# MINIMUM AND MAXIMUM DELAY PROBLEMS IN REAL-TIME SYSTEMS

*Costas Courcoubetis* *

Computer Science Institute, FORTH
and
Computer Science Dept.
University of Crete

*Mihalis Yannakakis*

AT&T Bell Laboratories
Murray Hill, NJ 07974

*ABSTRACT*

We consider a finite state system with a finite number of clocks, where the transitions may depend on the values of the clocks, and may reset some of the clocks. We address the complexity and provide algorithms for the following problems. Suppose that the system starts from a given current state with a given assignment of values to the clocks. Can a given target state ever appear in the history of the system? What is the earliest time it can appear? What is the latest time it can appear?

## 1. Introduction

The use of computers to control and interact with physical processes is rapidly growing as computing becomes faster and cheaper. An important characteristic which distinguishes such applications from traditional ones, is their real-time aspect. Real-time programs such as airplane controllers, real-time operating systems, switching software and process controllers in manufacturing plants are inherently reactive, and their interaction with the environment must occur in real-time. The correct operation of such systems is more than logical consistency in terms of event sequences, and extends to the satisfaction of "hard" real-time constraints: for example, it is not enough for the flight control to eventually react to an obstacle in the course of the plane, it must do this *on time*! Other systems in which the explicit notion of time plays an important role is communication systems and particularly communication protocols; the performance of such systems vitally depends on the value of timers, which control message retransmission. If the setting of these timers is incorrect and does not take into account the range of the round-trip message delay, the performance of the system will degrade due to unnecessary retransmissions. We should also mention the case of digital circuits, where the timing in the interaction of their components is crucial for their correct operation.

* Work partially supported by the BRA ESPRIT project SPEC

It is intuitively obvious that correctness of real-time systems is more subtle and harder than for traditional systems. One reason for this is that the parameter time is not discrete, does not range over a finite domain, and has its own dynamics beyond the control of the programs. This makes the verification of such systems a challenging and sometimes impossible task. Although for discrete time systems there has been extensive progress in the area of automatic verification, only recently there has been progress in the case of their real-time counterpart. There have been a number of models and logics to reason about real-time systems proposed in [AD90], [ACD90], [Di89], [Le90], [AH89]. Algorithms for the automatic verification of such systems have been proposed in [ACD90], [ACD91]. Correctness was defined in terms of the histories of a real-time system satisfying certain properties which explicitly depend on time. An important issue these papers did not address concerns the derivation of bounds for the time at which different events can occur. For example, although there are algorithms to check that "a message can arrive before time $t$", the problem of determining the maximum (or minimum) such time was still open. This is the type of problem we solve in this paper.

Our model of a real-time system is a *timed graph*, as introduced in [ACD90]. These graphs model "finite-state" real-time systems. Such a system has a finite set of states and a finite set of real-valued clocks. A clock can be reset simultaneously with any transition of the system. At any instant, the value of a clock is equal to the time elapsed since the last time this clock was reset. The real-time information is given in terms of enabling conditions of the edges; a transition is enabled if the values of the clocks satisfy a certain predicate. Formally, a timed graph is a tuple $(S, E, C, \pi, \tau)$, where $S$ is a finite set of states, $E \subset S \times S$ is the set of edges, $C$ is a finite set of clocks, $\pi : E \rightarrow 2^C$ tells which clocks should be reset with each transition, and $\tau$ is a function labeling each transition with an enabling condition built using the boolean connectives over the atomic formulas of the form $x \leq d$ and $d \leq x$, where $x \in C$, and $d \in N$ ($N$ denotes the non-negative integers). We could generalize our approach to include enabling conditions of the form $x - y \leq d$ and $d \leq x - y$, where $x, y \in C$, and $d \in N$; we will not do that for keeping the presentation simple. Note also that comparing the value of the clocks with integers is as powerful as comparing them to rationals, since we could always define a unit of time sufficiently small with respect to which the rational constraints will turn into integer ones. An example of a timed graph can be found in figure 1.

Although the semantics of this model will be formally described in the following section, we can describe them intuitively as follows. The system starts in some initial state $s_0$ with some initial clock assignment. The values of the clocks increase uniformly with time. At any point in time, the system can make a transition if the associated condition is enabled by the current values of the clocks. The transitions are instantaneous. With each transition $e$, the clocks in $\pi(e)$ get reset to 0 and start counting time with respect to the time the transition occurred. At any point in time, the complete configuration of the system is described by specifying the current state and the values of the clocks. Clearly, such a system has uncountably many configurations. A *real-time trajectory* is a function giving the value of the complete configuration of this system as a function of time.

We will examine the following problems in the context of timed graphs.

*The Timed Graph Reachability Problem:* Given a timed graph $G$, some initial state $s_0$, some initial clock assignment $v$, and some final state $s_f$, determine if $s_f$ appears in some real-time trajectory of $G$ starting from configuration $s_0, v$. A variant of this problem is when we are not given an initial clock assignment, but we want to determine whether we can reach $s_f$ starting from $s_0$ for *some* initial clock assignment $v$.

*The Minimum Delay Problem in Timed Graphs:* Given a timed graph $G$, some initial state $s_0$, some initial clock assignment $v$, and some reachable final state $s_f$, how fast can we reach $s_f$ starting from the configuration $s_0, v$? Technically, there may be no "best" path; i.e., it is possible that every path can be shortened by an infinitesimal amount, so to be precise, we seek the greatest lower bound on the time $t$ that it takes for all real-time trajectories starting from $s_0, v$ to reach (some configuration with) state $s_f$.

*The Maximum Delay Problem in Timed Graphs:* Given a timed graph $G$, some initial state $s_0$, some initial clock assignment $v$, and some final state $s_f$, find the least upper bound on the time $t$ at which any real-time trajectory of the system visits a configuration with state $s_f$. In both the maximum and the minimum delay problems we are also interested in the variants where we are not given an initial clock assignment $v$, but rather we wish to optimize over all possible $v$.

The example in figure 1 illustrates these problems. One can easily see that state $s_4$ is not reachable, that state $s_3$ can only appear during the open interval $(2, 3)$, and that state $s_2$ can appear at any time larger than 2.

The reachability problem can be solved using the techniques of [ACD90]. From a timed graph $G$, they show how to construct an ordinary graph, called the *region graph*, which provides a finitary representation of the system preserving the reachability properties of interest. The region graph has size polynomial in the number of states and edges of the timed graph, but exponential in (1) the number of clocks, and (2) the (binary) length of the constants that appear in the enabling conditions of the timed graph. We show that the reachability problem is PSPACE-complete. Furthermore, this holds even if the constants are small but there are many clocks, or there are few clocks but the constants are large, which indicates that both exponential dependencies are unavoidable. Our main positive results are efficient algorithms for the minimum and the maximum delay problems. We show that both problems can be solved in time that is essentially linear in the size of the region graph and the length of the initial clock assignment $v$. *

The paper is organized as follows. In Section 2 we give definitions and notation and review the concept of the region graph from [ACD90]. In Section 3 we prove the lower bounds for the simple reachability problem. In Sections 4 and 5 respectively we address the minimum and the maximum delay problems. Finally in Section 6 we offer some concluding remarks and address the remaining open problems.

## 2. Preliminaries

Let $G = (S, E, C, \pi, \tau)$ be a timed graph. We let $\Gamma(G)$ denote the set of *time assignments* for the clocks of $G$, i.e., the set of mappings from the set $C$ of clocks to the set of non-negative reals. We represent a configuration of the system by the tuple $<s,v>$, where $s \in S$ and $v \in \Gamma(G)$. In what follows, we will use the above notation and the terminology "configuration of $G$" to refer to a configuration of the real-time system modelled by $G$. Also we denote by $v_x$ the value of the clock $x$ in the clock assignment $v$. Let $v \in \Gamma(G)$ and $t \in R$. Then $v + t$ denotes the time assignment for the clocks which assigns to each $y \in C$ the value $v_y + t$, and $[x \rightarrow t]v$ denotes the time assignment for the clocks in $C$ which assigns $t$ to the clock $x$ and agrees with $v$ on the values of the rest of the clocks.

* Assumming that $v$ is rational. Our algorithms work also in the case that the initial clock assignment $v$ is real, in th usual model of infinite precision real arithmetic.

A *real-time trajectory* of $G$ starting from the configuration $<s_0,v_0>$ is a sequence of triplets $(s(i), v(i), t(i))$, $i=0, 1, \cdots$, where $s(i) \in S$ is a state of the timed graph, $v(i) \in \Gamma(G)$ is a time assignment to the clocks, $t(i) \in R$ is a real time satisfying the following conditions:

(a) $s(0) = s_0$, $v(0) = v_0$, $t(0) = 0$,

(b) the time of the $i$+1th transition is greater or equal to the time of the $i$th transition, $t(i+1) \geq t(i)$, $i=0,1, \cdots$,

(c) $e_i = (s(i), s(i+1))$ is an edge in $E$, and the time assignment $(v(i) + t(i+1) - t(i))$ satisfies the enabling condition $\tau(e_i)$,

(d) the time assignment $v(i+1)$ at time $t(i+1)$ equals $[\pi(e_i) \rightarrow 0](v(i) + t(i+1) - t(i))$.

(e) Every time is eventually reached; i.e., for every $t \in R$ there is an $i$ such that $t(i) > t$.

From this definition, such a trajectory gives us all the information we need in order to construct a complete evolution of our system as a function of continuous time. We can think of a trajectory to be the embedding of a system continuous time behaviour at the times at which a transition occurs. Note that this definition allows more than one transitions to occur in the same time. That is, the time of the clocks is stopped, and the system can perform instantaneously several transitions which are enabled one after the other; each transition is enabled by the clock assignment which the previous one produced. Our results hold also in the model where this is not allowed; i.e., if the inequality in condition (b) is strict.

From the above discussion it follows that we can think of trajectories as being defined over continuous time. Let $<s(t), v(t)>$, $t \geq 0$, be such a continuous time version of a trajectory $(s(i), v(i), t(i))$, $i=0, 1, \cdots$. If $t(i) < t(i+1) = \cdots = t(i+k) = t < t(i+k+1)$, we define $s(t)$ to be the list $s(i+1), \ldots, s(i+k)$, we let $v(t)$ be the list $v(i+1), \ldots, v(i+k)$, and define the function *last* such that $last(a_1, \ldots, a_k) = a_k$. Then if $t(j) \leq t < t(j+1)$ we define $s(t) = last(s(j))$ and $v(t) = last(v(j)) + t$. In the rest of the paper we will use both versions to represent trajectories, depending from the context. We will say that the trajectory $<s(t), v(t)>$ *hits* at time $t_1$ some configuration $<s, v>$ if $s(t_1) = s$ and $v(t_1) = v$, or $s$ and $v$ are corresponding elements of the lists $s(t_1)$ and $v(t_1)$ in case the trajectory makes several transitions at time $t_1$.

For each clock $x \in C$ we let $c_x$ be the largest constant to which $x$ is compared in any enabling condition of a transition of $G$. If $t$ is a real number, we use $fract(t)$ to denote its fractional part. Given two clock assignments $v, v' \in \Gamma(G)$, we say that they are *equivalent* ($v \simeq v'$) if the following two conditions are met:

(a) For each $x \in C$, either the integral part of $v_x$ and $v'_x$ are the same, or both $v_x$ and $v'_x$ are greater than $c_x$.

(b) For every $x, y \in C$ such that $v_x \leq c_x$ and $v_y \leq c_y$, we have that $fract(v_x) \leq fract(v_y)$ iff $fract(v'_x) \leq fract(v'_y)$, and that $fract(v_x) = 0$ iff $fract(v'_x) = 0$.

We denote by $[v]$ the equivalence class of $\Gamma(G)$ to which $v$ belongs. Consider the following example for a timed graph $G$ with $c_x = 2$ and $c_y = 1$. The equivalence classes are shown in figure 2. They correspond to corner points (e.g. (1,1)), open line segments (e.g. $\{(x,y) \mid 0<x<1$ and $x=y\}$, $\{(x,1) \mid x>2\}$), and open regions (e.g. $\{(x,y) \mid 0<x<y<1\}$, $\{(x,y) \mid 1<x<2$ and $y>1\}$).

We call an equivalence class $\alpha$ a *boundary class* if it lies on a hyperplane $v_i = d$; thus, for any $v \in \alpha$ and any $t > 0$, $v$ and $v + t$ are not equivalent. An equivalence class is *open* if it is not a boundary class. For an equivalence class $\alpha$ we define its *successor* class *succ* $(\alpha)$ to be the equivalence class $\beta$ with the following property. Consider the clocks starting at some

arbitrary assignment $v \in \alpha$ at time 0. As time elapses, the value of the clocks $v(t) = v + t$ will eventually switch from $\alpha$ to a different equivalence class $\beta$. Then $\beta = succ(\alpha)$. Note that *succ* is defined and is unique for all classes except the *end class*, the equivalence class satisfying $x > c_x$ for all clocks $x \in C$, that does not have a successor.

The following property concerning equivalent clock assignments is proved in [ACD90]. If $v \simeq v'$, then for any trajectory starting from a configuration $<s, v>$ there is another trajectory starting from $<s, v'>$ going through the same sequence of states and equivalent clock assignments, and with its transitions times occurring "almost" at the same time with the corresponding transitions of the first trajectory. This motivates the definition of the region graphs.

We define a *region* as a pair $<s,[v]>$, where $s \in S$, and $[v]$ is an equivalence class of clock assignments. We also call a region $<s,[v]>$ a *boundary (open)* region if $[v]$ is a boundary (open) equivalence class. We can think of the region as denoting a set of system configurations; they all have the same state component, and their clock assignment is in the corresponding equivalence class. The *region graph* $R(G)$ corresponding to a timed graph $G$ is a graph $(V, M)$ defined over the set $V$ of all possible regions, and its edge set $M$ consists of two types of edges:

(a) Edges representing the passage of time ("time edges"); each vertex $<s, [v]>$ such that $[v]$ is not an end class, has an edge to $<s, succ([v])>$.

(b) Edges representing the transitions of $G$ ("transition edges"); each vertex $<s, [v]>$ has for each edge $e = (s, s') \in E$ an edge to $<s', [[\pi(e) \rightarrow 0]v]>$, provided that $v$ satisfies the enabling condition $\tau(e)$.

One can easily see that the number of equivalence classes of $\Gamma(G)$ induced by the above equivalence relation $\simeq$ is bounded above by $|C|!\, 2^{|C|} \prod_{y \in C}(2c_y + 2)$. From this and the construction of $R(G)$ it follows that $|V| = O(|S|\, |C|!\, \prod_{y \in C} c_y)$, and $|M| = O((|S| + |E|)\, |C|!\, \prod_{y \in C} c_y)$.

The following lemma states the basic relation between trajectories of $G$ and paths of $R(G)$ [ACD90].

**Lemma 1:** (1) For every trajectory $(s(i), v(i), t(i)), i = 0, 1, \cdots$ of the timed graph $G$, the corresponding sequence $<s(i), [v(i)]>, i = 0, 1, \cdots$, is a path in the region graph $R(G)$. (2) For every path in the region graph $R(i) = <s(i), \alpha(i)>, i = 0, 1, \cdots$ and for every configuration $<s(0), v(0)>$ in $R(0)$, there is a trajectory $(s'(i), v(i), t(i)), i = 0, 1, \cdots$, in $G$ starting from that configuration, such that $s'(i) = s(i)$, and $[v(i)] = \alpha(i)$ for all $i = 0, 1, \cdots$.

## 3. The Timed Graph Reachability Problem

This problem reduces to an ordinary reachability problem in the corresponding region graph.

**Proposition 1:** There is a trajectory of the timed graph $G$ starting from a configuration $<s_0, v>$ that hits the state $s_f$ if and only if in the region graph $R(G)$ there is a path from the node $<s_0, [v]>$, to a node with first component $s_f$.

*Proof:* It is a direct consequence of Lemma 1.

□

Thus, we can determine whether a configuration $<s_0, v>$ of $G$ can reach a state $s_f$ by computing the set of nodes of the region graph that are reachable from the node $<s_0, [v]>$. If we are not given an initial clock assignment $v$, then we just add a new node $u$ to the region graph, add arcs to all the nodes with first component $s_0$, and compute the nodes reachable from $u$.

**Corollary 1:** The timed graph reachability problem can be solved in time linear in the size of the region graph, thus in time $O(|E| \cdot |C|! \prod_{y \in C} c_y)$.

Alur, Courcoubetis and Dill used the region graph to derive an algorithm for model checking for a timed CTL logic [ACD90]. They also proved that model checking for general (complicated) formulas in this logic is PSPACE-complete. We show that even the basic reachability problem is hard. The following two theorems indicate that both sources of the exponential complexity in Corollary 1, namely, many clocks and large constants $c_y$, are apparently inherent.

**Theorem 1:** The timed graph reachability problem, restricted to instances with "small" constants (say, the constants are given in unary), is PSPACE-complete.

*Sketch:* The reduction is from the LBA acceptance problem. Given an LBA (Linear Bounded Automaton) $M$, and an input $x$, we construct a timed graph $G$ with two distinguished states $s_0$ and $s_f$ such that $M$ accepts $x$ iff there is a trajectory that reaches $s_f$ starting from $s_0$ with the all zero clock assignment, iff there is a such a trajectory with an arbitrary initial clock assignment. The constants that appear in the enabling conditions of the transitions are 1 and 2. The states of $G$ record the state and the head position of the LBA. There is one timer $x_i$ for every cell of $M$. Assume without loss of generality that the tape alphabet of $M$ is $\{1,2\}$. A move of the LBA is simulated by the following cycle. At the beginning each timer $x_i$ has value 1 or 2 equal to the symbol in the $i$th cell. Reset "almost" all the 2's in zero time (by a sequence of instantaneous transitions); after one time unit, reset again "almost" all the 2's in zero time; let one time unit pass to complete the cycle and start the new cycle. By "almost all", we mean all except possibly the timer $x_i$ corresponding to the cell where the tape head is, which is reset as follows: if the new symbol written in the $i$th tape cell is 2, then we reset the timer $x_i$ in the first phase but not the second; if the new symbol is 1 then we reset $x_i$ in the second phase. (A similar construction works if we cannot perform more than one instantaneous transitions one after the other, by using constants smaller than $n$.)

□

**Theorem 2:** The reachability problem for timed graphs with three timers is PSPACE-complete.

*Sketch:* We reduce again from the LBA acceptance problem. Assume a tape alphabet of $\{0,1\}$, and view a tape as a binary number. A move of the LBA is simulated by a cycle at the beginning of which a certain timer $x_1$ has value equal to the value of the tape. There is a part of the construction that decodes the appropriate bit of $x_1$ that corresponds to the current head position in order to determine the next move and update the state and the contents accordingly. The main problem is that this has to be done while time is running and the timer is changing. The two auxiliary timers are used to pass the value back and forth so that we do not lose track of the tape contents. We defer the details to the full paper.

□

## 4. The Minimum Delay Problem

We are given a timed graph $G$ with set of clocks $C = \{x_1, \ldots, x_k\}$, a initial clock assignment $v_0 = t_1, \ldots, t_k$, an initial state $s_0$, and a final state $s_f$. Let $F$ be the set of times at which the system can be in state $s_f$; that is, $F$ is the set of times $w$ such that there is a trajectory $<s(t), v(t)>$, starting from $s(0) = s_0$, $v(0) = v_0$, such that $s(w) = s_f$. We would like to compute $T_{\min} = infimum(F)$. We solve this problem by solving the more general problem, where instead of specifying a state $s_f$ one specifies an arbitrary region $R$. Clearly if we solve this more general problem for all regions containing the state $s_f$ (there are finitely many of them), then the minimum of these solutions will be the solution of the original problem.

We solve this more general minimum time problem by reducing it to a shortest path problem in a ordinary weighted graph $G'$. Then, for efficiency reasons, we will transform it further to another graph $G''$. We proceed as follows.

*Construction of $G'$:* The states $V'$ consist of the regions in $R(G)$ with the addition of a source state $R_s$ corresponding to the initial configuration $<s_0, v_0>$. Let $R_0$ be the region that contains the initial configuration $<s_0, v_0>$. The edges of $G'$ are constructed as follows.

1. Every transition edge $R \rightarrow R'$ of the region graph is present in $G'$ with length 0. Also we have an edge from $R_s$ to the region $R_0$.
2. A time edge $R \rightarrow R'$ of the region graph is present in $G'$ iff $R$ is a boundary region (hence $R'$ must be an open region); the edge has length $\varepsilon \ll 1$, which we treat as a symbol standing for an arbitrarily small positive number.
3. If $R$ and $R'$ are both boundary regions with the same timer, say $x_i$, equal to a constant $c$ in $R$ and $c'$ in $R'$, where $c' > c$, and if there is a path in the region graph from $R$ to $R'$ which does not reset the clock $x_i$, then we include an edge $R \rightarrow R'$ in $G'$ with length $c' - c$. (It suffices actually to include these only for the case $c' = c + 1$.) Also, if the clock $x_i$ does not have constant value over the region $R_0$, and there is a path in the region graph from $R_0$ to $R'$ that does not reset the clock $x_i$, then we include an edge in $G'$ from $R_s$ to $R'$ with length $c' - t_i$. (Again, it suffices to include these edges only if $c'$ is equal to the integral part of $t_i$ plus one.)

Let $F_R$ denote the set of times that must elapse in order for the system to hit some configuration in the region $R$, starting from state $<s_0, v_0>$. Let $d(R, \varepsilon)$ denote the minimum distance in $G'$ from $R_s$ to $R$ (a function of $\varepsilon$), and let $d(R)$ be the above minimum distance with $\varepsilon$ set to 0 in $G'$. Then the following holds.

**Proposition 2:** The infimum of $F_R$ is $d(R)$. Furthermore, the infimum is achieved, i.e., there exist a trajectory which reaches $R$ in exactly $d(R)$ time units, if and only if $d(R, \varepsilon)$ does not depend on $\varepsilon$ (and thus, is equal to $d(R)$).

*Proof:* Omitted.

□

We can compute the minimum delay $T_{\min}$ to reach a specified state $s_f$ by adding a new node $R_f$ with zero length arcs from all regions with first component $s_f$, and computing the shortest path from $R_s$ to $R_f$. The case where an initial clock assignment is not specified, can be solved by a similar simple modification.

Proposition 2 leads to a (possibly) quadratic algorithm in the size of the region graph. The reason is that even though the region graph is "sparse" (all nodes have small degree), the graph $G'$ may be dense due to the edges that are included by part 3 of the construction; also determining these edges involves some form of transitive closure computation.

We can modify $G'$ as follows to obtain a graph $G''$. We include again one node for each region and the source node $R_s$. In addition, for every region $R$ and clock $i$ that is not constant in $R$ we include a node $(R,i)$. We include the edges of parts 1 and 2 in the construction of $G'$. Instead of the edges of part 3, we have the following edges:

a. $R \rightarrow (R',i)$ of length 1 if $x_i$ is constant in $R$ and $R' = succ(R)$;

b. $(R,i) \rightarrow (R',i)$ of length 0 if $R \rightarrow R'$ is an edge of the region graph that does not reset timer $i$;

c. $(R,i) \rightarrow R'$ of length 0 if $x_i$ is constant in $R'$ and $R' = succ(R)$.

d. $R_s \rightarrow (R_0,i)$ if $x_i$ is not constant in the region $R_0$; the edge has length $1-fract(t_i)$.

**Proposition 3:** For every region $R$, the distance from the source node $R_s$ to $R$ in the graph $G'$ is the same as the distance in the graph $G''$.

*Sketch:* The edges of parts a-d in the construction of $G''$ simulate the edges of part 3 in the construction of $G'$.

□

If the number of clocks is $k$, and the region graph has $V$ nodes and $M$ edges, then $G''$ has roughly $kV$ nodes and $kM$ edges; note that $k$ is in general much smaller than $V$ and $M$ (at most logarithmic). The best general bound for computing single source shortest paths on such a graph has complexity $O(kM+kVlogV)$. However, except for the edges coming out of $R_s$ all other edges have length 0, 1, or ε. We can take advantage of this to obtain linear time in the size of $G''$.

**Theorem 3:** Let $k$ be the number of clocks and $M$ the size of the region graph. Then the minimum delay problem can be solved in time $O(kM)$.

## 5. The Maximum Delay Problem

The formulation is similar to the one for the minimum delay problem: We are given again a timed graph $G$ with set of clocks $C = \{x_1, \ldots, x_k\}$, a initial clock assignment $v_0 = t_1, \ldots, t_k$, an initial state $s_0$, and a final state $s_f$. Let $F$ be the set of times at which the system can be in state $s_f$. We want to compute $T_{max}$, the least upper bound of $F$. Again, we will solve the more general problem of determining the supremum of this set for the case of a target region instead of a state. We will again reduce this problem to a longest path problem in a different graph $G'$. We do this as follows.

*Construction of $G'$:* The states $V'$ consist of the regions in $R(G)$ with the addition of a source state $R_s$ corresponding to the initial configuration $<s_0, v_0>$. We denote by $R_0$ the region that contains the initial configuration $<s_0, v_0>$. The edges of $G'$ are constructed as follows.

1. Every transition edge $R \rightarrow R'$ of the region graph is present in $G'$ with length 0.
2. If $R$ and $R'$ are both boundary regions with the same counter, say $x_i$, equal to a constant $c$ in $R$ and $c'$ in $R'$, where $c' > c$, and if there is a path in the region graph from $R$ to $R'$, then we include an edge $R \rightarrow R'$ in $G'$ with length $c' - c$. (It suffices actually to include these only for the case $c' = c + 1$, and only if there is a path in the region graph from $R$ to $R'$ which does not reset clock $x_i$.) Also, if there is a path in the region graph from $R_0$ to $R'$ and $c' > t_i$, then we include an edge in $G'$ from $R_s$ to $R'$ with length $c' - t_i$.
3. If $R'$ is an open region whose successor in time is region $R''$ that has $x_i = c'$, and $R$ is a boundary region with $x_i = c$, where $c < c'$, and there is a path in the region graph from

$R$ to $R'$, then we include the edge $R \rightarrow R'$ with weight $c' - c - \varepsilon$. Also, if there is a path in the region graph from $R_0$ to $R'$, we include the arc $R_s \rightarrow R'$ with weight $c' - t_i - \varepsilon$. If $R'$ is an end region (has no successor in time, and hence all clocks can become arbitrarily large), then add the edges of the region graph that go into $R'$, and add a self-loop edge $R' \rightarrow R'$ with weight equal to infinity (or we can treat these regions as special).

Let $F_R$ denote again the set of times that must elapse in order for the system to hit some configuration in the region $R$, starting from state $<s_0, v_0>$. Let $D(R,\varepsilon)$ denote the maximum length of a path in $G'$ from $R_s$ to $R$ (a function of $\varepsilon$); this may be $\infty$, either because it uses an edge of length $\infty$, or because it can become arbitrarily large. Let $D(R) = D(R,0)$ be the above distance with $\varepsilon$=0. Then the following holds.

**Proposition 4:** The supremum of $F_R$ is $D(R)$. Furthermore, there exists a trajectory which reaches $R$ in exactly $D(R)$ time units if and only if $D(R,\varepsilon)$ is finite and does not depend on $\varepsilon$.

*Proof:* Omitted.

□

For the above graph $G'$ we compute the maximum distance to a region $R$, $D(R,\varepsilon)$, as follows. We find the strong components (that are reachable from $R_s$), and we set $D(R,\varepsilon)$ to infinity if and only if there is a strong component $B$ that is reachable from $R_s$ and can reach $R$, which contains an edge of nonzero weight. After determining these regions, we remove them from the graph, we shrink all remaining strong components (their edges have 0 weight) to get an acyclic graph. Then we compute the longest path from the source $R_s$ to every remaining node in a standard way processing the nodes in reverse topological order.

As in the previous section, simple modifications suffice to compute the version of the problem where the destination is a state of the timed graph instead of a region, and/or no initial clock assignment is specified. We can also transform the problem to a second graph $G''$ that is "almost" unweighted and sparse, and is more suitable for efficiency purposes.

**Theorem 4:** Let $k$ be the number of clocks and $M$ the size of the region graph. Then the maximum delay problem can be solved in time $O(kM)$.

## 6. Conclusions

There is another similar problem which we did not discuss, concerning the maximum time that a state can be visited for the first time. In other words: find the smallest time $t$ (if it exists) such that if a trajectory starting from configuration $<s_0,v_0>$ has not visited state $s_f$ by time $t$, then it will never visit it in the future. This problem can also be solved by a simple variant of the construction for the case of the maximum delay problem. We postpone the details to the full paper.

An important observation is that in the case that the initial clock assignment consists of rational numbers, one can always first refine the time scale of the system so that this configuration becomes a boundary region, and then there is a way of reducing the problem to a standard reachability problem by adding an extra clock. Although this is possible and is conceptually simple, the complexity of doing this grows much more rapidly since we have to refine our time scale and hence increase the constants $c_x$, $x \in C$. As a result, this approach would lead to an algorithm that is quadratic in the region graph and exponential (instead of linear) in the length of the given initial clock assignment. We avoid doing so in our approach.

A remaining open problem is the variant of the problems we considered in this paper in which the target is not a region (or a set of regions) but a configuration of the form $<s_f, v_f>$, for some arbitrary clock assignment $v_f$.

**Acknowledgment:** We would like to thank R. Alur and D. Dill for many helpful discussions.

## References

[ACD90] R. Alur, C. Courcoubetis, D. Dill, "Model-Checking for Real-Time Systems", 5th IEEE LICS, 1990.

[ACD91] R. Alur, C. Courcoubetis, D. Dill, "Probabilistic Model Checking of Real-Time Systems", to appear in 18th ICALP, 1991.

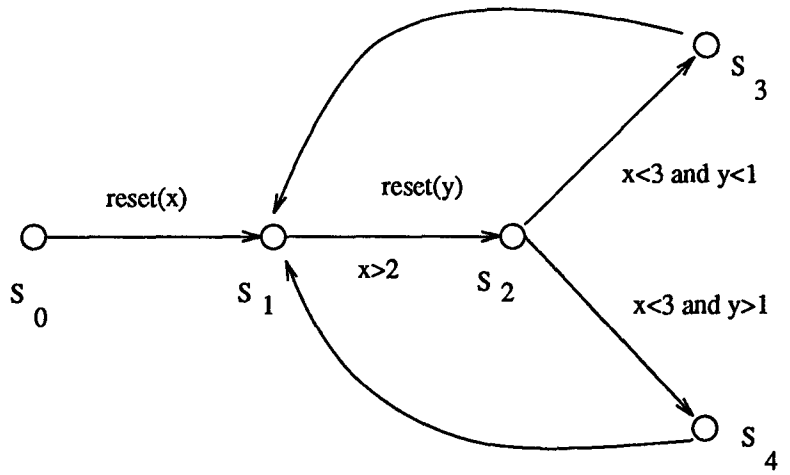
[AD90] R. Alur, D. Dill, "Automata for Modelling Real-Time Systems", 17th ICALP, 1990.

[AH89] R. Alur, T. Henzinger, "A Really Temporal Logic", 30th IEEE FOCS, 1989.

[AK83] S. Aggarwal, R. Kurshan, "Modelling Elapsed Time in Protocol Specification", *Protocol Specification, Testing and Verification*, III, 1983.

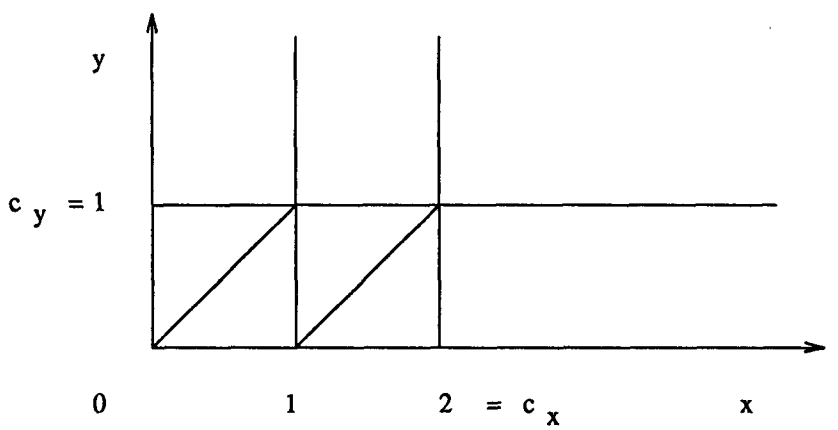
[Di89] D. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems", *Automatic Verification Methods for Finite-State Systems, LNCS* 407, 1989.

[Le90] H. Lewis, "A Logic of Concrete Time Intervals", 5th IEEE LICS, 1990.




A timed graph

Figure 1




The time regions for $c_x = 2$, $c_y = 1$

Figure 2