Efficient Guiding Towards Cost-Optimality in UPPAAL*

Gerd Behrmann[1], Ansgar Fehnker[3†], Thomas Hune[2], Kim Larsen[4], Paul Pettersson[5], and Judi Romijn[3]

[1] Basic Research in Computer Science, Aalborg University, E-mail: behrmann@cs.auc.dk
[2] Basic Research in Computer Science, Aarhus University, E-mail: baris@brics.dk
[3] Computing Science Institute, University of Nijmegen, E-mail: [ansgar,judi]@cs.kun.nl
[4] Department of Computer Science, University of Twente[§], E-mail: kgl@cs.auc.dk
[5] Department of Computer Systems, Information Technology, Uppsala University, E-mail: paupet@docs.uu.se.

Abstract. In this paper we present an algorithm for efficiently computing the minimum cost of reaching a goal state in the model of Uniformly Priced Timed Automata (UPTA). This model can be seen as a submodel of the recently suggested model of linearly priced timed automata, which extends timed automata with prices on both locations and transitions. The presented algorithm is based on a symbolic semantics of UTPA, and an efficient representation and operations based on difference bound matrices. In analogy with Dijkstra's shortest path algorithm, we show that the search order of the algorithm can be chosen such that the number of symbolic states explored by the algorithm is optimal, to be optimal, in the sense that the number of explored states can not be reduced by any other search order. We also present a number of techniques inspired by branch-and-bound algorithms which can be used for limiting the search space and for quickly finding near-optimal solutions.
The algorithm has been implemented in the verification tool UPPAAL. When applied on a number of experiments the presented techniques reduced the explored state-space with up to 90%.

1 Introduction

Recently, formal verification tools for real-time and hybrid systems, such as UPPAAL [LPY97], KRONOS [BDM+98] and HYTECH [HHWT97], have been applied

* This work is partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems).
§ On sabbatical from Basic Research in Computer Science, Aalborg University.
† Research supported by Netherlands Organization for Scientific Research (NWO) under contract SION 612-14-004.

to solve realistic scheduling problems [Feh99b,HLP00,NY99]. The basic common idea of these works is to reformulate a scheduling problem to a reachability problem that can be solved by verification tools. In this approach, the automata based modeling languages of the verification tools serve as the input language in which the scheduling problem is described. These modeling languages have been found to be very well-suited in this respect, as they allow for easy and flexible modeling of systems consisting of several parallel components that interact in a time-critical manner and constrain the behavior of each other in a multitude of ways.

A main difference between verification algorithms and dedicated scheduling algorithms is in the way they search a state-space to find solutions. Scheduling algorithms are often designed to find optimal (or near optimal) solutions and are therefore based on techniques such as branch-and-bound to identify and prune parts of the states-space that are guaranteed to not contain any optimal solutions. In contrast, verification algorithms do normally not support any notion of optimality and are designed to explore the entire state-space as efficiently as possible. The verification algorithms that do support notions of optimality are restricted to simple trace properties such as shortest trace [LPY95], or shortest accumulated delay in trace [NTY00].

In this paper we aim at reducing the gap between scheduling and verification algorithms by adopting a number of techniques used in scheduling algorithms in the verification tool UPPAAL. In doing so, we study the problem of efficiently computing the minimal cost of reaching a goal state in the model of *Uniformly Priced Timed Automata* (UPTA). This model can be seen as a restricted version of the recently suggested model of *Linearly Priced Timed Automata* (LPTA) [BFH$^+$01], which extends the model of timed automata with *prices* on all transitions and locations. In these models, the cost of taking an action transition is the price associated with the transition, and the cost of delaying d time units in a location is $d \cdot p$, where p is the price associated with the location. The cost of a trace is simply the accumulated sum of costs of its delay and action transitions. The objective is to determine the minimum cost of traces ending in a goal state.

The infinite state-spaces of timed automata models necessitates the use of symbolic techniques in order to simultaneously handle sets of states (so-called symbolic states). For pure reachability analysis, tools like UPPAAL and KRONOS use symbolic states of the form (l, Z), where l is a location of the timed automaton and $Z \subseteq \mathbb{R}^{\mathbb{C}}$[1] is a convex set of clock valuations called a *zone*. For the computation of minimum costs of reaching goal states, we suggest the use of *symbolic cost states* of the form (l, C), where $C : \mathbb{R}^{\mathbb{C}} \to (\mathbb{R}_{\geq 0} \cup \{\infty\})$ is a cost function mapping clock valuations to real valued costs or ∞. The intention is that, whenever $C(u) < \infty$, reachability of the symbolic cost state (l, C) should ensure that the state (l, u) is reachable with cost $C(u)$.

Using the above notion of symbolic cost states, an abstract algorithm for computing the minimum cost of reaching a goal state satisfying φ of a uniformly

[1] $\mathbb{C}$ denotes the set of clocks of the timed automata, and $\mathbb{R}^{\mathbb{C}}$ denotes the set of functions from $\mathbb{C}$ to $\mathbb{R}_{\geq 0}$.

$\text{Cost} := \infty$
$\text{Passed} := \emptyset$
$\text{Waiting} := \{(l_0, C_0)\}$
while $\text{Waiting} \neq \emptyset$ **do**
  select (l, C) from Waiting
  if $(l, C) \models \varphi$ **and** $min(C) < \text{Cost}$ **then**
    $\text{Cost} := min(C)$
  if for all (l, C') in Passed: $C' \not\sqsubseteq C$ **then**
    add (l, C) to Passed
    for all (m, D) such that $(l, C) \rightsquigarrow (m, D)$: add (m, D) to Waiting
return Cost

Fig. 1. Abstract Algorithm for the Minimal-Cost Reachability Problem.

priced timed automaton is shown in Fig. 1. The algorithm is similar to a standard state-space traversal algorithm that uses two data-structures Waiting and Passed to store states waiting to be examined, and states already explored, respectively. Initially, Passed is empty and Waiting holds an initial (symbolic cost) state. In each iteration, the algorithm proceeds by selecting a state (l, C) from Waiting, checking that none of the previously explored states (l, C') has a "smaller" cost function, written $C' \sqsubseteq C$[2], and if this is the case, adds it to Passed and its successors to Waiting. In addition the algorithm uses the global variable Cost, which is initially set to ∞ and updated whenever a goal state is found that can be reached with a lower cost than the current value of Cost. The algorithm terminates when Waiting is empty, i.e. when no further states are left to be examined. Thus, the algorithm always searches the entire state-space of the analyzed automaton.

In [BFH+01] an algorithm for computing the minimal cost of reaching designated goal states was given for the full model of LPTA. However, the algorithm is based on a cost-extended version of regions, and is thus guaranteed to be extremely inefficient and highly sensitive to the size of constants used in the models. As the first contribution of this paper, we give for the subclass of UPTA an efficient zone representation of symbolic cost states based on *Difference Bound Matrices* [Dil89], and give all the necessary symbolic operators needed to implement the algorithm. As the second contribution we show that, in analogy with Dijkstra's shortest path algorithm, if the algorithm is modified to always select from Waiting the (symbolic cost) state with the smallest minimum cost, the state-space exploration may terminate as soon as a goal state is to be explored. This means that we can solve the minimal-cost reachability problem without necessarily searching the entire state-space of the analyzed automaton. In fact, it can even be shown that the resulting algorithm is optimal in the sense that choosing to search a symbolic cost state with non-minimal minimum cost can never reduce the number of symbolic cost states explored.

The third contribution of this paper is a number of techniques inspired by branch-and-bound algorithms [AC91] that have been adopted in making the

[2] Formally $C' \sqsubseteq C$ iff $\forall u.\, C'(u) \leq C(u)$.

algorithm even more useful. These techniques are particularly useful for limiting the search space and for quickly finding solutions near to the minimum cost of reaching a goal state. To support this claim, we have implemented the algorithm in an experimental version of the verification tool UPPAAL and applied it to a wide variety of examples. Our experimental findings indicate that in some cases as much as 90% of the state-space searched in ordinary breadth-first order can be avoided by combining the techniques presented in this paper. Moreover, the techniques have allowed pure reachability analysis to be performed in cases which were previously unsuccessful.

The rest of this paper is organized as follows: In Section 2 we formally define the model of uniformly priced timed automata and give the symbolic semantics. In Section 3 we present the basic algorithm and the branch-and-bound inspired techniques. The experiments are presented in Section 4. We conclude the paper in Section 5.

2 Uniformly Priced Timed Automata

In this section linearly priced timed automata are formalized and their semantics are defined. The definitions given here resemble those of [BFH+01], except that the symbolic semantics uses cost functions whereas [BFH+01] uses priced regions. Zone-based data-structures for compact representation and efficient manipulation of cost functions are provided for the class of uniformly priced timed automata. It is simple to extend linearly priced timed automata to networks of linearly priced timed automata, but for brevity parallel composition is omitted here.

2.1 Linearly Priced Timed Automata

Formally, linearly priced timed automata (LPTA) are timed automata with prices on locations and transitions. We also denote prices on locations as rates. Let $\mathbb{C}$ be a set of clocks. Then $\mathcal{B}(\mathbb{C})$ is the set of formulas that are conjunctions of atomic constraints of the form $x \bowtie n$ and $x - y \bowtie n$ for $x, y \in \mathbb{C}$, $\bowtie \in \{<, \leq, =, \geq, >\}$ and n being a natural number. Elements of $\mathcal{B}(\mathbb{C})$ are called clock constrains over $\mathbb{C}$. $\mathcal{P}(\mathbb{C})$ denotes the power set of $\mathbb{C}$.

Definition 1 (Linearly Priced Timed Automata). *A linearly priced timed automaton A over clocks $\mathbb{C}$ and actions Act is a tuple (L, l_0, E, I, P) where L is a finite set of locations, l_0 is the initial location, $E \subseteq L \times \mathcal{B}(\mathbb{C}) \times Act \times \mathcal{P}(\mathbb{C}) \times L$ is the set of edges, where an edge contains a source, a guard, an action, a set of clocks to be reset, and a target, $I : L \to \mathcal{B}(\mathbb{C})$ assigns invariants to locations, and $P : (L \cup E) \to \mathbb{N}$ assign prices to both locations and edges. In the case of $(l, g, a, r, l') \in E$, we write $l \xrightarrow{g,a,r} l'$.*

Clock values are represented as functions called clock valuations from $\mathbb{C}$ to the non-negative reals $\mathbb{R}_{\geq 0}$. We denote by $\mathbb{R}^{\mathbb{C}}$ the set of clock valuations for $\mathbb{C}$.

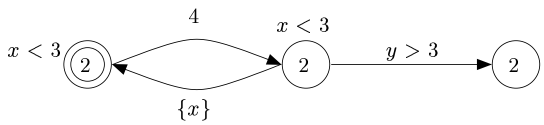

Fig. 2. An example of an LPTA with two clocks, x and y. The number in the states is the rate of the state and the number on the transitions is the cost of taking the transition. A minimal trace to the rightmost state needs to visit the initial state twice, and has cost 14.

Definition 2 (Semantics). *The semantics of a linearly priced timed automaton A is defined as a labeled transition system with the state-space $L \times \mathbb{R}^{\mathbb{C}}$ with initial state (l_0, u_0) (where u_0 assigns zero to all clocks in $\mathbb{C}$) and with the following transition relation:*

- $(l, u) \xrightarrow{\epsilon(d),p} (l, u + d)$ *if* $\forall 0 \leq e \leq d : u + e \in I(l)$, *and* $p = d \cdot P(l)$,
- $(l, u) \xrightarrow{a,p} (l', u')$ *if there exists* g, r *s.t.* $l \xrightarrow{g,a,r} l'$, $u \in g$, $u' = u[r \mapsto 0]$, *and* $p = P((l, g, a, r, l'))$,

where for $d \in \mathbb{R}_{\geq 0}$, $u + d$ *maps each clock* x *in* $\mathbb{C}$ *to the value* $u(x) + d$, *and* $u[r \mapsto 0]$ *denotes the clock valuation which maps each clock in* r *to the value 0 and agrees with* u *over* $\mathbb{C} \setminus r$.

The transitions are decorated with a delay-quantity or an action, together with the cost of the transition. The cost of an execution trace is simply the accumulated cost of all transitions in the trace, see Fig. 2.

Definition 3 (Cost). *Let* $\alpha = (l_0, u_0) \xrightarrow{a_1,p_1} (l_1, u_1) \cdots \xrightarrow{a_n,p_n} (l_n, u_n)$ *be a finite execution trace. The cost of* α, $cost(\alpha)$, *is the sum* $\Sigma_{i=1}^{n} p_i$. *For a given state* (l, u) *the minimum cost* $mincost(l, u)$ *of reaching the state, is the infimum of the costs of finite traces ending in* (l, u). *For a given location* l *the minimum cost* $mincost(l)$ *of reaching the location, is the infimum of the costs of finite traces ending in* (l, u) *for some* u.

2.2 Cost Functions

The semantics of LPTA yields an uncountable state-space and is therefore not suited for state-space exploration algorithms. To overcome this problem, the algorithm in Fig. 1 uses symbolic cost states, quite similar to how timed automata model checkers like UPPAAL use symbolic states.

Typically, symbolic states are pairs on the form (l, Z), where $Z \subseteq \mathbb{R}^{\mathbb{C}}$ is a convex set of clock valuations, called a zone, representable by *Difference Bound Matrices* (DBMs) [Dil89]. The operations needed for forward state-space exploration can be efficiently implemented using the DBM data-structure. In the priced setting we must in addition represent the costs with which individual states are reached. For this we suggest the use of *symbolic cost states*, (l, C),

Table 1. Common operations on cost functions.

Operation	Cost Function ($\mathbb{R}^{\mathbb{C}} \to \mathbb{R}_{\geq 0}$)
Delay	$delay(C,p) : u \mapsto \inf\{C(v) + p \cdot d \mid d \in \mathbb{R}_{\geq 0} \wedge v + d = u\}$
Reset	$r(C) : u \mapsto \inf\{C(v) \mid u = r(v)\}$
Satisfaction	$g(C) : u \mapsto \min\{C(v) \mid v \models g \wedge u = v\}$
Increment	$C + k : u \mapsto C(u) + k, k \in \mathbb{N}$
Comparison	$D \sqsubseteq C \overset{def}{\Leftrightarrow} \forall u : D(u) \leq C(u)$
Infimum	$min(C) = \inf\{C(u) \mid u \in \mathbb{R}^{\mathbb{C}}\}$

where C is a cost function mapping clock valuations to real valued costs. Thus, within a symbolic cost state (l, C), the cost of a state (l, u) is given by $C(u)$.

Definition 4 (Cost Function). *A cost function $C : \mathbb{R}^{\mathbb{C}} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ assigns to each clock valuation, u, a positive real valued cost, c, or infinity. The support $sup(C) = \{u \mid C(u) < \infty\}$ is the set of valuations mapped to a finite cost.*

Table 1 summarizes several operations that are used by the symbolic semantics and the algorithm in Fig. 1. In terms of the support of a cost function, the operations behave exactly as on zones; e.g.: $sup(r(C)) = r(sup(C))$. The operations effect on the cost value reflect the intent to compute the minimum cost of reaching a state, e.g., $r(C)(u)$ is the infimum of $C(v)$ for all v that reset to u.

2.3 Symbolic Semantics

The symbolic semantics for LPTA is very similar to the common zone based symbolic semantics used for timed automata.

Definition 5 (Symbolic Semantics). *Let $A = (L, l_0, E, I, P)$ be a linearly priced timed automaton. The symbolic semantics is defined as a labelled transition system over symbolic cost states on the form (l, C), l being a location and C a cost function with the transition relation:*

- $(l, C) \xrightarrow{\epsilon} \Big(l, I(l)\Big(delay\big(I(l)(C), P(l)\big)\Big)\Big)$,
- $(l, C) \xrightarrow{a} \Big(l', I(l)\big(r(g(C))\big) + p\Big)$ *iff* $l \xrightarrow{g,a,r} l'$, *and* $p = P((l, g, a, r, l'))$.

The initial state is (l_0, C_0) where $sup(C_0) = \{u_0\}$ and $C_0(u_0) = 0$.

Notice that the support of any cost function reachable by the symbolic semantics is a zone.

Lemma 1. *Given LPTA A, for each trace α of A that ends in state (l, u), there exists a symbolic trace β of A, that ends up in a symbolic cost state (l, C), such that $C(u) = cost(\alpha)$.*

Lemma 2. *Whenever (l, C) is a reachable symbolic state and $u \in sup(C)$, then $mincost(l, u) \leq C(u)$ for all u.*

Theorem 1. *$mincost(l) = \min\{min(C) \mid (l, C)$ is reachable$\}$*

Theorem 1 ensures that the algorithm in Fig. 1 indeed does find the minimum cost, but since the state-space is still infinite there is no guarantee that the algorithm ever terminates. For zone based timed automata model checkers, termination is ensured by normalizing all zones with respect to a maximum constant M [Rok93], but for LPTA ensuring termination also depends on the representation of cost functions.

2.4 Representing Cost Functions

As stated in the introduction, we provide an efficient implementation of cost functions for the class of Uniformly Priced Timed Automata (UPTA).

Definition 6 (Uniformly Priced Timed Automata). *An uniformly priced timed automaton is an LPTA where all locations have the same rate. We refer to this rate as the rate of the UPTA.*

Lemma 3. *Any UPTA A with positive rate can be translated into an UPTA B with rate 1 such that $mincost(l)$ in A is identical to $mincost(l)$ in B.*

Thus, in order to find the infimum cost of reaching a satisfying state in UPTA, we only need to be able to handle rate zero and rate one.

In case of rate zero, all symbolic states reachable by the symbolic semantics have very simple cost functions: The support is mapped to the same integer (because the cost is 0 in the initial state and only modified by the increment operation). This means that a cost function C can be represented as a pair (Z, c), where Z is a zone and c an integer, s.t. $C(u) = c$ when $u \in Z$ and ∞ otherwise. Delay, reset and satisfaction are easily implementable for zones using DBMs. Increment is a matter of incrementing c and a comparison $(Z_1, c_1) \sqsubseteq (Z_2, c_2)$ reduces to $Z_2 \subseteq Z_1 \wedge c_1 \leq c_2$. Termination is ensured by normalizing all zones with respect to a maximum constant M.

In case of rate one, the idea is to use zones over $\mathbb{C} \cup \{\delta\}$, where δ is an additional clock keeping track of the cost, s.t. every clock valuation u is associated with *exactly one* cost $Z(u)$ in zone Z[3]. Then, $C(u) = c$ iff $u[\delta \mapsto c] \in Z$. This is possible because the continuous cost advances at the same rate as time. Delay, reset, satisfaction and infimum are supported directly by DBMs. Increment $C + c$ translates to $Z[\delta \mapsto \delta + k] = \{u[\delta \mapsto u(\delta) + k] \mid u \in Z\}$ and is also realizable using DBMs. For comparison between symbolic cost states, notice that $Z_2 \subseteq Z_1 \Rightarrow Z_1 \sqsubseteq Z_2$, whereas the implication in the other direction does not hold in general, see Fig. 3. However, it follows from the following Lemma 4 that comparisons can still be reduced to set inclusion provided the zone is extended in the δ dimension, see Fig. 3.

Lemma 4. *Let $Z^\dagger = \{u[\delta \mapsto u(\delta) + d] \mid u \in Z \wedge d \in \mathbb{R}_{\geq 0}\}$. Then $Z_1 \sqsubseteq Z_2 \Leftrightarrow Z_2^\dagger \subseteq Z_1^\dagger$.*

[3] We define $Z(u)$ to be ∞ if u is not in Z.

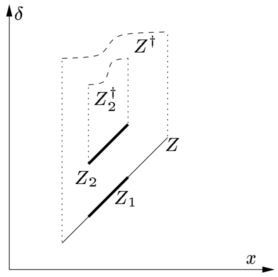

Fig. 3. Let x be a clock and let δ be the cost. In the figure, $Z \sqsubseteq Z_1 \sqsubseteq Z_2$, but only Z_1 is a subset of Z. The $()^\dagger$ operation removes the upper bound on δ, hence $Z_2^\dagger \subseteq Z^\dagger \Leftrightarrow Z \sqsubseteq Z_2$.

It is straightforward to implement the $()^\dagger$-operation on DBMs. However, a useful property of the $()^\dagger$-operation is, that its effect on zones can be obtained without implementing the operation. Let $(l_0, Z_0^\dagger)$, where Z_0 is the zone encoding C_0, be the initial symbolic state. Then $Z = Z^\dagger$ for any reachable state (l, Z) — intuitively because δ is never reset and no guards or invariants depend on δ.

Termination is ensured if all clocks except for δ are normalized with respect to a maximum constant M. It is important that normalization never touches δ. With this modification, the algorithm in Fig. 1 will essentially encounter the same states as the traditional forward state-space exploration algorithm for timed automata, except for the addition of δ.

3 Improving the State-Space Exploration

As mentioned, the major drawback of the algorithm in Fig. 1 is that it requires the entire state-space to be searched before the minimum cost of reaching a goal state can be declared. In this section we will discuss a number of possibilities for improving this in some cases.

3.1 Minimum Cost Order

In realizing the algorithm of Fig. 1, and in analogy with Dijkstra's algorithm for finding the shortest path in a directed weighted graph, we may choose always to select a (symbolic cost) state (l, C) from Waiting for which C has the smallest minimum cost. With this choice, we may terminate the algorithm as soon as a goal state is selected from Waiting. We will refer to this strategy as the Minimum Cost order (MC order).

Lemma 5. *Using the MC order, an optimal solution is found by the algorithm in Fig. 1 when a goal state is selected from* Waiting *the first time.*

When applying the MC order, the algorithm in Fig. 1 can be simplified since the variable Cost is not needed any more. Again in analogy with Dijkstra's shortest path algorithm, the MC ordering finds the minimum cost of reaching a goal state with guarantee of its optimality, in a manner which requires exploration of a *minimum number* of symbolic cost states.

Lemma 6. *Using the algorithm in Fig. 1, it can never reduce the number of explored states to prefer exploration of a symbolic cost state of* WAITING *with non-minimal minimum cost.*

In situations when WAITING contains more than just one symbolic cost state with smallest minimum cost, the MC order does not offer any indication as to which one to explore first. In fact, for exploration of the symbolic state-space for timed automata without cost, we do not know of a definite strategy for choosing a state from WAITING such that the fewest number of symbolic states are generated. However, any improvements gained with respect to the search-order strategy for the state-space exploration of timed automata will be directly applicable in our setting with respect to the strategy for choosing between symbolic cost states with same minimum cost.

3.2 Using Estimates of the Remaining Cost

From a given state one often has an idea about the cost remaining in order to reach a goal state. In branch-and-bound algorithms this information is used both to delete states and to search the most promising states first. Using information about the remaining cost can also decrease the number of states searched before an optimal solution is reached.

For a state (l, u) let $rem((l, u))$ be the minimum cost of reaching a goal state from that state. In general we cannot expect to know exactly what the remaining cost of a state is. We can instead use an estimate of the remaining cost as long as the estimate does not exceed the actual cost. For a symbolic cost state (l, C) we require that $\mathrm{REM}(l, C)$ satisfies $\mathrm{REM}(l, C) \leq \inf\{rem((l, u)) \mid u \in sup(C)\}$, i.e. $\mathrm{REM}(l, C)$ offers a lower bound on the remaining cost of all the states with location l and clock valuation within the support of C.

Combining the minimum cost $min(C)$ of a symbolic cost state (l, C) with the estimate of the remaining cost $\mathrm{REM}(l, C)$, we can base the MC order on the sum of $min(C)$ and $\mathrm{REM}(l, C)$. Since $min(C) + \mathrm{REM}(l, C)$ is smaller than the actual cost of reaching a goal state, the first goal state to be explored is guaranteed to have optimal cost. We call this the MC+ order but it is also known as Least-Lower-Bound order. In Section 4 we will show that even simple estimates of the remaining cost can lead to large improvements in the number of states searched to find the minimum cost of reaching a goal state.

One way to obtain a lower bound is for the user to specify an initial estimate and annotate each transition with updates of the estimate. In this case it is the responsibility of the user to guarantee that the estimate is actually a lower bound in order to ensure that the optimal solution is not deleted. This also allows the user to apply her understanding and intuition about the system.

3.3 Heuristics and Bounding

It is often useful to quickly obtain an upper bound on the cost instead of waiting for the minimum cost. In particular, this is the case when faced with a state-space too big for the MC order to handle. As will be shown in Section 4, the techniques described here for altering the search order using heuristics are very useful. In addition, techniques from branch-and-bound algorithms are useful for improving the upper bound once it has been found.

Applying knowledge about the goal state has proven useful in improving the state-space exploration [RE99,HLP00], either by changing the search order from the standard depth or breadth-first, or by leaving out parts of the state-space.

To implement the MC order, a suitable data-structure for WAITING would be a priority queue where the priority is the minimum cost of a symbolic cost state. We can obviously generalize this by extending a symbolic cost state with a new field, *priority*, which is the priority of the state used by the priority queue. Allowing various ways of assigning values to *priority* combined with choosing either to first select a state with large or small priority opens for a large variety of search orders.

Annotating the model with assignments to *priority* on the transitions, is one way of allowing the user to guide the search. Because of its flexibility it proves to be a very powerful way of guiding the search. The assignment works like a normal assignment to integer variables and allows for the same kind of expressions.

When searching for an error state in a system a *random* search order might be useful. We have chosen to implement what could be called *random depth-first order* which as the name suggests is a variant of a depth-first search. The only difference between this and a standard depth-first search is that before pushing all the successors of a state on to WAITING (which is implemented as a stack), the successors are randomly permuted.

Once a reachable goal state has been found, an upper bound on the minimum cost of reaching a goal state has been obtained. If we choose to continue the search, a smaller upper bound might be obtained. During state-space exploration the cost never decreases therefore states with cost bigger than the best cost found in a goal state cannot lead to an optimal solution, and can therefore be deleted. The estimate of the remaining cost defined in Section 3.2 can also be used for pruning exploration of states since whenever $min(C) + \textsc{Rem}(l, C)$ is larger than the best upper bound, no state covered by (l, C) can lead to a better solution than the one already found.

All of the methods described in this section have been implemented in UPPAAL. Section 4 reports on experiments using these new methods.

4 Experiments

In this section we illustrate the benefits of extending UPPAAL with heuristics and costs through several verification and optimization problems. All of the examples have previously been studied in the literature.

4.1 The Bridge Problem

The following problem was proposed by Ruys and Brinksma [RB98]. A timed automaton model of this problem is included in the standard distribution of UPPAAL[4].

Four persons want to cross a bridge in the dark. The bridge is damaged and can only carry two persons at the same time. To cross the bridge safely in the darkness, a torch must be carried along. The group has only one torch to share. Due to different physical abilities, the four cross the bridge at different speeds. The time they need per person is (one-way) 25, 20, 10 and 5 minutes, respectively. The problem is to find a schedule such that all four cross the bridge within a given time. This can be done with standard UPPAAL. With the proposed extension, one can also find the best possible time for the solders to cross the bridge.

We compare four different search orders: Breadth-First (BF), Depth-First (DF), Minimum Cost (MC) and an improved Minimum Cost (MC+). In this example we choose the lower bound on the remaining cost, REM(C), to be the time needed by the slowest person, who is still on the "wrong" side of the bridge.

Table 2 shows the number of states explored and the cost found for the first and the optimal solution. The third column shows the number of states explored and the cost when states are deleted based on the estimate of the remaining cost (this does not apply to MC and MC+ because the search stops when the first solution is found). As can be seen from the table, only about 10% of the states searched to find an initial solution using breadth first order is needed for the MC+ order to find the optimal solution.

Table 2. Bridge problem by Ruys and Brinksma.

	Initial Solution		**Optimal Solution**		**With est. remainder**	
	states	cost	states	cost	states	cost
BF	4491	65	4539	60	4493	60
DF	169	685	25780	60	5081	60
MC	1536	60	1536	60	N/A	N/A
MC+	404	60	404	60	N/A	N/A

4.2 Job Shop Scheduling

A well known class of scheduling problems are the Job Shop problems. The problem is to optimally schedule a set of *jobs* on a set of *machines*. Each job is a chain of operations, usually one on each machine, and the machines have a limited capacity, also limited to one in most cases. The purpose is to allocate starting times to the operations, such that the overall duration of the schedule, the *makespan*, is minimal.

[4] The distribution can be obtained at `http://www.uppaal.com`.

In this section, we apply UPPAAL to 25 of the smaller Lawrence Job Shop problems.[5] Our models are based on the timed automata models in [Feh99a]. In order to estimate the lower bound on the remaining cost, we calculate for each job and each machine the duration of the remaining operations. The final estimate of the remaining cost is then estimated to be the maximum of these durations. Table 3 shows results obtained for the first 15 problems for the search orders DF, Random DF, and a combined heuristic. The latter is based on depth-first but takes also into account the remaining operation times and the lower bound on the cost, via a weighted sum. We also tried using BF and MC order, but we did not obtain any results even if we allow MC order to search for more than 30 minutes using more than 2Gb of memory no solution is found. With the MC+ order we could only find solutions to la05 and la14 exploring 9791 and 10653 states respectively.

As can be seen from the table UPPAAL are handling the first 15 examples quite well. For the 10 largest problems (la16 to la25) we did not find optimal solutions though in some cases we were very close to the optimal solution. Since branch-and-bound algorithms generally do not scale too well when the number of machines and jobs increase, this is not surprising. The branch-and-bound algorithm for [AC91], who solves about 10 out the 15 problems in the same setting, faces the same problem. Note that the results of this algorithm depend sensitively on the choice of an upper bound. Also the algorithm used in [BJS95], who combines a good heuristic with an efficient branch and bound algorithm and thus solves all of these 15 instances, can not find solutions for instances with 15 jobs and 10 machines or larger. It is important to notice that the combined heuristic used includes a clever choice between states with the same values of cost plus remaining cost. This is the reason it is able to outperform the MC+ order.

Table 3. Results for the smaller 15 job shop problems with 5 machines and 10 jobs (la1-la5), 15 jobs (la6-la10) and 20 jobs (la11-la15). The table shows the best solution found by different search orders within 60 seconds cputime on a Pentium II 300 MHz. If the search terminated also the number of explored states is given. The last row gives the makespan of an optimal solution.

problem	instance	la01	la02	la03	la04	la05	la06	la07	la08	la09	la10	la11	la12	la13	la14	la15
DF	cost	2466	2360	2094	2212	1955	3656	3410	3520	3984	3681	4974	4557	4846	5145	5264
	states	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RDF	cost	842	806	769	783	696	1076	1113	1009	1154	1063	1303	1271	1227	1377	1459
	states	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
comb.	cost	666	672	626	639	593	926	890	863	951	958	1222	1039	1150	1292	1289
heur	states	292	-	-	-	284	480	-	400	425	454	642	633	662	688	-
minimal	makespan	666	655	597	590	593	926	890	863	951	958	1222	1039	1150	1292	1207

[5] These and other benchmark problems for Job shop scheduling can be found on `ftp://ftp.caam.rice.edu/pub/people/applegate/jobshop/`.

4.3 The Sidmar Steel Plant

Proving schedulability of an industrial plant via a reachability analysis of a timed automaton model was firstly applied to the SIDMAR steel plant, which was included as case study of the Esprit-LTR Project 26270 VHS (Verification of Hybrid Systems). The plant consists of five machines placed along two tracks and a casting machine where the finished steel leaves the system. The two tracks and the casting machine are connected via two overhead cranes on one track. Each quantity of raw iron enters the system in a ladle and depending on the desired steel quality undergoes treatments in the different machines of different durations. The aim is to control the plant in particular the movement of the ladles with steel between the different machines, taking the topology of the plant into consideration.

We use a model based on the models and descriptions in [BS99,Feh99b,HLP99]. A full model of the plant that includes all possible behaviors was however not immediate suitable for verification. Using BF or DF search it was impossible to generate a schedule for a model with only three ladles. Priorities can be used to influence the search order of the state space, and thus to improve the results. Based on a depth-first strategy, we reward transitions that are likely to serve in reaching the goal, whereas transitions that may spoil a partial solution result in lower priorities.

A schedule for three ladles was produced in [Feh99b] for a slightly simplified model using Uppaal. In [HLP99] 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 in [Sto00], who uses constraint programming to schedule 30 ladles. All these works only consider ladles with the same quality of steel and the initial solutions cannot be improved.

Using a search order based on the priorities we can generate a schedule for ten ladles, compared to two without priorities, with varying qualities of steel within 60 seconds cputime on a Pentium II 300 MHz. The initial solution found is improved by 5% within the time limit. Importantly, in this approach we do not rule out optimal solutions. Allowing the search to go on for longer, models with more ladles can be handled.

4.4 Pure Heuristics: The Biphase Mark Protocol

The Biphase Mark protocol is a convention for transmitting strings of bits and clock pulses simultaneously as square waves. This protocol is widely used for communication in the ISO/OSI physical layer; for example, a version called "Manchester encoding" is used in the Ethernet. The protocol ensures that strings of bits can be submitted and received correctly, in spite of clock drift, jitter and filtering by the channel. A formal parameterized timed automaton model of the Biphase Mark Protocol was given in [Vaa00]. We will use the corresponding Uppaal models to investigate the benefits of heuristics in pure reachability analysis.

Table 4. Results for nine erroneous instances of the Biphase Mark protocol. Numbers of state explored before reaching an error state

	nondetection mark subcell			sampling early			sampling late		
	(16,3,11)	(18,3,10)	(32,3,23)	(16,9,11)	(18,6,10)	(32,18,23)	(15,8,11)	(17,5,10)	(31,16,23)
breadth first	1931	2582	4049	990	4701	2561	1230	1709	3035
`in==1` heuristic	1153	1431	2333	632	1945	1586	725	1039	1763

The three parameters in the model are the size of the mark and code cell of the sending process and the size of the sampling distance at the receiver. Basically, for each bit send, two points needs to be read for the receiver to interpret the bit correctly. Three kinds of errors can occur: the 'middle point' (called mark subcell) is missed, the end point is sampled too early or too late. Two of the three errors occur only if input "1" is offered to the receiver, and the third error can occur in any case. Therefore we will guide the model to make a breadth first search but only in the part of the state-space where a "1" is send. Table 4 shows the number of states searched in order to find the error in three erroneous instances of the protocol. Using the heuristic almost halves the number of states searched before the error is found.

5 Conclusion

On the preceding pages, we have contributed with (1) a cost function based symbolic semantics for the class of linearly priced timed automata; (2) an efficient, zone based implementation of cost functions for the class of uniformly priced timed automata; (3) an, in some sense, optimal search order for finding the minimum cost of reaching a goal state; and (4) experimental evidence that these techniques can lead to dramatic reductions in the number of explored states. In addition, we have shown that it is possible to quickly obtain upper bounds on the minimum cost of reaching a goal state by manually guiding the exploration algorithm using priorities.

References

[AC91] D. Applegate and W. Cook. A Computational Study of the Job-Shop Scheduling Problem. *OSRA Journal on Computing 3*, pages 149–156, 1991.

[BDM+98] 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 the 10th Int. Conf. on Computer Aided Verification*, number 1427 in Lecture Notes in Computer Science, pages 546–550. Springer–Verlag, 1998.

[BFH+01] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. Accepted for Hybrid Systems: Computation and Control, 2001.

[BJS95] P. Brucker, B. Jurisch, and B. Sievers. Code of a Branch & Bound Algorithm for the Job Shop Problem. Available at url `http://www.mathematik.uni-osnabrueck.de/research/OR/`, 1995.

[BS99] R. Boel and G. Stremersch. Report for VHS: Timed Petri Net Model of Steel Plant at SIDMAR. Technical report, SYSTeMS Group, University Ghent, 1999.

[Dil89] D. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 197–212. Springer–Verlag, 1989.

[Feh99a] A. Fehnker. Bounding and heuristics in forward reachability algorithms. Technical Report CSI-R0002, Computing Science Institute Nijmegen, 1999.

[Feh99b] A. Fehnker. Scheduling a steel plant with timed automata. In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA99)*, pages 280–286. IEEE Computer Society, 1999.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybird Systems. In Orna Grumberg, editor, *Proc. of the 9th Int. Conf. on Computer Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 460–463. Springer–Verlag, 1997.

[HLP99] T. Hune, K. G. Larsen, and P. Pettersson. Guided synthesis of control programs using UPPAAL for VHS case study 5. VHS deliverable, 1999.

[HLP00] T. Hune, K. G. Larsen, and P. Pettersson. Guided Synthesis of Control Programs Using Uppaal. In Ten H. Lai, editor, *Proc. of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation*, pages E15–E22. IEEE Computer Society Press, April 2000.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Diagnostic Model-Checking for Real-Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 575–586. Springer–Verlag, October 1995.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[NTY00] P. Niebert, S. Tripakis, and S. Yovine. Minimum-time reachability for timed automata. In *IEEE Mediteranean Control Conference*, 2000. Accepted for publication.

[NY99] P. Niebert and S. Yovine. Computing optimal operation schemes for multi batch operation of chemical plants. VHS deliverable, May 1999. Draft.

[RB98] T. C. Ruys and E. Brinksma. Experience with Literate Programming in the Modelling and Validation of Systems. In Bernhard Steffen, editor, *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, number 1384 in Lecture Notes in Computer Science (LNCS), pages 393–408, Lisbon, Portugal, April 1998. Springer-Verlag, Berlin.

[RE99] F. Reffel and S. Edelkamp. Error Detection with Directed Symbolic Model Checking. In *Proc. of Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 195–211. Springer–Verlag, 1999.

[Rok93] T. G. Rokicki. *Representing and Modeling Digital Circuits*. PhD thesis, Stanford University, 1993.

[Sto00] M. Stobbe. Results on scheduling the sidmar steel plant using constraint programming. Internal report, 2000.

[Vaa00] F. Vaandrager. Analysis of a biphase mark protocol with Uppaal. to appear, 2000.