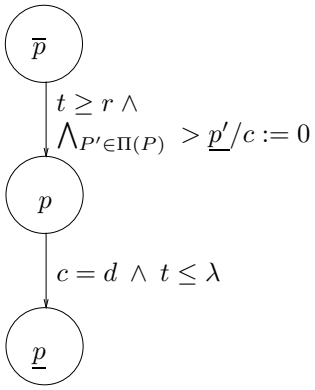


Figure 7. An automaton for a task with release time r and deadline λ.

relative deadlines the scheduling has a "monotone" nature and if a schedule st is feasible, so is and schedule $st' \leq st$ obtained from st by moving forward the start time of one or more tasks. With relative deadlines, it can make sense to delay the beginning of P', not in order to give the machine to another task but in order not to start it too early relative to a later task P.

2. Communication costs: if the tasks communicate with each other to transfer data, the cost of this communication may depend on whether they execute on the same machines. To capture this phenomenon we need a model in which the identity of the machines on which tasks are executed is represented, and in which the structure of the task may vary according to scheduler decisions. For example, if $P \prec P'$ and they run on different machines, a new "communication" task has to be inserted between them.

3. Other resources: a task may need additional resources such as disks and printers that are taken and released at certain points in its execution. This feature can be integrated into the model by annotating states with the sets of resources their associated tasks occupy and modify the definition of composition accordingly.

To summarize, we have shown how "formal" state-based models can be used to express parallel scheduling problems and support efficient algorithmics for solving these problems. A more comprehensive description of the automata-theoretic approach to scheduling can be found in [A02]. We are currently working on various extensions of our modeling framework in order to get closer to real-world problems.

References

[A02] Y. Abdedadïm, *Scheduling with Timed Automata*, PhD Thesis, INPG, Grenoble, 2002.

[AM01] 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.

[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.

[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.

[BFH+01] 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.

[C76] E.G. Coffman, *Computer and Job-Shop Scheduling Theory*, Wiley, 1976.

[D50] R.P. Dilworth, A Decomposition Theorem for Partially Ordered Sets, *Ann. Math.* 51, 161-166, 1950.

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

[KI99] K.-Y. Kwong, A. Ishfaq, Benchmarking and Comparison of the Task Graph Scheduling, *Journal of Parallel and Distributed Computing 59*, 381-422, 1999.

[LPW97b] K.G. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, *International Journal of Software Tools for Technology Transfer* 1, 1997

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

[TKK00] T. Tobita, M. Kouda and H. Kasahara, Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set, *Proc. PDPTA'2000*, 745-751, 2000.

[Y97] S. Yovine, Kronos: A Verification Tool for Real-time Systems, *International Journal of Software Tools for Technology Transfer* 1, 123-133, 1997.