Priced Timed Automata: Algorithms and Applications

Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen * **

Aalborg University

Abstract. This contribution reports on the considerable effort made recently towards extending and applying well-established timed automata technology to optimal scheduling and planning problems. The effort of the authors in this direction has to a large extent been carried out as part of the European projects VHS [22] and AMETIST [17] and are available in the recently released UPPAAL CORA [12], a variant of the real-time verification tool UPPAAL [20, 5] specialized for cost-optimal reachability for the extended model of priced timed automata.

1 Introduction and Motivation

Since its introduction by Alur and Dill [2] the model of timed automata has established itself as a standard modeling formalism for describing real-time system behavior. A number of mature model checking tools (e.g. KRONOS, UPPAAL, IF [11, 20, 16]) are by now available and have been applied to the quantitative analysis of numerous industrial case-studies [25].

An interesting application of real-time model checking that has recently been receiving substantial attention is to extend and re-target the timed automata technology towards optimal scheduling and planning. The extensions include most importantly an augmentation of the basic timed automata formalism allowing for the specification of the accumulation of cost during behavior [7, 3]. The state-exploring algorithms have been modified to allow for "guiding" the (symbolic) state-space exploration in order that "promising" and "cheap" states are visited first, and to apply branch-and-bound techniques [6] to prune parts of the search tree that are guaranteed not to improve on solutions found so far. Also new symbolic data structures allowing for efficient symbolic state-space representation with additional cost-information have been introduced and implemented in order to efficiently obtain optimal or near-optimal solutions [19]. Within the VHS and AMETIST projects successful applications of this technology have been made to a number of benchmark examples and industrial case studies. With this new direction, we are entering the area of Operations Research and Artificial Intelligence with a well-established and extensive list of existing techniques (MILP, constraint programming, genetic programming, etc.). However, what we put forward is a completely new and promising technology based

* BRICS, Aalborg University, Denmark

** Work partially done within the European IST project AMETIST.

on the efficient algorithms/data structures coming from timed automata analysis, and allowing for very natural and compositional descriptions of even highly non-standard scheduling problems with timing constraints.

Abstractly, a scheduling or planning problem may be understood in terms of a number of *objects* (e.g. a number of different cars, persons) each associated with various distinguishing attributes (e.g. speed, position). The possible plans solving the problem are described by a number of *actions*, the execution of which may depend on and affect the values of (some of) the objects attributes. Solutions, or feasible schedules, come in (at least) two flavors:

Finite Schedule: a finite sequence of actions that takes the system from the initial configuration to one of a designated collection of desired goal configurations.

Infinite Schedule: an infinite sequence of actions that – when starting in the initial configuration – ensures that the system configuration stays indefinitely within a designated collection of desired configurations.

In order to reinforce quantitative aspects, actions may additionally be equipped with constraints on durations and have associated costs. In this way one may distinguish different feasible schedules according to their accumulated cost or time (for finite schedules) or their cost per time ratio in the limit (for infinite schedules) in identifying *optimal* schedules. It is understood that independent actions, in terms of the set of objects the actions depend upon and affect, may overlap time-wise.

One concrete scheduling problem is that of optimal *task graph scheduling* (TGS) consisting in scheduling a number of interdependent tasks (e.g. performing some arithmetic operations) onto a number of heterogeneous processors. The interdependencies state that a task cannot start executing before all its predecessors have terminated. Furthermore, each task can only execute on a subset of the processors. An example task graph with three tasks is depicted in Fig. 1. The task t_3 cannot start executing until both tasks t_1 and t_2 have terminated. The available resources are two processors p_1 and p_2. The tasks (nodes) are annotated with the required execution times on the processors, that is, t_1 can only execute on p_1, t_2 only on p_2 while t_3 can execute on both p_1 and p_2. Furthermore, the idling costs per time unit of the processors are 2 and 1, respectively, and operations costs per time unit are 5 and 4, respectively.

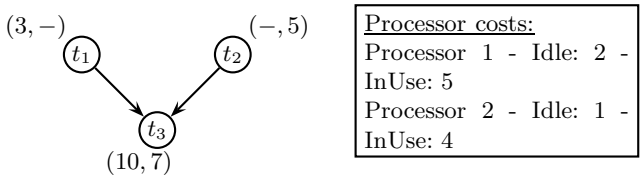


Fig. 1. Task graph scheduling problem with 3 tasks and 2 processors.

Now, scheduling problems are naturally modeled using networks of timed automata. Each object is modeled as a separate timed automaton annotated with

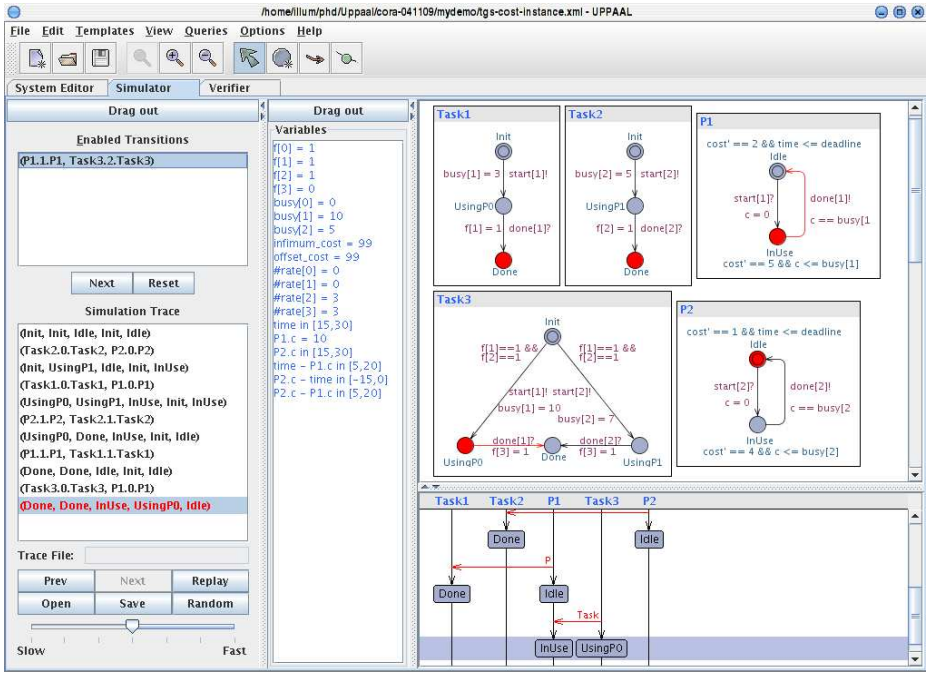


Fig. 2. Screen shot of the UPPAAL CORA simulator for the task graph scheduling problem of Fig. 1.

local, discrete variables representing the attributes associated with the object. Interaction often involves only a few objects and can be modeled as synchronizing edges in the timed automata models of the involved objects. Actions involving time durations are naturally modeled using guarded edges over clock variables. Furthermore, operation costs can be associated with states and edges in the model of priced timed automata (PTA) which was, independently, introduced in [7] and [3]. The separation of independent objects into individual processes and representing interaction between objects as synchronizing actions allows timed automata to make the control flow of scheduling problems explicit. In turn, this makes the models intuitively understood and easy to communicate. Figure 2 depicts PTA models for the task graph in Fig. 1 and is explained in detail in Section 4.2.

The outline of the remainder of the paper is as follows: In Sections 2 and 3 we introduce the model of PTA, the problem of cost-optimal reachability and sketch the symbolic branch-and-bound algorithm used by UPPAAL CORA for solving this problem. Then in Section 4 we show how to model a range of generic scheduling problems using PTA, provide experimental results and describe two industrial scheduling case-studies. Finally, in Section 5, we comment on other PTA-related optimization problems to be supported in future releases of UPPAAL CORA.

2 Priced Timed Automata

In this section we give a formal definition of priced timed automata (PTA) and their semantics[1]. Let X be a set of clocks. Intuitively, clocks are non-negative real valued variables that can be reset to zero and grow at a fixed rate with the passage of time. A priced timed automaton over X is an annotated directed graph with a distinguished vertex called the initial location. In the tradition of timed automata, we call vertices *locations*. An edge is decorated with a guard, an action and a reset set. We say that an edge is enabled if the guard evaluates to true and the source location is active. A reset set is a set of clocks. The intuition is that the clocks in the reset set are set to zero whenever the edge is taken. Note that following edges is instantaneous and thus takes no time. Finally, locations are labeled with invariants. Intuitively, an invariant must evaluate to true whenever its location is active. Both guards and invariants are conjunctions of simple constraints $x \bowtie k$, where x is a clock in X, k is a non-negative integer value, and $\bowtie \in \{<, \leq, =, \geq, >\}$. Let $\mathcal{B}(X)$ be the set of all such expressions. The previous definition is in fact that of a timed automaton. To form a priced timed automaton, we annotate the edges and the locations with costs and cost rates, respectively. The above is summarized in the following definition.

Definition 1 (Priced Timed Automata). *Let X be a set of clocks and Act a set of actions. A priced timed automaton over X and Act is a tuple $A = (L, E, l_0, I, P)$, where L is a set of locations, $E \subseteq L \times \mathcal{B}(X) \times Act \times 2^X \times L$ is a set of edges, $l_0 \in L$ is the initial location, $I : L \to \mathcal{B}(X)$ assigns invariants to locations, and $P : L \cup E \to \mathbb{N}_0$ assigns cost rates and costs to locations and edges, respectively.*

The semantics of a PTA is defined as a *priced transition system*. A priced transition system is a labeled transition system, where the transition relation is given by a partial function from transitions to the non-negative reals, intuitively being the cost of the transition. We write $s \xrightarrow{a}_p s'$ whenever the function is defined on the transition (s, a, s') and the cost is p.

Definition 2 (Priced Transition System). *A priced transition system is a tuple $\mathcal{T} = (S, s_0, \Sigma, \to)$, where S is a (possibly infinite) set of states, $s_0 \in S$ is the initial state, Σ is a set of labels, and $\to$: $(S \times \Sigma \times S) \hookrightarrow \mathbb{R}_{\geq 0}$ is a partial function from transitions to the non-negative reals.*

In case of PTA, a state consists of the active location $l \in L$ and a valuation of all clocks $v : X \to \mathbb{R}_{\geq 0}$ such that the invariant of l evaluates to true for v. There are two types of transitions between these states: *discrete transitions* and *delay transitions*. That is, transitions that instantaneously change the control location of the automaton without time passing and transitions that pass time in a fixed control location, respectively. Consequently, the labels of the corresponding priced transition system consists of the labels of the priced timed automaton and the non-negative reals. We formalize this in the following definition.

[1] We ignore the syntactic extensions of discrete variables and parallel composition of automata and note that these can be added easily.

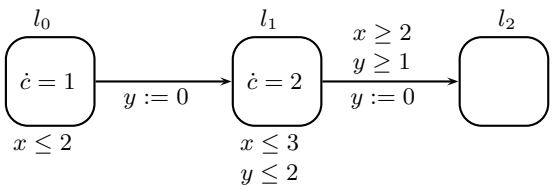


Fig. 3. A priced timed automaton, A.

Definition 3 (Semantics of a Priced Timed Automaton). *The semantics of a PTA $A = (L, E, l_0, I, P)$ over clocks X and actions Act is given by a priced transition system $\mathcal{T} = (S, s_0, \Sigma, \rightarrow)$, where $S = \{(l, u) \in L \times \mathbb{R}^X_{\geq 0} \mid u \models I(l)\}$ is the set of states satisfying the invariants, $s_0 = (l_0, u_0)$ is the initial state for u_0 evaluating to zero for all clocks in X, $\Sigma = Act \cup \mathbb{R}_{\geq 0}$ is the set of labels, and $\rightarrow$ consists of discrete and delay transitions as defined below.*

Discrete transitions are the result of following an enabled edge in the PTA. As a result, the destination location is activated and the clocks in the reset set are set to zero. The cost of the transition is given by the cost of the edge.

Definition 4 (Discrete transitions). *A transition $(l, v) \xrightarrow{a}_p (l', v')$ is a discrete transition iff there is an edge (l, g, a, r, l') from l to l', such that the guard, g, evaluates to true in the source state (l, v), v' is derived from v by resetting all clocks in the reset set, r, and $p = P(e)$ is the cost of the edge.*

Delay transitions are the result of the passage of time and do not cause a change of location. A delay is only valid if the invariant of the active location is satisfied by all intermediary states. The cost of a delay transition is given by the product of the duration of the delay and the cost rate of the active location.

Definition 5 (Delay transitions). *A transition $(l, v) \xrightarrow{d}_p (l, v')$ is a delay transition iff $p = d \cdot P(l)$, $v' = v + d$,[2] and the invariant of l is satisfied by the source, target and all intermediary states, i.e., for all non-negative delays d' less than or equal to d we have $v + d' \models I(l)$.*

For networks of timed automata we use vectors of locations and the cost rate of a vector of locations is the sum of cost rates in the locations of the vector.

Example 1. Now consider the priced timed automaton A in Fig. 3 having two clocks x and y, a single goal location l_2 and two locations l_0 and l_1 with cost rate 1 and 2 respectively. Below we offer three sample traces of A:

$$\begin{aligned} \alpha_0 = (l_0, x = 0, y = 0) &\rightarrow_0 (l_1, x = 0, y = 0) \xrightarrow{2}_4 (l_1, x = 2, y = 2) \\ &\rightarrow_0 (l_2, x = 2, y = 0) \end{aligned}$$

[2] $v + d$ is the clock valuation derived from v by incrementing all clocks by d.

$$\alpha_1 = (l_0, x=0, y=0) \xrightarrow{2}_2 (l_0, x=2, y=2) \rightarrow_0 (l_1, x=2, y=0)$$
$$\xrightarrow{1}_2 (l_1, x=3, y=1) \rightarrow_0 (l_2, x=3, y=0)$$
$$\alpha_2 = (l_0, x=0, y=0) \xrightarrow{1}_1 (l_0, x=1, y=1) \rightarrow_0 (l_1, x=1, y=0)$$
$$\xrightarrow{1}_2 (1_1, x=2, y=1) \rightarrow_0 (l_2, x=2, y=0)$$

□

3 Optimal Scheduling

We now turn to the definition of the optimal reachability problem for PTA and provide a brief and intuitive overview of Uppaal Cora's branch and bound algorithm for cost-optimal reachability analysis.

Cost-optimal reachability is the problem of finding the minimum cost of reaching a given goal location. More formally, an execution of a PTA is a path in the priced transition system defined by the PTA (see above), i.e., $\alpha = s_0 \xrightarrow{a_1}_{p_1} s_1 \xrightarrow{a_2}_{p_2} s_2 \cdots \xrightarrow{a_n}_{p_n} s_n$. The cost, $cost(\alpha)$, of execution α is the sum of all the costs along the execution, i.e. $\sum_i p_i$. The minimum cost, $mincost(s)$ of reaching a state s is the infimum of the costs of all finite executions from s_0 to s. Given a PTA with location l, the *cost-optimal reachability problem* is to find the largest cost k such that $k \leq mincost((l, v))$ for all clock valuations v.

Example 2. Referring to example 1, the accumulated cost of the three traces are, respectively, $cost(\alpha_0) = cost(\alpha_1) = 4$ and $cost(\alpha_2) = 3$. Thus, among the three suggested traces, α_2, leads to l_2 with minimum cost. In fact, as we shall see later, this is the minimum cost by which l_2 may be reached by *any* trace of A. □

Since clocks are defined over the non-negative reals, the priced transition system generated by a PTA can be uncountably infinite, thus an enumerative, explicit state approach to the cost-optimal reachability problem is infeasible. Instead, we build upon the work done for timed automata by using *priced symbolic states*. Priced symbolic states provide symbolic representations of possibly infinite sets of actual states and their association with costs. The idea is that during exploration, the infimum cost along a symbolic path (a path of symbolic states) is stored in the symbolic state itself. If the same state is reached with different costs along different paths, the symbolic states can be compared, discarding the more expensive state.

Analogous to timed automata, a priced symbolic state of a PTA can be represented as a location and a priced zone. Priced zones describe sets of clock valuations and their associated costs. The set of clock valuations is described as a simple constraint system over clocks and differences between clocks, called zones. The cost is an affine hyperplane in an $|X| + 1$ dimensional Euclidean space, where each point is a clock valuation and an associated cost, i.e., for each clock valuation in the zone, the hyperplane provides a cost of that valuation.

We observe that the constraint system describing a zone can be simplified by adding an additional clock, $\mathbf{0}$, that by definition is zero in all valuations. Then constraints on individual clocks can be represented as constraints on differences between clocks, e.g., $x < 5$ becomes $x - \mathbf{0} < 5$. A zone be efficiently represented as a *difference bound matrix* or DBM [13]. It represents the constraint system describing the zone as a $|X| + 1$ dimensional matrix, with entries c_{ij} meaning $x_i - x_j \leq c_{ij}$ for clocks $x_i, x_j \in X \cup \{\mathbf{0}\}$. Extending the data structure to a priced DBM, we add an affine hyperplane. The offset point is the unique valuation such that all valuations in the zone are component-wise equal or larger than the offset point.[3]

Definition 6 (Priced Zone). *A priced zone over a set of clocks X is a pair (Z, f), where Z is a zone, i.e., a conjunction of constraints on clocks or differences between clocks, and f is an affine function over X providing the cost of the clock valuations satisfying the constraints of Z.*

Without going into details on how to compute the successors of a priced symbolic state, we notice that the representation of priced zones as priced DBMs support the necessary operations to do so. In particular, the data structure supports computing the set of delay successors of a priced zone and computing the projection of a priced zone (for resetting clocks). The crucial addition compared to regular DBMs is the efficient manipulation of the hyperplane in such a manner, that any state in the resulting zone is associated with the lowest cost of immediately reaching that state from a state in the predecessor. Also, given two symbolic states S and S', computing whether one dominates the other is efficiently computable. E.g. S is dominated by S' if for all states s in S, s is in S' and the cost of s in S' is lower than the cost in S.

Example 3. Figure 4 illustrates a symbolic exploration of the priced timed automaton A from Fig. 3 in terms of the following symbolic trace:

$$\begin{aligned} \Gamma = (l_0, (Z_0, x)) &\rightarrow (l_1, (Z_1, x)) \\ &\rightarrow (l_1, (Z_2, x + y)) \rightarrow (l_2, (Z_3, x + 1)) \end{aligned}$$

where $Z_0 = (x = y \wedge x \leq 2)$, $Z_1 = (y = 0 \wedge x \leq 2)$, $Z_2 = (y \leq 2 \wedge x \leq 3 \wedge 0 \leq x - y \leq 2)$ and $Z_3 = (y = 0 \wedge 2 \leq x \leq 3)$. The zone $Z_2' = (1 \leq y \leq 2 \wedge 2 \leq x \leq 3)$ is the subset of Z_2 for which the edge from l_1 to l_2 is enabled. Now, from the final symbolic state $(l_2, (Z_3, x + 1))$ we see that we may reach l_2 with cost 3 given as the minimum value of the affine function $x + 1$ with respect to the constraints of the zone Z_3. This minimum value is clearly obtained at the state $(l_2, x = 2, y = 0)$. Now, we may follow this state backwards within the given symbolic trace Γ constantly selecting the predecessor-state with minimum cost. In this way we are to (re)produce the concrete minimum-cost trace:

$$\begin{aligned} \alpha_2 = (l_0, x = 0, y = 0) &\xrightarrow{1}_1 (l_0, x = 1, y = 1) \rightarrow_0 (l_1, x = 1, y = 0) \\ &\xrightarrow{1}_2 (l_1, x = 2, y = 1) \rightarrow_0 (l_2, x = 2, y = 0) \end{aligned}$$

[3] Alternatively, the cost at the origin could be given.

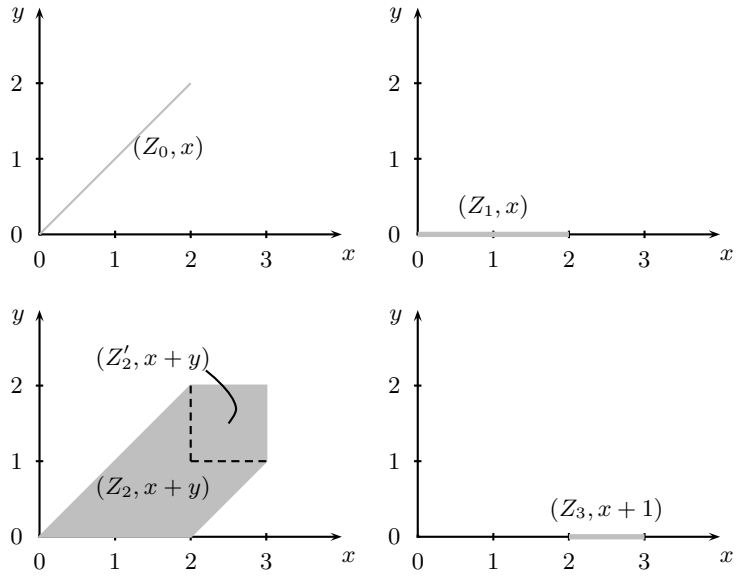


Fig. 4. Symbolic exploration of A using priced zones.

□

In Uppaal Cora, cost-optimal reachability analysis is performed using a standard branch and bound algorithm. Branching is based on various search strategies implemented in Uppaal Cora which, currently, are breadth-first, ordinary, random, or best depth-first with or without random restart, best-first, and user supplied heuristics. The latter enables the user to annotate locations of the model with a special variable called *heur* and the search can be ordered according to either largest or smallest *heur* value. Bounding is based on a user-supplied, lower-bound estimate of the *remaining* cost to reach the goal from each location, i.e. an admissible heuristic.

The algorithm depicted in Fig. 5 is the cost-optimal reachability algorithm used by Uppaal Cora. It maintains a Passed-list of symbolic states that have been explored and a Waiting-list of symbolic state that need to be explored and is instantiated with the initial symbolic state $\mathcal{S}_0$. The variable Cost holds the currently best known cost of reaching the goal location; initially it is infinite. The algorithm iterates until no more symbolic states need to be explored. Inside the while-loop we select and remove a symbolic state, $\mathcal{S}$, from Waiting based on the branching strategy. If $\mathcal{S}$ is dominated by another symbolic state that has already been explored or it is not possible to reach the goal with a lower cost than Cost, we skip this symbolic state. Otherwise, we add $\mathcal{S}$ to Passed and if $\mathcal{S}$ is a goal location we update the best known cost to the best cost in $\mathcal{S}$. If not, we add all successors of $\mathcal{S}$ to Waiting and continue to the next iteration. Note

```
COST := ∞
PASSED := ∅
WAITING := {S_0}
while WAITING ≠ ∅ do
    select S ∈ WAITING //based on branching strategy
    C ← infimum(S)
    if PASSED ⋠_dom S and C + remain(S) < COST then
        PASSED ← PASSED ∪ {S}
        if S ∈ GOAL then
            COST ← C
        else
            WAITING ← {S' | S' ∈ WAITING or S → S'}
return COST
```

Fig. 5. Branch and bound algorithm for cost optimal reachability analysis of priced timed automata. The algorithm works on priced symbolic states and uses auxiliary functions for computing the infimum cost of all states in a symbolic state and for checking whether a symbolic state dominates another symbolic state.

that the algorithm does not terminate when the first goal location is discovered which is custom with a best-first branch and bound algorithm. The reason is that we allow various branching strategies some which do not guarantee the first found goal location to be optimal.

4 Modeling

As mentioned earlier, one of the main strengths of using priced timed automata for specifying and analyzing scheduling problems is the simplicity of the modeling aspect in terms of compositional descriptions. In this section, we show how to model well-known, generic scheduling problems, provide experimental results, and describe two industrial case studies.

Scheduling problems often consist of a set of passive objects, called resources, and a set of active objects, called tasks. The resources are passive in the sense that they provide a service that tasks can utilize. Traditionally, the scheduling problem is to complete the tasks as fast as possible using the available resources under some constraints, e.g. limited availability of the resource, no two tasks can, simultaneously, use the same resource, etc. The models we provide in this section are all cost extensions of classical scheduling problems.

A generic resource model (see Fig. 6a) is a two-location cyclic process with a single local clock, **c**. The two locations indicate whether the resource is **Idle** or **InUse**. The resource moves from **Idle** to **InUse**, when a task initiates a synchronization over the channel **start** and in the process, **c** is reset. The resource will maintain **InUse** until the clock reaches some usage time, **busy**, it then initiates synchronization over the channel **done**.

A generic task model (see Fig. 6b) is an acyclic process progressing from an initial location, **Init**, to a final location **Done**, indicating task completion. Inter-

mediate locations describe acquiring resources and releasing them, i.e. the task will transit to state **Using** by initiating synchronization over a **start** channel and setting the **busy** variable of the resource. The task will remain here until the resource initiates synchronization using the **done** channel.

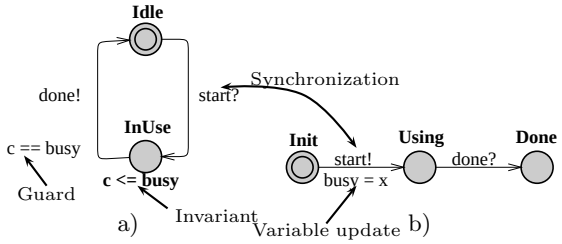


Fig. 6. a) Resource template with clock **c**. b) Task template.

To solve the scheduling problem, we pose the reachability question of whether we can reach a state in which all tasks are in the location **Done**. In the following four sections we present some classical scheduling problems, all of which are slight modifications of the generic templates.

4.1 Job Shop Scheduling

Problem: We are given a number of machines (resources) and a number jobs (tasks) with corresponding recipes. A recipe for a job dictates the subset of machines that the job should be processed by, the order in which the processing should happen, and the duration of each processing step. Now, the scheduling problem is to assign to each job a starting time for every required machine such that no machine is occupied by two jobs at the same time.

Cost: The model can be extended with costs by assigning to each machine an idling cost and a operation cost.

Modeling: Figure 7a depicts a job and a machine. The model of the machine is identical to the resource template, except that both locations have been extended with cost rates. The job model is a "serial composition" of the task template, i.e. the job serially requests the machines described by the recipe, in this case machines 0, 1, and 2 for 7, 5, and 15 time units, respectively.

4.2 Task Graph Scheduling

Problem: This problem is described in Section 1.

Cost: We assign to each processor an energy consumption rate while idle and while executing. Now, the overall objective is to find the schedule that minimizes the total cost while respecting a global (or task individual) deadline.

Modeling: The models for a task and a processor are depicted in Fig. 2. Again, the processor model is an exact instance of the resource template with added cost rates. Tasks 1 and 2 are exact instances of the task template, while task 3 is not. The reason is that tasks 1 and 2 can only execute on one processor each, while task 3 can execute on both, thus, task 3 is an extension of the task template with a nondeterministic choice between the processors. Furthermore, the edges leaving the initial state have been extended with a guard specifying the dependencies of the task graph, i.e. task 3 requires tasks 1 and 2 to be finished, i.e. **f[1]** && **f[2]**.

4.3 Vehicle Routing with Time Windows

Problem: We are given a depot owning a fleet of vehicles (resources) with limited capacity of some good and a number of dispersed customers (tasks) with individual demands and time windows. The scheduling problem is to assign routes to each vehicle such that customers are served within their time windows and the total demand of a route does not exceed the capacity of the vehicle. Usually, the unloading process is also associated with a delay linear in the amount to unload and each vehicle is expected to return to the depot.

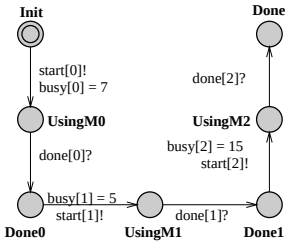
Cost: For a schedule, costs are incurred while vehicles are in operation (driver salary, **ds**), i.e. away from the depot, and extra costs are added while driving corresponding to the fuel consumption, **fc**.

Modeling: Figure 7b depicts models for a customer and for a vehicle. The customer model (having time window [30,90]) is a combination of the job model and the model of task 3 above. The customer can acquire either vehicle, hence the nondeterministic choice and the sequential part corresponds to acquiring the vehicle to arrive (**ComingHere**) and to unload the goods (**Unloading**). Note that besides updating the vehicle busy time with a driving distance, the vehicle capacity, **carcap**, is updated to reflect the demand. The requirements for the time window are realized through guards and invariants on the global time. The vehicle model is a slight variation of the resource template, as the **InUse** location has been replaced by two locations to distinguish between **Driving** and **Unloading**. Furthermore, there is an acyclic part reflecting the possibility of **DrivingHome** to the depot and thus completing the route. In all locations except **Home**, there is a cost rate corresponding to the driver salary and in the driving locations there is an added fuel cost.

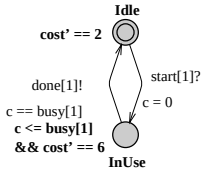
4.4 Aircraft Landing

Problem: Given a number of aircrafts (tasks) with designated type and landing time window, assign a landing time and runway (resource) to each aircraft such that the aircraft lands within the designated time window while respecting a minimum wake turbulence separation delay between aircrafts of various types landing on the same runway.

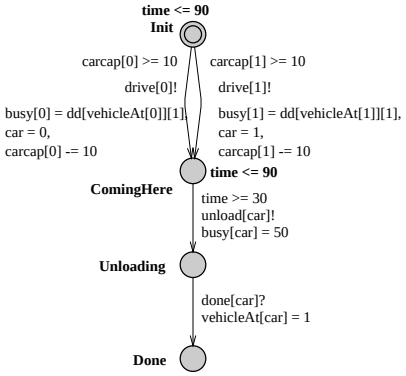
a) Job:



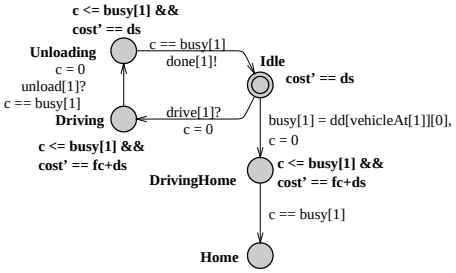
Machine:



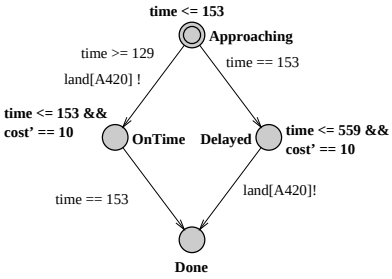
b) Customer:



Vehicle:



c) Aircraft:



Runway:

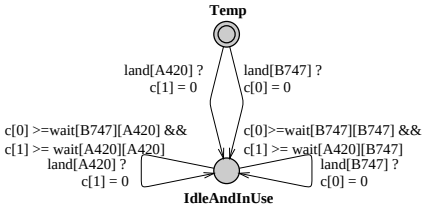


Fig. 7. Priced timed automata models for two classical scheduling problems.

Cost: The cost extended problem associates with each aircraft an additional target landing time corresponding to approaching the runway at cruise speed. Now, if an aircraft is assigned a landing time earlier than the target landing time, a cost per time unit is incurred, corresponding to powering up the engines. Similarly, if an aircraft is assigned a later landing time than the target landing time a cost per time unit is added corresponding to increased fuel consumption while circling above the airport.

Modeling: Figure 7c depicts a runway that can handle aircrafts of types B747 and A420, and an aircraft with target landing time 153, type A420 and time window [129,559]. Unlike the other models, the runway model has only a single location in its cycle indicating both that the resource is **IdleAndInUse**. A single location is used since the duration that a runway is occupied depends solely on the types of consecutively landing aircrafts. Thus, the runway maintains a clock per aircraft type holding the time since the latest landing of an aircraft of the given type and access to the runway is controlled by guards on the edges. The nondeterminism of the aircraft model does not distinguish between the runway to use, but whether to land early ([129,153]) or late ([153,559]). Choosing to land early, the aircraft model moves to the **OnTime** location and must remain here until the target landing time while incurring a cost rate per time unit for landing early, similarly, the aircraft can choose to land late and move to **Delayed** may remain there until the latest landing time while paying a cost rate for landing late.

4.5 PTA versus MILP

We only provide experimental results for the aircraft landing problem comparing the PTA approach to that of MILP. For performance results of the job shop and task graph scheduling problems, we refer to [6, 23, 1].

Figure 8 displays experimental results for various instances of the aircraft landing problem using MILP and PTA. The results for MILP have been taken from [4] and the results for PTA have been executed on a comparable computer. Factors in bold indicate the performance difference in favor of PTA and similarly for italics and MILP. The experiments clearly indicate that PTA is a competitive approach to solving scheduling problems and for one non-trivial instance it is even more than a factor 250 faster than the MILP approach. However, the required computation time of the PTA approach grows exponentially with the number of added runways (and thus clocks) while no similar statement can be made for the MILP approach. The exponential growth of the PTA approach is no surprise as reachability is exponential in the number of clocks. However, this does not mean that PTA are unsuited for larger problems, but merely that the models should be carefully considered to minimize the number of clocks. Furthermore, techniques from timed automata theory to deal with clocks such as omitting certain "inactive" clocks from locations has been extended to PTA.

In conclusion, PTA is a promising method for solving scheduling problems, but further experiments need to be conducted before saying anything more conclusive.

RW	Planes	10	15	20	20	20	30	44
	Types	2	2	2	2	2	4	2
1	MILP (s)	*0.4*	*5.2*	*2.7*	*220.4*	*922.0*	*33.1*	*10.6*
	MC (s)	**0.8**	**5.6**	**2.8**	**20.9**	**49.9**	**0.6**	**2.2**
	Factor	*2.0*	*1.08*	*1.04*	**10.5**	**18.5**	**55.2**	**48.1**
2	MILP (s)	*0.6*	*1.8*	*3.8*	*1919.9*	*11510.4*	*1568.1*	*0.2*
	MC (s)	**2.7**	**9.6**	**3.9**	**138.5**	**187.1**	**6.0**	**0.9**
	Factor	*4.5*	*5.3*	*1.02*	**13.9**	**61.5**	**261.3**	*4.5*
3	MILP (s)	*0.1*	*0.1*	*0.2*	*2299.2*	*1655.3*	*0.2*	N/A
	MC (s)	**0.2**	**0.3**	**0.7**	**1765.6**	**1294.9**	**0.6**	
	Factor	*2.0*	*3.0*	*3.5*	**1.30**	**1.28**	*3.0*	
4	MILP (s)	N/A	N/A	N/A	*0.2*	*0.2*	N/A	N/A
	MC (s)				**3.3**	**0.7**		
	Factor				*16.5*	*3.5*		

Fig. 8. Computational result for the aircraft landing problem using PTA and MILP on comparable machines.

4.6 Industrial Case Study: Steel Production

Problem: Proving schedulability of an industrial plant via reachability analysis of a timed automaton model was first applied to the SIDMAR steel plant, which was included as a case study of the Esprit-LTR Project 26270 VHS (Verification of Hybrid Systems). The plant consists of five processing machines placed along two tracks and a casting machine where the finished steels leaves the system. The tracks and machines are connected via two overhead cranes. Each quantity of raw iron enters the system in a ladle and depending on the desired final steel quality undergoes treatments in the different machines for different durations. The planning problem consists in controlling the movement of the ladles of steel between the different machines, taking the topology (e.g. conveyor belts and overhang cranes) into consideration.

Performance: A schedule for three ladles was produced in [14] for a slightly simplified model using UPPAAL. In [15] schedules for up to 60 ladles were produced also using UPPAAL. However, in order to do this, additional constraints were included that reduce the size of the state-space dramatically, but also prune possibly sensible behavior. A similar reduced model was used by Stobbe [24] using constraint programming to schedule 30 ladles. All these works only consider ladles with the same quality of steel. In [6], using a search order based on priorities, a schedule for ten ladles with varying qualities of steels is computed within 60 seconds CPU-time on a Pentium II 300MHz. The initial solution found is improved by 5% within the time limit. Allowing the search to go on for longer, models with more ladles can be handled.

4.7 Industrial Case Study: Lacquer Production

Problem: The problem was provided by an industrial partner of the European AMETIST project as a variation on job shop scheduling. The task is to schedule lacquer production. Lacquer is produced according to a recipe involving the use of various resources, possibly concurrently, see Fig. 9. An *order* consists of a recipe, a quantity, an earliest starting date and a delivery date. The problem is then to assign resources to the order such that the constraints of the recipes and of the orders are met. Additional constraints are provided by the resources, as they might require cleaning when switching from one type of lacquer to another, or might require manual labor and thus are unavailable during the night or in weekends.

Cost: The cost model is similar to that of the aircraft landing problem. Orders finished on the delivery date do not incur any costs (except regular production costs which are not modeled as these are fixed). Orders finishing late are subject to *delay costs* and orders finishing too early are subject to *storage costs*. Cleaning resources might generate additional costs.

Modeling: Resources are modeled using the resource template. Resources requiring cleaning are extended with additional information to keep track of the last type of lacquer produced on the resource. Cleaning costs are typically a fixed amount and are added to the cost when cleaning is performed. Orders are modeled similarly to tasks in the task graph scheduling problem, except that multiple resources may be acquired simultaneously. Storage and delay costs are modeled similarly to costs in the aircraft landing problem.

5 Other Optimization Problems

At present Uppaal Cora supports cost-optimal location-reachability for PTAs. However, a number of other optimization problems are planned to be included in future releases. In the following we give a brief description of these extensions with illustrating examples.

Infinite Schedules

For several planning problems the objective is to *repeat* a treatment or process *indefinitely* and to do so in a cost-optimal manner. Now let $\alpha = s_0 \xrightarrow{a_1}_{p_1} s_1 \xrightarrow{a_2}_{p_2} s_2 \cdots \xrightarrow{a_n}_{p_n} s_n \cdots$ be an infinite execution of a given PTA, let c_n (t_n) denote the accumulated cost (time) after n steps (i.e. $c_n = \sum_{i=1}^{n} p_i$). Then the limit of c_n/t_n when $n \to \infty$ describes the cost per time of α in the long run and is the cost of α. The optimization problem is to determine the (value of the) optimal such infinite execution α^*.

Example 4. Consider the priced timed automaton B of Fig. 10 being a cyclic extension of the priced timed automaton A of Fig. 3. Below we offer two infinite

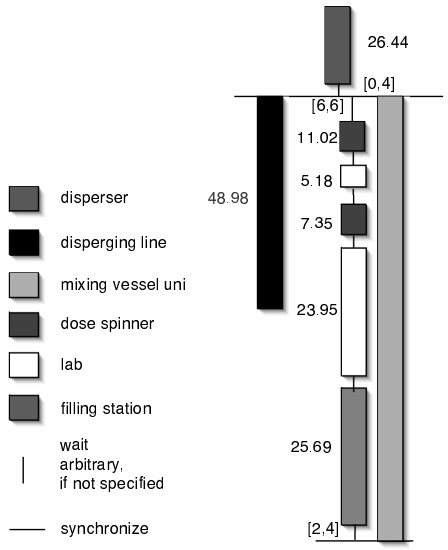


Fig. 9. A lacquer recipe. Each bar represents the use of a resource. Horizontal lines indicate synchronization points. Timing constraints for how long resources are used or separation times between the use of resources can be provided either as a fixed time or time window.

(cyclic) traces (* indicates the nested cycle):

$$\begin{aligned}\beta_0 = (l_0, x=0, y=0) &\xrightarrow{1}_1 (l_0, x=1, y=1) \rightarrow_0 (l_1, x=1, y=0)^* \\ &\xrightarrow{2}_4 (l_1, x=3, y=2) \rightarrow_0 (l_2, x=3, y=0) \\ &\xrightarrow{1}_3 (l_2, x=3, y=1) \rightarrow_0 (l_0, x=0, y=1) \\ &\xrightarrow{1}_1 (l_0, x=1, y=2) \rightarrow_0 (l_1, x=1, y=0)^*\end{aligned}$$

$$\begin{aligned}\beta_1 = (l_0, x=0, y=0) &\rightarrow_0 (l_1, x=0, y=0)^* \xrightarrow{2}_4 (l_1, x=2, y=2) \\ &\rightarrow_0 (l_2, x=0, y=0) \xrightarrow{2}_6 (l_2, x=2, y=2) \\ &\rightarrow_0 (l_0, x=0, y=2) \rightarrow_0 (l_1, x=0, y=0)^*\end{aligned}$$

For the two infinite traces β_0 and β_1 their cyclic nature entails that the limit of cost per time is given as the ratio of cost per time of the nested cycles. Thus we find that:

$$\begin{aligned}ratio(\beta_0) &= (4+3+1)/(2+1+1) = 2 \\ ratio(\beta_1) &= (4+6)/(2+2) = 2.5\end{aligned}$$

and hence that β_0 offers the better solution. □

In [8] the problem of identifying the optimal infinite execution (and the limit-ratio of this execution) has been shown decidable for PTA using a so-called

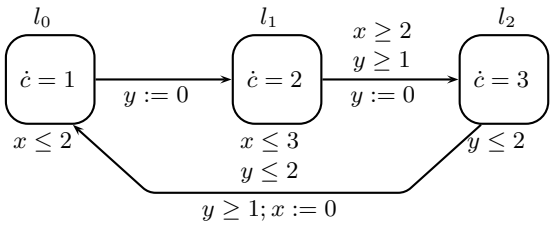


Fig. 10. A cyclic priced timed automaton, B.

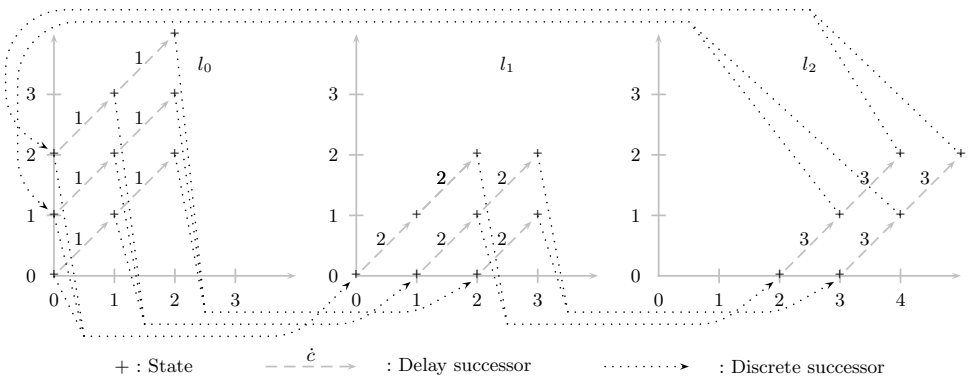


Fig. 11. Discrete time semantics for PTA, B.

"corner-point" abstraction which is an extension of the classical region-technique for timed automata. In case of non-strict guards — as is the case of the priced timed automaton of Fig. 10 — the "corner-point" abstraction is identical to the discrete-time semantics of the automaton. The problem now reduces to that of identifying a cycle with minimum mean-cost in the corresponding finite weighted graph, a problem for which Karp's algorithm [18] provides a cubic solution. Figure 11 illustrates the discrete semantics of the priced timed automaton B of Fig. 10.

Though the "corner-point" abstraction technique nicely demonstrates decidability of the problem (and many other decision problems for timed automata) it does not provide a practical implementation, which is still to be identified. However, a method for determining *approximate optimal* infinite schedules have been identified and applied to the synthesis of Dynamic Voltages Scaling scheduling strategies.

Multiple Cost Variables

Optimization problems may involve *multiple* cost variables (e.g. money, energy, pollution, etc.). Currently Uppaal Cora is only capable of optimizing with respect to single costs. However, for scheduling problems with multiple costs, there might well be several optimal solutions due to "negative" dependencies

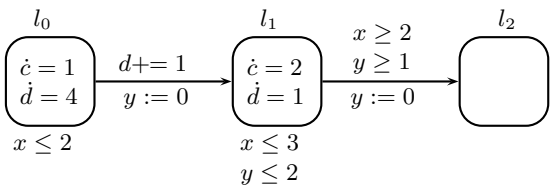


Fig. 12. A dual-priced timed automaton, C.

between costs, i.e. minimizing one cost variable (e.g. money) might maximize others (e.g. pollution).

Example 5. Figure 12 illustrates a dual-priced TA C extending the (single) priced TA A of Fig. 3 with a second cost variable d. The following two traces both reach the goal location l_2 but with incompatible cost-pairs, namely $(4, 2)$ versus $(3, 5)$.

$$\begin{aligned}\gamma_0 = (l_0, x = 0, y = 0) &\rightarrow_{(0,0)} (l_1, x = 0, y = 0) \xrightarrow{2}_{(4,2)} (l_1, x = 2, y = 2) \\ &\rightarrow_{(0,0)} (l_2, x = 2, y = 0)\end{aligned}$$

$$\begin{aligned}\gamma_1 = (l_0, x = 0, y = 0) &\xrightarrow{1}_{(1,4)} (l_0, x = 1, y = 1) \rightarrow_{(0,0)} (l_1, x = 1, y = 0) \\ &\xrightarrow{1}_{(2,1)} (1_1, x = 2, y = 1) \rightarrow_{(0,0)} (l_2, x = 2, y = 0)\end{aligned}$$

□

In [21] the notion of priced zone for PTA has been extended to multi-price TA allowing efficient synthesis of solutions optimal with respect to a chosen *primary* cost variable, but subject to user-specified upper bounds on the remaining *secondary* cost variables. More precisely, the symbolic exploration of multi-priced TAs uses multi-priced zones of the type $P = (Z, \{\mathbf{c}_1, \ldots, \mathbf{c}_n\})$, where $\mathbf{c}_i$ is a vector of affine cost-functions (one for each cost variable). Now, any concrete trace to a given state will be associated with a cost-vector giving a value (cost) for each of the involved cost variables. As illustrated by Example 5, two different traces to a given state may, in the multi-priced case, be associated with incomparable cost-vectors in the sense that neither one dominates the other with respect to component-wise $\leq$. In our symbolic treatment we deal with this phenomenon by associating the zone Z with *sets* of cost-function vectors. Now for any clock-valuation $u \in Z$ the multi-priced zone P will associate not only the set of cost-vectors $\{\mathbf{c}_1(u), \ldots, \mathbf{c}_n(u)\}$ but also all convex combinations of these vectors. We refer the interested reader to [21] for more information on this. In the following example we try to illustrate our symbolic treatment.

Example 6. Figure 13 illustrates the symbolic exploration of the dual-priced TA C of Fig. 12. In the final symbolic state the zone Z_3 is associated with a set

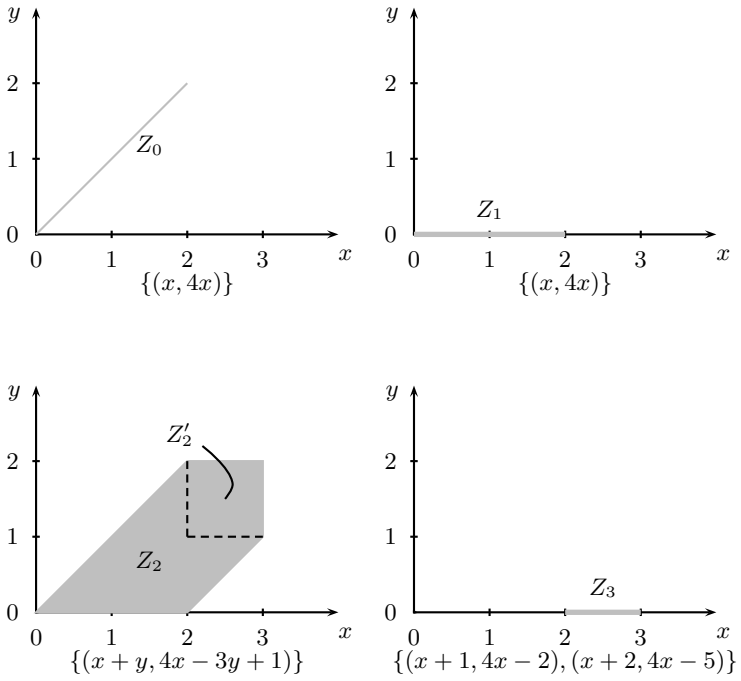


Fig. 13. Symbolic exploration of C using dual-priced zones.

containing two cost-function pair: $\{(x+1, 4x-2), (x+2, 4x-5)\}$. Now evaluating these two pairs with respect to the extrema points of Z_3 we obtain a set of 4 cost-pairs: $\{(3, 6), (4, 10), (4, 3), (5, 7)\}$, all combinations of which are (according to the theory) realizable. Thus, in case we want to minimize c subject to the condition that d stays below 4 it can be seen that $c = \frac{11}{4}$ is the minimum such value. □

Uncertainty

Finally, scheduling problems may involve *uncertainties* due to certain actions being under the control of an adversary. In this case the (optimal) scheduling problem is a game-theoretic problem consisting of determining a winning and optimal strategy for how to respond to any action chosen by this adversary. In [9] the problem of synthesizing optimal, winning strategies for priced timed games has been shown to be computable under certain non-zenoness assumptions. However, the problem is not solvable using zone-based technology, but needs general polyhedral support in order to represent the optimal strategies (see [10] for a methodology using HyTech).

Example 7. Consider the priced timed game automaton D of Fig. 14. Here the cost-rates in locations l_0, l_2 and l_3 are $5, 10$ and 1 respectively. In l_1 the adversary may choose to move to either l_2 or l_3 (dashed arrows are under control of the

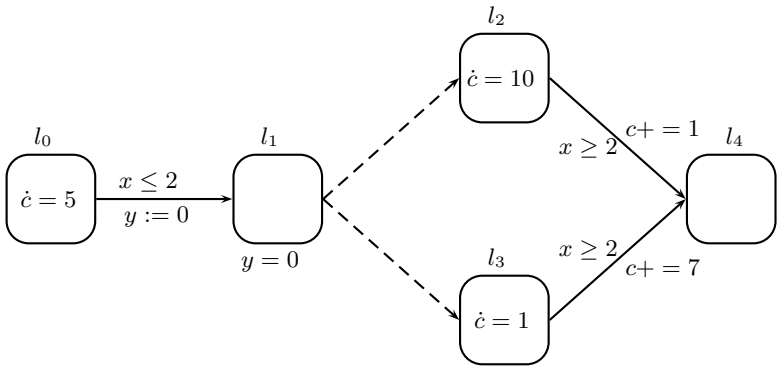


Fig. 14. A priced timed game automaton, D

adversary). However, due to the invariant $y = 0$ this choice must be made instantaneously. Obviously, once l_2 or l_3 has been reached the optimal strategy for the controller is to move to the goal location l_4 immediately. Note that there is a discrete cost (respectively 1 and 7) on each discrete transition. The crucial (and only remaining) question is how long the controller should wait in l_0 before taking the transition to l_1. Obviously, in order for the controller to win this duration must be no more than two time units. However, what is the optimal choice for the duration in the sense that the overall cost of reaching l_4 will be minimal? Denote by t the chosen delay in l_0. Then $5t + 10(2 - t) + 1$ is the minimal cost through l_2 and $5t + (2 - t) + 7$ is the minimal cost through l_3. As the adversary chooses between these two transitions the best choice for the controller is to delay $t \leq 2$ such that $max(21 - 5t, 9 + 4t)$ is minimum, which is obtained for $t = \frac{4}{3}$ giving a minimal cost of $14\frac{1}{3}$. □

References

1. Y. Abdeddaim, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. *In Proc. of International Parallel and Distributed Processing Symposium*, pages 8–15, 2003.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. R. Alur, S. La Torre, and G. Pappas. Optimal paths in weighted timed automata. *Lecture Notes in Computer Science*, 2034:pp. 49–62, 2001.
4. J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34(2):pp. 180–197, 2000.
5. G. Behrmann, A. David, and K. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, number 3185 in Lecture Notes in Computer Science, pages 200–236. Springer Verlag, 2004.
6. G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, and J. Romijn. Efficient guiding towards cost-optimality in Uppaal. In *Proc. of Tools and Algorithms for the Construction and Analysis of System.s*, number 2031 in Lecture Notes in Computer Science, pages 174–188. Springer–Verlag, 2001.

7. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In *Proc. of Hybrid Systems: Computation and Control*, number 2034 in Lecture Notes in Computer Sciences, pages 147–161. Springer–Verlag, 2001.
8. P. Bouyer, E. Brinksma, and K. Larsen. Staying alive as cheaply as possible. In *Proc. of Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 203–218. Springer–Verlag, 2004.
9. P. Bouyer, F. Cassez, E. Fleury, and K. Larsen. Optimal strategies in priced timed game automata. In *Proc. of Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer–Verlag, 2004.
10. Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Synthesis of optimal strategies using hytech. In *Workshop on Games in Design and Verification*, volume 119(1) of *Electronic Notes in Theoretical Computer Science*, pages 11–31, Boston, MA, USA, July 2004. Elsevier Science Publishers.
11. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proc. of Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer-Verlag, 1998.
12. UPPAAL CORA. `http://www.cs.aau.dk/~behrmann/cora`, Jan. 2005.
13. D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proc. Of Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notex in Computer Science*, pages 197–212. Springer–Verlag, 1989.
14. A. Fehnker. Scheduling a steel plant with timed automata. In *Proc. of Real-Time and Embedded Computing Systems and Applications.*, page 280. IEEE Computer Society, 1999.
15. T. Hune, K. Larsen, and P. Pettersson. Guided synthesis of control programs using Uppaal. *Nordic Journal of Computing*, 8(1):43–64, 2001.
16. IF. `http://www-verimag.imag.fr/~async/IF`, Jan. 2005.
17. Advanced Methods in Timed Systems (AMETIST). `http://ametist.cs.utwente.nl`, Jan. 2005.
18. R. M. Karp. A characterization of the minimum mean-cycle in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978.
19. K. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. of Computer Aided Verification*, volume 2102, pages pp. 493+, 2001.
20. K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
21. K. Larsen and J. Rasmussen. Optimal conditional reachability for multi-priced timed automata. In *Proc. of Foundations of Software Science and Computation Structures*, volume 3441 of *Lecture Notes in Computer Science*, pages 234–249. Springer–Verlag, 2005.
22. Verification of Hybrid Systems (VHS). `http://www-verimag.imag.fr/VHS/`, Jan. 2005.
23. J. Rasmussen, K. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *Proc. of Tool and Algortihms for the Construction and Analysis of Systems*, volume 2988 of *Lecture Notes in Computer Science*, pages 220–235. Springer Verlag, 2004.

24. M. Stobbe. Results on scheduling the sidmar steel plant using constraint programming. Internal report, 2000.
25. UPPAAL. http://www.uppaal.com, Jan. 2005.