Optimal Scheduling using Priced Timed Automata

Gerd Behrmann Kim G. Larsen Jacob I. Rasmussen [1] [2]

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 [20] and AMETIST [16] and are available in the recently released UPPAAL CORA [12], a variant of the real-time verification tool UPPAAL [18, 5] specialized for cost-optimal reachability for the extended model of so-called 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 behaviour. A number of mature model checking tools (e.g. KRONOS, UPPAAL, IF [11, 18, 15]) are by now available and have been applied to the quantitative analysis of numerous industrial case-studies [23].

An interesting application of real-time model checking that has recently been receiving substantial attention is to extend and retarget 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 acculumation 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 [17]. 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 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 on the efficient algorithms/data structures coming from timed automta analysis, and allowing for very natural and compositional descriptions of 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:

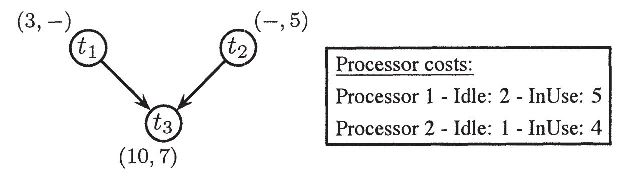
Finite Schedule: a finite sequence of actions that takes the system from the initial configuration to one of a designated collection of desired final configurations.

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 objects the actions depend upon and may 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 heterogenous 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 Figure 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.

[1]BRICS, Aalborg University, Denmark
[2]Work partially done within the European IST project AMETIST.



$(3,-)$
t_1
$(-,5)$
t_2
t_3
$(10,7)$
Processor costs:
Processor 1 - Idle: 2 - InUse: 5
Processor 2 - Idle: 1 - InUse: 4

Figure 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 modelled as a separate timed automaton annotated with 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 explicit the control flow of scheduling problems. In turn, this makes the models intuitively understood and easy to communicate. Figure 2 depicts PTA models for the task graph in Figure 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 sketches 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 evaluation 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 more precise and formal definition of priced timed automata (PTA) and their semantics[1]. Let X be a set of clocks. Intuitively, a clock is a non-negative real valued variable that can be reset to zero and increments at a fixed rate with the passage of time. A priced timed automaton over X is an annotated directed graph with a vertex set L, an edge set E and a distinguished vertex $l_0 \in L$ called the initial location. In the tradition of timed automata, we call vertices *locations*. Edges are labelled with guard expressions and a reset set. A guard is a conjunction of simple constraints $x \bowtie k$, where x is a clock in X, k is a non-negative integer value, and $\bowtie \in \{<, \leq, =, \geq, >\}$. We say that an edge is enabled if the guard evaluates to true and the source location is active. A reset set is a subset of X. The intuition is that the clocks in the reset set are set to zero whenever the edge is taken. Finally, locations are labelled with invariants. An invariant is a conjunction of simple conditions $x \prec k$, where x is a clock in X, k is a non-negative integer-value, and $\prec \in \{<, \leq\}$. Intuitively, an invariant must evaluate to true whenever its location is active. 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. Formally, this is done by introducing a function $P : L \cup E \rightarrow \mathbb{N}_0$.

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

The semantics of a PTA is easily defined as a *priced transition system.* Transitions of a priced transition system are labelled with a non-negative real-valued cost p. We skip the formal definition of a priced transition system and proceed with the semantics of PTA. A state of a PTA contains the active location $l \in L$ and a valuation of all clocks $v : X \rightarrow \mathbb{R}_{\geq 0}$ such that the invariant of l evaluates to true for v. There are two types of transitions: *edge transitions* and *delay transitions.* Edge 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.
More formally, we have $(l, v) \rightarrow_p (l', v')$ if there is an edge e from l to l', such that the guard of e evaluates to true in the source state (l, v), v' is derived from v by resetting all clocks in the reset set of e, 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 intermediate states. The cost of a delay transition is given by the product of the length of the delay and the cost rate of the active location.

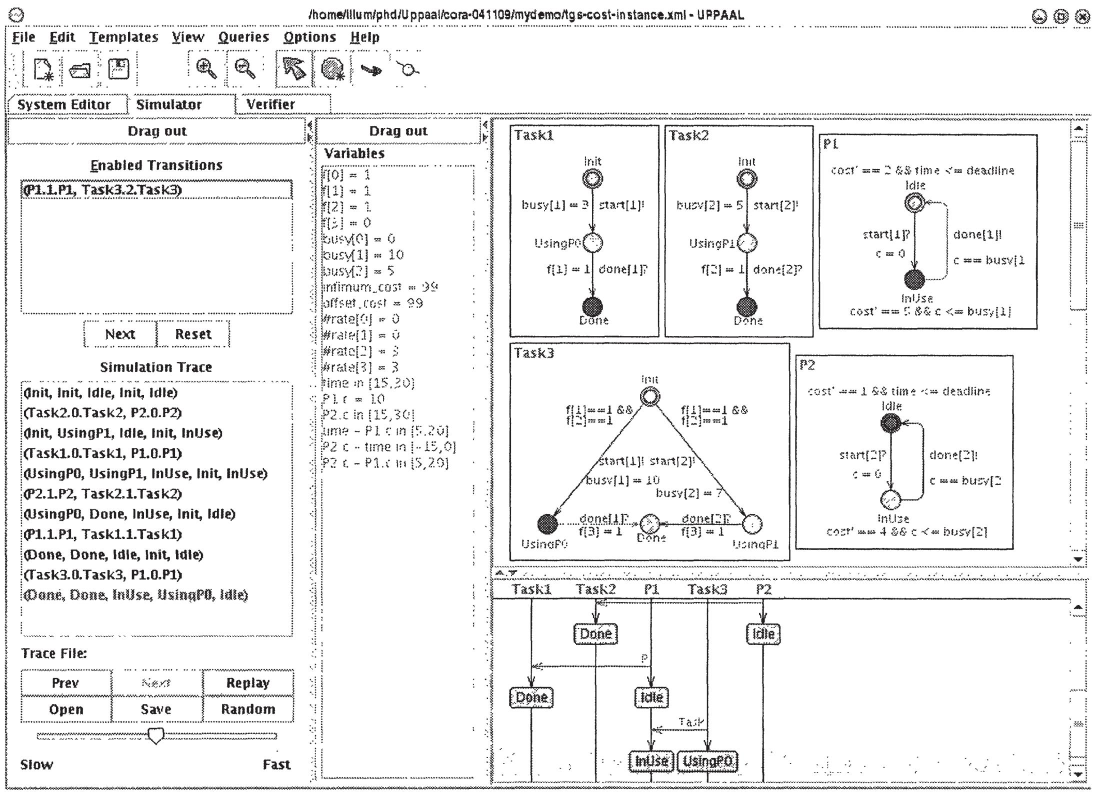
More formally, we have $(l, v) \xrightarrow{\delta}_p (l, v')$ if $p = \delta \cdot P(l)$, v' is derived from v by incrementing all clocks by δ and the invariant of l is satisfied by (l, v), (l, v') and all intermediary states.

Finally, the initial state is $s_0 = (l_0, v_0)$, where l_0 is the initial location, and v_0 evaluates to zero for all clocks. For networks of timed automata we use vectors of locations and the cost rate of a vector is the sum of cost rates in locations.

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, i.e., $\alpha = s_0 \xrightarrow{a_1}_{p_1} s_1 \xrightarrow{a_2}_{p_2} s_2 \cdots \xrightarrow{a_n}_{p_n} s_n$.



Enabled Transitions:

```
(P1.1.P1, Task3.2.Task3)
```

Variables:

```
f[0] = 1
f[1] = 1
f[2] = 1
f[3] = 0
busy[0] = 0
busy[1] = 10
busy[2] = 5
infimum_cost = 99
offset_cost = 99
#rate[0] = 0
#rate[1] = 0
#rate[2] = 3
#rate[3] = 3
time in [15,20]
P1.c = 10
P2.c in [15,30]
time - P1.c in [5,20]
P2.c - time in [-15,0]
P2.c - P1.c in [5,20]
```

Simulation Trace:

```
(Init, Init, Idle, Init, Idle)
(Task2.0.Task2, P2.0.P2)
(Init, UsingP1, Idle, Init, InUse)
(Task1.0.Task1, P1.0.P1)
(UsingP0, UsingP1, InUse, Init, InUse)
(P2.1.P2, Task2.1.Task2)
(UsingP0, Done, InUse, Init, Idle)
(P1.1.P1, Task1.1.Task1)
(Done, Done, Idle, Init, Idle)
(Task3.0.Task3, P1.0.P1)
(Done, Done, InUse, UsingP0, Idle)
```

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

```
COST := $\infty$
PASSED := $\emptyset$
WAITING := $\{\mathcal{S}_0\}$
while WAITING $\neq \emptyset$ do
    select $\mathcal{S} \in$ WAITING //based on branching strategy
    C $\leftarrow$ $mincost(\mathcal{S})$
    if PASSED $\not\preceq_{dom}$ $\mathcal{S}$ and C + $remain(\mathcal{S})$ < COST then
        PASSED $\leftarrow$ PASSED $\cup$ $\{\mathcal{S}\}$
        if $\mathcal{S}$ $\in$ GOAL then
            COST $\leftarrow$ C
        else
            WAITING $\leftarrow$ $\{\mathcal{S}' \mid \mathcal{S}' \in$ WAITING or $\mathcal{S} \rightarrow \mathcal{S}'\}$
return COST
```

Figure 3: branch and bound algorithm.

The cost, $cost(\alpha)$, of execution α is the sum of all the costs along the execution. 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.

Since clocks are defined over the non-negative reals, the priced transition system generated by a PTA can be uncountably infinite, thus an enumerative 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 states 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, the priced symbolic states we encounter for PTA are representable by simple constraint systems over clock differences (often refered to as a clock-zone in the timed automata literature). The cost is given by an affine plane over the clock-zone. For a formal description of priced symbolic states and priced zones we refer to [17, 21].

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.

The algorithm depicted in Figure 3 is the cost-optimal reachability algorithm used by UPPAAL CORA. It maintains a PASSED-list of elements that have been explored and a WAITING-list of elements 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 states need to be explored. Inside the while-loop we select and remove a state, $\mathcal{S}$, from WAITING based on the branching strategy. If $\mathcal{S}$ is dominated[2] by another state that has already been explored or it is not possible to reach the goal with a lower cost than COST, we skip this 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.

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 this section, we show how to model 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 the classical scheduling problem.

A generic resource model (see Figure 4a) 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 Figure 4b) is an acyclic process progressing from an initial location, **Init**, to a final location **Done**, indicating that the task is complete. Intermediate 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.

To solve the scheduling problem, we pose the reachability

[2]A state, $\mathcal{S}'$, dominates another state, $\mathcal{S}$, if $\mathcal{S}'$ contains at least the same actual states as $\mathcal{S}$, all of which have been reached with a lower cost.

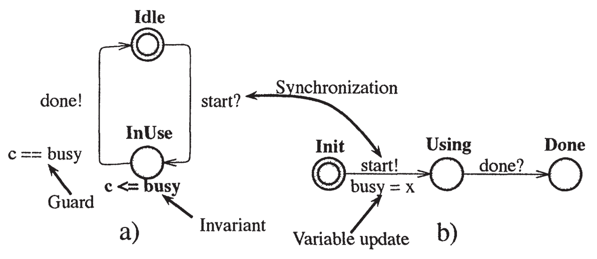


Figure 4: a) Resource template with clock **c**. b) Task template.

question of whether we can reach a state in which all tasks are in the location **Done**. In the following three 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 5a 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 Figure 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, **f[1]** && **f[2]**.

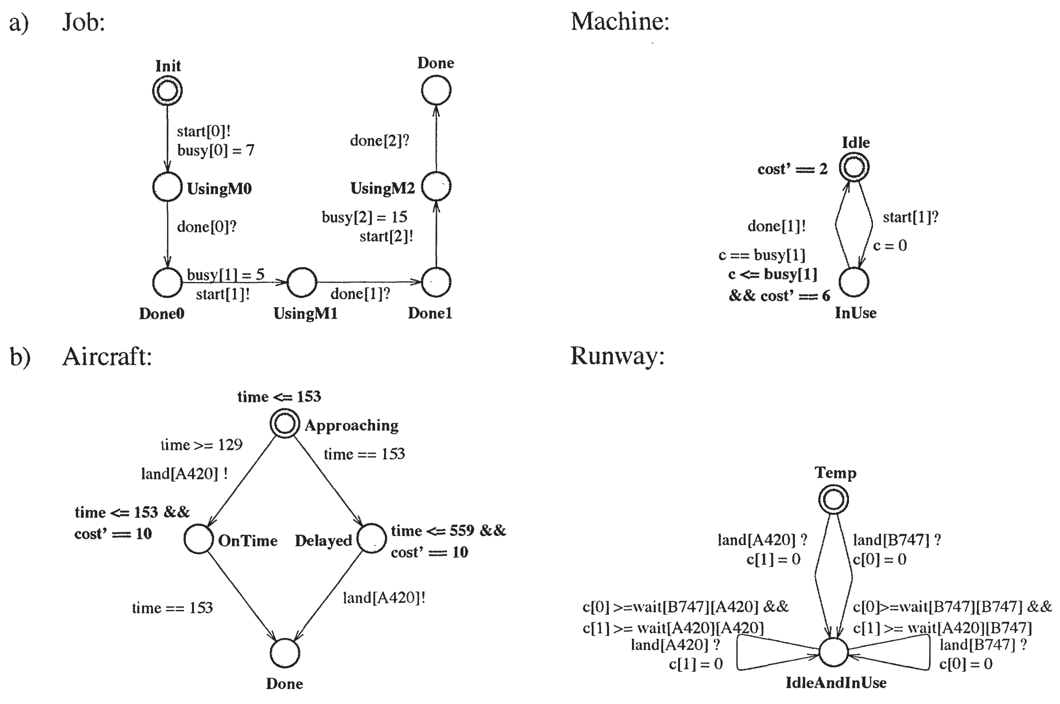


Figure 5: Priced timed automata models for two classical scheduling problems.

4.3 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.
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 5b 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

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**
	MILP (s)	*0.6*	*1.8*	*3.8*	*1919.9*	*11510.4*	*1568.1*	*0.2*
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*		

Figure 6: Computational result for the aircraft landing problem using PTA and MILP on comparable machines.

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.

4.4 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, 21, 1].

Figure 6 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. Thus, PTA is a promising method for solving scheduling problems, but further experiments need to be conducted before saying anything more conclusive.

4.5 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 [13] for a slightly simplified model using UPPAAL. In [14] 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 [22] 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.

4.6 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 Figure 7. 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

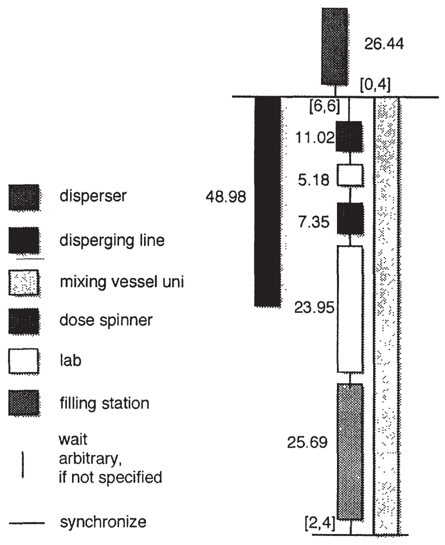


Figure 7: A lacquer recipe. Each bar represents the use of a resource. Horizontal lines indicate synchronization points. Timing constraints for how long resources are used or separation times between the use of resources can be provided either as a fixed time or time window.

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.

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 $n \longrightarrow \infty$ 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^*. In [8] this problem has been shown decidable for PTA using an extension of the so-called regiontechnique. Though this 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 so-called Dynamic Voltages Scaling scheduling strategies.

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 between costs: minimizing one cost-variable (e.g. money) might maximize others (e.g. pollution). In [19] an extension of the priced zone technology 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.

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).

References

[1] Y. Abdeddaim, A. Kerbaa, and O. Maler. Task graph scheduling using timed automata. *Proc. of IPDPS'03*, 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 TACAS'01*, number 2031 in Lecture Notes in Computer Science, pages 174–188. Springer–Verlag, 2001.

[7] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. *Lecture Notes in Computer Science*, 2034:pp. 147+, 2001.

[8] P. Bouyer, E. Brinksma, and K. Larsen. Staying alive as cheaply as possible. In *Proc. of HSCC'04*, 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 FSTTCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer–Verlag, 2004.

[10] P. Bouyer, F. Cassez, E. Fleury, and K. Larsen. Synthesis of optimal strategies using HyTech. In *Proc. GDV'04*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2004. To appear.

[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 CAV'98*, volume 1427, pages 546–550. Springer-Verlag, 1998.

[12] UPPAAL CORA. `http://www.cs.aau.dk/~behrmann/cora`, Jan. 2005.

[13] A. Fehnker. Scheduling a steel plant with timed automata. In *Proc. of RTCSA '99.*, page 280. IEEE Computer Society, 1999.

[14] T. Hune, K. Larsen, and P. Pettersson. Guided synthesis of control programs using Uppaal. *Nordic J. of Computing*, 8(1):43–64, 2001.

[15] IF. `http://www-verimag.imag.fr/~async/IF`, Jan. 2005.

[16] Advanced Methods in Timed Systems (AMETIST). `http://ametist.cs.utwente.nl`, Jan. 2005.

[17] 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. *Lecture Notes in Computer Science*, 2102:pp. 493+, 2001.

[18] 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.

[19] K. Larsen and J. Rasmussen. Optimal conditional reachability for multi-priced timed automata. *To appear in proceedings of FOSSACS'05*, 2005.

[20] Verification of Hybrid Systems (VHS). `http://www-verimag.imag.fr/VHS/`, Jan. 2005.

[21] J. Rasmussen, K. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *Proc. of TACAS'04*, volume 2988 of *Lecture Notes in Computer Science*, pages pp. 220–235. Springer Verlag, 2004.

[22] M. Stobbe. Results on scheduling the sidmar steel plant using constraint programming. Internal report, 2000.

[23] UPPAAL. `http://www.uppaal.com`, Jan. 2005.