Min-max Computation Tree Logic

Pallab Dasgupta, P.P. Chakrabarti,
Jatindra Kumar Deka and Sriram Sankaranarayanan

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, INDIA 721302
pallab,ppchak,jatin@cse.iitkgp.ernet.in

Abstract

This paper introduces a branching time temporal query language called Min-max CTL which is similar in syntax to the popular temporal logic, CTL [8]. However unlike CTL, Min-max CTL can express timing queries on a timed model. We show that interesting timing queries involving a combination of min and max can be expressed in Min-max CTL. While model checking using most timed temporal logics is PSPACE complete or harder[1, 2], we show that many practical timing queries, where we are interested in the worst case or best case timings, can be answered in polynomial time by querying the system using Min-max CTL.

1 Introduction

Temporal logic model-checking [8] is one of the most popular and well studied paradigms for formal verification of hardware and other concurrent systems (see [9] for a survey). In this approach, each concurrent process is modeled as a finite state non-deterministic transition system. The correctness property which needs to be verified on the given set of concurrent transition systems is specified in terms of a temporal logic formula. Model checking has been extensively studied for two broad categories of temporal logics, namely *linear time temporal logic* and *branching time temporal logic* [5, 9].

Traditional temporal logics such as *Linear Temporal Logic* (LTL), *Computation Tree Logic* (CTL), and CTL* [8] can be used to reason about the temporal behavior of systems without explicitly quantifying time. These logics are therefore inadequate to quantitatively reason about timing properties of a system. On the other hand timed temporal logics such as *Timed CTL* (TCTL) [1] and *Real Time CTL* (RTCTL) [11] allow us to verify actual timing properties on a timed transition system. However, the verification of timed temporal logics has been found to be much more complex than their untimed counterparts (see [4] for a survey). For example, while LTL model checking is PSPACE complete, TLTL model checking is undecidable [3]. The problem is less severe in the case of branching time timed logics, where TCTL model checking is PSPACE complete [1, 2] (where as CTL model checking is possible in polynomial time). It has been shown in [2] that TCTL model checking is PSPACE complete even in discrete-time models. Model checking in discrete-time models has been studied by other researchers [6, 12] as well.

While analyzing timed transition systems, we often like to reason about the extremal (best case or worst case) temporal properties of the system. Determining such temporal properties can also be used as a strategy for verification [7]. For example, we may be interested in determining the worst case delay that may occur between a *request* and a *grant* in an arbiter circuit. The answer to this query also allows us to verify whether the worst case delay is bounded by some δ.

In this paper, we propose a temporal query language called Min-max CTL which allows the quantification of CTL state and path properties in terms of a cost function over real time. Min-max CTL is capable of expressing queries which involve a combination of extremal (min and max) quantifiers both along paths (or walks) in the timed model, as well as across paths. It is possible to express complex queries such as the following:

> Determine the minimum among the worst-case response times for the earliest *request* along all possible computation paths of a client process, where the worst-case response time for a *request* of a client is defined as the maximum delay by which the *grant* is given by the server.

Fig 1 shows a sample timed model, where the answer to this query is 3, since the worst-case response time of the requests at states a, b, and c, are 4, 3, and 5 respectively.

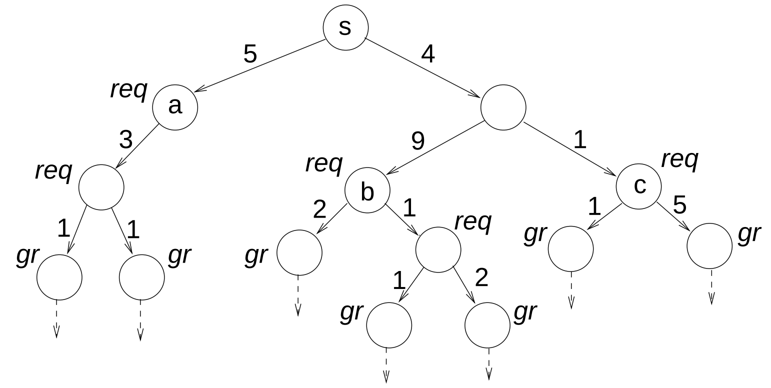

Figure 1: Timed model: Client-Server

We show that for most practical cost functions, Min-max CTL queries can be evaluated in polynomial time. The result assumes significance since model checking timed logics such as TCTL and RTCTL are PSPACE complete [1, 2]. Thus Min-max CTL captures an interesting subset of timing properties (namely most extremal timing properties) which can be evaluated in polynomial time.

The main contributions of this paper are as follows:

- We present a temporal query language called Min-max CTL which can express queries related to the worst-case and best-case timings along computation paths, as well as across computation paths.

- We show that a restriction of Min-max CTL (which covers many practical min-max timing queries of interest) can be verified in time polynomial in the number of states of the system and the length of the formula. We also show that the general problem is DP-hard.

The paper is organized as follows. In Section 2 we define the syntax and semantics of Min-max CTL, and illustrate Min-max CTL through examples. Section 3 shows that a monotonic restriction of Min-max CTL can be evaluated in polynomial time, while the general problem is DP hard. We present a polynomial time algorithm for monotonic Min-max CTL and analyze it's correctness and complexity. Section 4 presents three illustrative models of practical interest where Min-max CTL is used for reasoning about extremal timing behavior. We also present experimental results demonstrating the time and memory requirements for a prototype Min-max CTL evaluator on scaled up versions of these models. Section 5 outlines two possible extensions.

2 Syntax and Semantics

The syntax of Min-max CTL is similar to that of CTL, except for two special types of *until* operators, which we call *min-until* ($U_{\min}$) and *max-until* ($U_{\max}$) respectively, and two special types of existential (E) and universal (A) quantifiers, which (like CTL) must immediately precede each *min-until* and *max-until* path formula. These special quantifiers act as *min* or *max* operators across paths. Also, since we consider timed models, we exclude the next-time (X) operator of CTL.

2.1 Syntax

We first describe the model and the formal syntax of Min-max CTL, and then illustrate it with examples.

Definition 2.1 [Timed Model:]
A timed model is a tuple $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$*, where:*

- $\mathcal{AP}$ *is the set of atomic propositions,*
- S *is the finite set of states,*
- $\mathcal{R} \subseteq S \times S \times \mathcal{N}$ *is a transition relation, where* $\mathcal{N}$ *denotes the set of positive integers, and* $(s_i, s_j, \tau_{ij}) \in \mathcal{R}$ *implies that the delay between successive states* s_i *and* s_j *is* τ_{ij} *units of time,*
- $s_0 \in S$ *is the initial state,*
- $\mathcal{L} : S \to 2^{\mathcal{AP}}$ *is a labeling of states with atomic propositions true in that state.* □

The syntax of Min-max CTL is as follows. B denotes boolean formulas, S denotes CTL formulas, and Z denotes Min-max CTL formulas. C and C' are user defined functions.

- $B = false|true|q|\neg q|B \wedge B|B \vee B|\neg B$
- $S = B|S \wedge S|S \vee S|E(S\ U\ S)|A(S\ U\ S)|\neg S$
- $Q = \min | \max$
- $Z = Z \wedge S|QE_{C(g,h)}(SU_QZ)|QE_{C'(g)}(SU_QS)|QA_{C(g,h)}(SU_QZ)|QA_{C'(g)}(SU_QS)$

We use the following abbreviations:

Fq:	$true\ U\ q$
$F_{\min}q$:	$true\ U_{\min}\ q$
$F_{\max}q$:	$true\ U_{\max}\ q$

Throughout this paper, Z-formulas, S-formulas and B-formulas refer to formulas derived out of Z, S and B respectively. We first present a few examples to informally explain the semantics of Min-max CTL and then proceed to define the formal semantics.

In traditional CTL model checking we seek to determine whether a given CTL formula (of the form of S) is true at a state of the model. In Min-max CTL, we have extended the syntax of CTL by allowing the quantified until operators ($U_{\min}$ and $U_{\max}$) and the *min* and *max* path quantifiers, which are defined over the cost functions C or C'. Thereby Min-max CTL also has a semantics of evaluation which returns a numeric value whenever the formula is true at a state. This numeric value quantifies the value of the objective function C (or C') which we seek to optimize, and is the answer to the Min-max CTL query at that state. On the other hand, if the formula is false at a state of the model, then the evaluation of the formula at that state returns null.

A Min-max CTL formula is said to be true at a state when the *CTL restriction* of the Min-max CTL formula is true at that state. This restriction is obtained by removing the functions C, C' and the quantifiers Q and Q' from the Min-max CTL formula. We present a couple of examples to illustrate the CTL restriction of a Min-max CTL formula:

- The CTL restriction of the Min-max CTL formula $\max A_{2g}(F_{\min}q)$ is AFq.
- The CTL restriction of the Min-max CTL formula $\min E_{g+h}(pU_{\min}(\min E_g(qU_{\min}r)))$ is $E(pU(E(qUr)))$.

Min-max CTL evaluation returns a numeric *Min-max value* corresponding to the Min-max CTL formula at states where the CTL restriction of the formula is true. We first illustrate the semantics of evaluation through examples, and then define the formal semantics of the evaluation function.

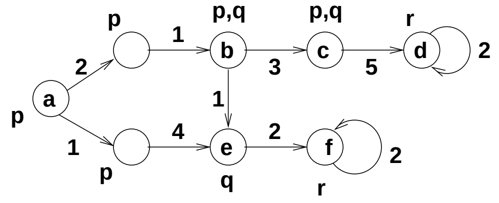

Figure 2: A timed model

- Figure 2 shows a sample timed model. Suppose we are interested in determining the earliest occurrence of a state where q is true, starting from state a. We can pose this query as $\min E_g(F_{\min}q)$ to be evaluated at state a. Using the $U_{\min}$ operator indicates that we are interested in the earliest occurrence of q along the path. Hence the path formula $F_{\min}q$ evaluates to 3 on all paths through state b. This value is the delay to reach state b from state a. We refer to the state b as the *closing state* of these paths. Similarly, for all paths which go through state e and not through state b, the path formula $F_{\min}q$ evaluates to 5, and e is the closing state. The delay from the start state of the path to the closing state is called the g-value of the path. The quantifier $\min E_g$ specifies that the path with minimum g-value has to be chosen, which in this case is any path through state b. Hence the Min-max CTL formula $\min E_g(F_{\min} q)$ evaluates to 3 at state a.

- Consider the query $\min E_{2g}(F_{\min} q)$. This query differs from the previous one only in the cost function, which is $2g$ instead of g. The best path is still the same, but the value returned at state a is now 6.

- Using the $U_{\max}$ operator, we can determine the latest occurrence of q along a path. For example, in Figure 2, evaluating the formula $\min E_g(F_{\max} q)$ at state a yields 4 and the best path is the one through state b followed by state e. State e is the closing state of the path.

- Consider the Min-max CTL formula $\varphi = \min E_{g+h}(pU_{\min}(\min E_g(qU_{\min} r)))$. Here we use the cost function $C(g,h) = g + h$. The h-value of a state with respect to φ is the Min-max value computed at that state for the subformula $\psi = \min E_g(qU_{\min} r)$. Since we use $U_{\min}$, the possible closing states for the path formula $pU_{\min}\psi$ are b (for all paths through b) and e (for all paths through e which do not go through b). Evaluation of ψ yields $h = 3$ at state b and $h = 2$ at state e. Thus for all paths where b is the closing state, $C(g,h) = g + h = 6$, and for all paths where e is the closing state, $C(g,h) = g + h = 7$. Since the path quantifier is $\min E_{g+h}$, the value returned by evaluating φ at state a is 6.

Thus the numeric values returned by the evaluation of a Min-max CTL formula f at states of the timed model are determined by the values (namely h-values) returned by evaluating the subformulas of f at the states of the model and the *lengths* (namely g-values) computed by evaluating the special until operator U_Q (which may be $U_{\min}$ or $U_{\max}$) over the paths of the model. The objective function may have two parameters, namely g and h. The g-value is associated with the path cost computed by virtue of the U_Q operator, and the h-value is associated with evaluating the subformula[1] following the U_Q operator. The user defined objective functions C and C' compute the merits of paths as a function of the g-values and h-values, and the *min* and *max* path quantifiers specifies the criterion for choosing between values returned by evaluating these paths.

2.2 Semantics

In this subsection we present the formal semantics of Min-max CTL. We first present the formal definition of the CTL-restriction of a formula.

Definition 2.2 [CTL-restriction $CR(f)$ of a formula f:]
The *CTL-restriction*, $CR(f)$ of a Min-max CTL formula, f, is the CTL formula, which is obtained by dropping the min, max quantifiers and cost functions in f. Formally,

- If f is an S-formula or B-formula, then $CR(f) = f$.
- For a formula $f = z_1 \wedge z_2$ where z_2 is an S-formula, $CR(f) = CR(z_1) \wedge z_2$.
- For a formula $f = QE_{C'(g)}(z_1 U_{Q'} z_2)$, $CR(f) = E(z_1\ U\ z_2)$.
- For a formula $f = QE_{C(g,h)}(z_1 U_{Q'} z_2)$, $CR(f) = E(z_1\ U\ CR(z_2))$.

[1]It may be noted that a Min-max CTL formula may have at a Min-max CTL subformula only after the U operator.

- For a formula $f = QA_{C'(g)}(z_1 U_{Q'} z_2)$, $CR(f) = A(z_1 \ U \ z_2)$.
- For a formula $f = QA_{C(g,h)}(z_1 U_{Q'} z_2)$, $CR(f) = A(z_1 \ U \ CR(z_2))$. □

The CTL-restriction of a Min-max CTL formula is an untimed CTL formula whose semantics can be defined in the standard way [8] over a timed model by ignoring the delays.

A *path*, π, in a timed model $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ is an infinite sequence of states, $\nu_0, \nu_1, \ldots$, such that for all i, $\nu_i \in S$ and $(\nu_i, \nu_{i+1}, \delta_{i,i+1}) \in \mathcal{R}$. ν_0 is called the starting state of π. Since the timed model has a finite set of states, one or more states of the timed model will appear multiple number of times on a path. In other words, a path (as defined here) is an infinite walk over the state transition graph. π^i denotes the suffix of π starting from the i^{th} state, ν_i, of π.

If a CTL state formula f is true in a state s we write $s \models f$. Likewise, if a CTL path formula f' is true in a path π we write $\pi \models f'$.

- $\forall s \in S, s \models True$ and $s \not\models False$
- $s \models p$ iff $p \in \mathcal{L}(s)$
- $s \models \neg p$ iff $p \notin \mathcal{L}(s)$
- $s \models \varphi_1 \wedge \varphi_2$ iff $s \models \varphi_1$ and $s \models \varphi_2$
- $s \models \varphi_1 \vee \varphi_2$ iff $s \models \varphi_1$ or $s \models \varphi_2$
- $s \models E(\varphi U \psi)$ iff there exists a path $\pi = s_0, s_1, s_2, \ldots$ starting at $s = s_0$ and some $i \geq 0$ such that $s_i \models \psi$ and for all $j < i$, $s_j \models \varphi$. We also say that $\pi \models \varphi U \psi$, that is, the path formula $\varphi U \psi$ is true in the path π.

Definition 2.3 [Closing state and g-value of a path:]
Given a CTL path formula, $f' = f_1 \ U \ f_2$, and a path $\pi = \nu_0, \nu_1, \ldots$, where $\pi \models f'$, the boolean function $isclosing(i, \pi, f')$ is defined as follows. $isclosing(0, \pi, f')$ is true iff $\nu_0 \models f_2$. For $i > 0$, $isclosing(i, \pi, f')$ is true iff $\nu_i \models f_2$ and $\forall j, 0 \leq j < i, \nu_j \models f_1$. Let $\delta_{i,i+1}$ denote the delay between ν_i and ν_{i+1}, that is, $(\nu_i, \nu_{i+1}, \delta_{i,i+1}) \in \mathcal{R}$. Given a Min-max CTL path formula $f = z_1 \ U_Q z_2$, and a path π where $\pi \models CR(f)$, the closing state, $CLS(\pi, f)$, and the g-value $GVAL(\pi, f)$ of π with respect to f are defined as follows:

- If Q is min, then let $i = \min\{j : \ isclosing(j, \pi, CR(f))\}$. Then:

$$\begin{aligned} CLS(\pi, f) &= \nu_i \\ GVAL(\pi, f) &= \begin{cases} 0 & \text{if } i = 0 \\ \sum_{j=0}^{i-1} \delta_{j,j+1} & \text{otherwise} \end{cases} \end{aligned}$$

- If Q is max, then we have two cases. If $\forall j, \exists i, i > j$, and $isclosing(i, \pi, CR(f))$ is true, then $CLS(\pi, f)$ is not well defined and $GVAL(\pi, f) = \infty$. Otherwise, let $i = \max\{j : isclosing(j, \pi, CR(f))\}$. Then:

$$\begin{aligned} CLS(\pi, f) &= \nu_i \\ GVAL(\pi, f) &= \begin{cases} 0 & \text{if } i = 0 \\ \sum_{j=0}^{i-1} \delta_{j,j+1} & \text{otherwise} \end{cases} \end{aligned}$$

□

The Min-max value of a state s in a timed model $J = \langle \mathcal{AP}, S, \mathcal{R}, s_0, \mathcal{L} \rangle$ with respect to a Z-formula f will be denoted by $EvalS(f, s)$. The semantics of $EvalS(f, s)$ is as follows.

- If $s \not\models f$, then $EvalS(f, s) = null$.
- If $f = z_1 \wedge z_2$, where $s \models f$ and (wlog) z_1 is a S-formula and z_2 is a Z-formula, then $EvalS(f, s) = EvalS(z_2, s)$.
- If $f = QE_{C(g,h)}(z_1 U_{Q'} z_2)$, or $f = QA_{C(g,h)}(z_1 U_{Q'} z_2)$ then
  1. If Q' is min then:
     - If Q is min, then:

$$\begin{aligned} EvalS(f, s) &= \min\{C(GVAL(\pi, f), EvalS(z_2, CLS(\pi, f))) \\ &\qquad \text{among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\} \end{aligned}$$

     - If Q is max, then:

$$\begin{aligned} EvalS(f, s) &= \max\{C(GVAL(\pi, f), EvalS(z_2, CLS(\pi, f))) \\ &\qquad \text{among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\} \end{aligned}$$

  2. If Q' is max, then we have a special case to consider. If in some path $\pi = \nu_0, \nu_1, \ldots$ starting from $\nu_0 = s$, we have $\forall j, \exists i, i > j$ and $isclosing(i, \pi, CR(f))$ is true, then $CLS(\pi, f)$ is not well defined. However, $GVAL(\pi, f) = \infty$. In order to have a well defined semantics, we restrict the set of admissible cost functions to those which have the following limiting behavior:

$$\lim_{g \to \infty} C(g, h) = \lim_{g \to \infty} C(g, k)$$

     where k is an arbitrary constant. In other words, we allow only cost functions where the value of $C(g, h)$ is independent of h when g approaches ∞. Under this restriction, we define $EvalS(f, s)$ as follows.
     - If Q is min, then:

$$\begin{aligned} EvalS(f, s) &= \min\{C(GVAL(\pi, f), EvalS(z_2, \eta)) \\ &\qquad \text{among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\} \end{aligned}$$

       where $\eta = CLS(\pi, f)$ if $CLS(\pi, f)$ is well defined, and some arbitrary constant k otherwise.
     - If Q is max, then:

$$\begin{aligned} EvalS(f, s) &= \max\{C(GVAL(\pi, f), EvalS(z_2, \eta)) \\ &\qquad \text{among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\} \end{aligned}$$

       where $\eta = CLS(\pi, f)$ if $CLS(\pi, f)$ is well defined, and some arbitrary constant k otherwise.
- If $f = QE_{C(g)}(z_1 U_{Q'} z_2)$, or $f = QA_{C(g)}(z_1 U_{Q'} z_2)$ then
  1. If Q is min, then

$$\begin{aligned} EvalS(f, s) &= \min\{C(GVAL(\pi, f)) \\ &\qquad \text{among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\} \end{aligned}$$

2. If Q is max, then

$$EvalS(f,s) = \max\{C(GVAL(\pi,f)) \text{ among all paths } \pi = \nu_0, \nu_1, \ldots \text{ starting from } \nu_0 = s\}$$

From the above semantics it follows that $EvalS(f,s)$ returns the value returned by the cost function C for some *best* value path π, where *best* is either the minimum cost or the maximum cost. We shall refer to the set of *best* value paths with respect to $EvalS(f,s)$ as $BestP(f,s)$.

The following example illustrates the semantics of Min-max CTL.

Example 2.1 We illustrate the syntax and semantics of Min-max CTL through some examples on the timed models shown in Fig 3, Fig 4 and Fig 5. In these figures the transition delays are shown on the edges, and the labeling of states with the set of atomic propositions true in them are shown besides the states.

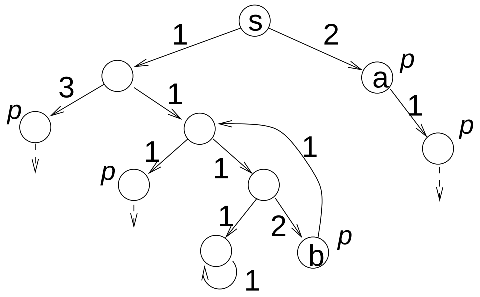

Figure 3: Sample timed model - 1

Sample timed model–1: Consider the timed model shown in Fig 3, and the following Min-max CTL formulas:

$$\begin{aligned} f &= \min E_g(F_{\min}\ p) \\ y &= \max E_g(F_{\min}\ p) \\ z &= \min A_g(F_{\min}\ p) \end{aligned}$$

In each of these formulas the cost function is $C(g) = g$ and the *until* operator is of the *min* type. In other words we are interested in finding out the delay to the earliest state where p is true on each path.

- For evaluating f at s, we require the minimum among these delays across all paths starting at s. Therefore, *EvalS(f,s)* $=$ *2* and the optimal path is the one leading to state a.
- For evaluating y at s, we require the maximum among these delays across all paths starting at s. Therefore, *EvalS(y,s)* $=$ *5* and the optimal path is the one leading to state b.
- The CTL-restriction z' of z is $z' = AFp$. It is easy to see that $s \not\models z'$. Thus $s \not\models z$, and *EvalS(z,s)* is null.

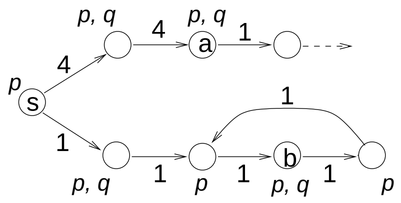

Figure 4: Sample timed model - 2

Sample timed model–2: Consider the timed model shown in Fig 4, and the following Min-max CTL formulas:

$$
\begin{aligned}
f &= \min E_g(p\ U_{\max}\ q) \\
y &= \max E_g(p\ U_{\max}\ q)
\end{aligned}
$$

In both these formulas the cost function is $C(g) = g$ and the *until* operator is of the *max* type. In other words in each path we are interested in finding out the delay to the latest state where q is true and p is true in all previous states.

Consider the set of paths which include state b. It is easy to see that the latest state where q is true along each of these paths is not at a finite delay. In fact, since the state b is in an infinite loop (where each state is labeled p), there is no well defined latest state where q is true. However, since for every ϵ, we have a q-state at a delay greater than ϵ, we choose to evaluate the distance to the *so called* last state as ∞.

- For evaluating f at s, we require the minimum among the maximum delays across all paths starting at s. Therefore, *EvalS(f,s)* $=$ *8* and the optimal path is the one leading to state a.
- For evaluating y at s, we require the maximum among the maximum delays across all paths starting at s. Since all paths through state b have q at infinite delay (and p on all previous states), *EvalS(y,s)* $= \infty$.

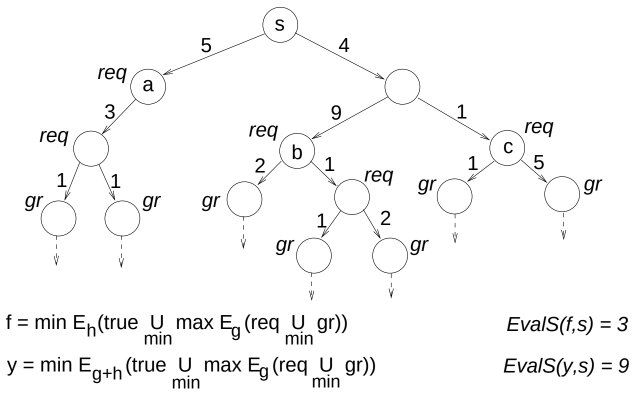

Figure 5: Sample timed model - 3

Sample timed model–3: Consider the timed model shown in Fig 5, and the following Min-max CTL formulas:

$$
\begin{aligned}
f &= \min E_h(F_{\min} \max E_g(\text{req}\ U_{\min}\ \text{gr})) \\
y &= \min E_{g+h}(F_{\min} \max E_g(\text{req}\ U_{\min}\ \text{gr}))
\end{aligned}
$$

Both of these formulas have the subformula $z = \max E_g(\text{req } U_{\min} \text{ gr})$. At each state t where $t \models z$ (that is, the CTL-restriction $z' = E(\text{req } U \text{ gr})$, of z is true at t), we get a non-null value for *EvalS(z,t)*. For example, in Fig 5, *EvalS(z,a) = 4*, *EvalS(z,b) = 3*, and *EvalS(z,c) = 5*.

- For evaluating f at s, note that the cost function is $C(g,h) = h$, that is, the cost function is independent of the distance g to the state t where z is true. On the other hand the cost depends only on the value h evaluated at the state t by *EvalS(z,t)*. Thus *EvalS(f,s) = 3*, and the optimal path goes through state b.
- For evaluating y at s, note that the cost function is $C(g,h) = g + h$, that is, we require to minimize the sum of the h-value evaluated by *EvalS(z,t)* and the delay to the state t. It is easy to see that *EvalS(f,s) = 9*, and the optimal path goes through state a.

□

The problem of evaluating the minimum and maximum delays between state sets satisfying given properties have been studied in the context of measuring performance deviation in systems [7]. Min-max CTL presented here, provides a more general framework for specifying a wide variety of extremal timing queries embedded in the elegant syntactic structure of CTL. Thereby it allows the composition of nested extremal timing queries using the basic reachability operators. Min-max CTL provides operators to compose extremal properties along paths, as well as across paths. This paper also provides efficient polytime algorithms for evaluating Min-max CTL queries.

In the following section, we shall show that for certain types of cost functions, C, the evaluation of Min-max CTL formulas is DP-hard. However, for a large class of practically relevant cost functions, the evaluation can be done in time polynomial in the number of states of the system and the length of the formula. This is comparable to the complexity of pure CTL model checking.

3 Complexity of Evaluation

In this section, we analyze the complexity of evaluating Min-max CTL formulas over timed models. We first show that the problem is DP-hard in general. Then, we show that for most practical restrictions the problem is solvable in polynomial time.

Theorem 3.1 *Evaluating a Min-max CTL query on a state of a timed model is DP-hard in general.*
Proof: We reduce the EXACT KNAPSACK problem to this problem to show that it is DP-hard. The EXACT KNAPSACK problem is defined as follows. We are given a set of N items, $i_1, i_2, \ldots, i_N$, where the item i_j has a weight, w_j and a profit p_j. Given a sack of capacity M, we are required to determine whether the maximum profit possible (by filling the sack with items without exceeding the sack capacity) is exactly P. The EXACT KNAPSACK problem is known to be DP-complete[13], and can be easily shown to be DP-complete even when $p_j = w_j$ for each item i_j.

Given an instance of EXACT KNAPSACK where each $p_j = w_j$, we create an instance of Min-max CTL query evaluation as follows. Consider the timed model shown in Fig 6. The

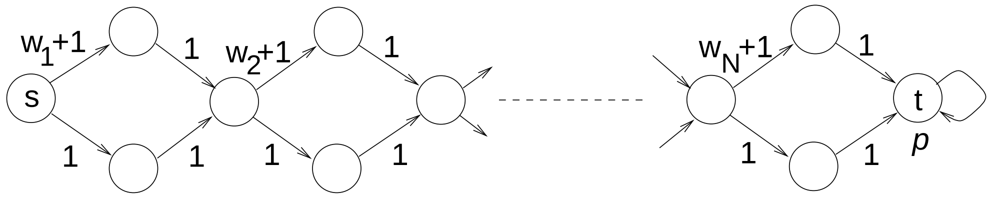

Figure 6: Timed model equivalent to EXACT KNAPSACK

transition delays are shown on the edges, and state t is labeled with the atomic proposition p. We are required to evaluate the following formula, f, at state s:

$$f = \max E_{C(g)}(F_{\min} p) \quad \text{where} \quad C(g) = \begin{cases} g & \text{if } g \leq M + 2N \\ 0 & \text{otherwise} \end{cases}$$

Each path from s through t represents a selection of items whose weights get added in the path delay g. The cost function, $C(g)$, returns a non-zero value for only those paths where the delay g is bounded by $M + 2N$ (that is, the total weight of the selected items is bounded by M). Clearly $EvalS(f, s)$ returns $P^* + 2N$, where P^* is the maximum profit possible. Checking whether P^* is equal to the given value P, yields the answer to the EXACT KNAPSACK problem. Hence EXACT KNAPSACK reduces to Min-max CTL query evaluation. □

Though the complexity of Min-max CTL evaluation is hard in general, there are important special cases which cover most of the practical queries of interest and can be evaluated in polynomial time. We now define one such category.

Definition 3.1 [Monotonic Min-max CTL:]
A function $f(x)$ is said to be monotonically increasing iff $f(a) > f(b)$ whenever $a > b$. The function is said to be monotonically decreasing iff $f(a) < f(b)$ whenever $a > b$. Also a function $f(x, y)$ is said to be monotonically increasing (decreasing) with respect to x, iff for each constant k, the function $f(x, k)$ is monotonically increasing (decreasing). A Min-max CTL formula is said to be monotonic iff its cost function $C(g, h)$ (or $C(g)$) is monotonically increasing or decreasing with respect to g, and each of it's subformulas are monotonic. Monotonic Min-max CTL is the language consisting of monotonic Min-max CTL formulas only.
□

Many temporal queries of practical interest can be expressed in Monotonic Min-max CTL. For example, all the queries shown in Example 2.1 are Monotonic Min-max CTL queries. We shall now show that Monotonic Min-max CTL can be evaluated in polytime. Throughout this paper, we assume that the cost functions specified in the Monotonic Min-max CTL formulas can be computed in constant time. Therefore they do not contribute significantly to the complexity of the algorithm.

Definition 3.2 [f-path and f-cycle:]
A path, π, starting at a state s and going through a state s' is called a "f-path from s through s'" iff the state formula f holds in all states preceding s' in π. A f-cycle through a state t is a f-path from t through t. □

A shortest length f-path from a state s through a state s' is one where s' occurs as early as in any other f-path from s through s'. The longest length f-path from s through s' is defined similarly, where s' occurs at least as late as any other f-path from s through s'. Obviously, a shortest length f-path will have no f-cycles. Hence shortest length f-paths can be found using any standard shortest path algorithms on the state transition graph in polynomial time.

If any f-path from s through s' contains an intermediate state which is in a f-cycle, then it is possible to have f-paths of infinite length from s through s', and hence the length of the longest f-path from s through s' is ∞. Determining the set of states which belong to f-cycles can be done in polynomial time. Finding whether there exists any f-path from s through s' via any of these states can also be done in polynomial time. If no such f-path exists, then by dropping all states where f is false, we are left with finding a longest length path from s through s' in an acyclic graph, which is also solvable in polynomial time.

The algorithm which we describe is a labeling algorithm. A state, s, in the timed model is labeled by a sub-formula, f, iff its CTL restriction is true in that state. Further, if the sub-formula is a Min-max CTL formula, then the evaluation algorithm augments the label with the value $EvalS(f, s)$.

Algorithm Evaluate(f,s)

1. Use CTL model checking techniques to label the states of the model with the sub-formulas of the CTL restriction of f. During this step, we ignore the delays on the transitions of the timed model.

2. The evaluation at a state, s, of a Min-max CTL formula, $f = QE_C(f_1 U_{Q'} f_2)$, where Q and Q' are min or max quantifiers, is done as follows. *The procedure for evaluating formulas of the form $f = QA_C(f_1 U_{Q'} f_2)$ is exactly similar.*

   2.1 Recursively evaluate all Min-max CTL subformulas of f at all states of the model.

   2.2 If Q' is max then define $\varphi = f_1$ else define $\varphi = f_1 \wedge \neg f_2$.

   2.3 Let H denote the set of states labeled f_2, which are reachable from s along φ-paths.

   2.4 If Q' is max then:
   2.4.1 Remove each state n from H such that:
   (a) n does not belong to any φ-cycle, and
   (b) For each successor n' of n, $n' \models A(f_1 U f_2)$.
   2.4.2 Label a state n from H with the symbol ∞ if:
   (a) n belongs to a φ-cycle, and
   (b) For each successor n' of n, $n' \models A(f_1 U f_2)$.

   2.5 For each state $t \in H$ do:
   2.5.1 If t is labeled ∞ then set $g = \infty$ and compute $W(t) = C(g, h)$, where $h = EvalS(f_2, t)$.
   2.5.2 Else Consult Table 1 to determine the required path type from s to t, determine the length g of that path, and compute $W(t) = C(g, h)$, where $h = EvalS(f_2, t)$.
   2.6 If Q is min, set $EvalS(f, s) = \min\{W(t) | t \in H\}$ else set $EvalS(f, s) = \max\{W(t) | t \in H\}$.

C-Type	Q-Type	Q'-Type	Best path type from s to t
Increasing	min	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Increasing	min	max	Shortest length f_1-path
Increasing	max	min	Longest length $(f_1 \wedge \neg f_2)$-path
Increasing	max	max	Longest length f_1-path
Decreasing	min	min	Longest length $(f_1 \wedge \neg f_2)$-path
Decreasing	min	max	Longest length f_1-path
Decreasing	max	min	Shortest length $(f_1 \wedge \neg f_2)$-path
Decreasing	max	max	Shortest length f_1-path

Table 1: Best path types for $f = QE_C(f_1 U_{Q'} f_2)$

We prove the correctness of the algorithm with respect to Monotonic Min-max CTL formulas. We establish the correctness of evaluation for the formula $f = QE_C(f_1 U_{Q'} f_2)$. The correctness of evaluation for A formulas follows from the fact that the evaluation procedure for E formulas and A formulas are essentially the same.

Lemma 3.1 *If Q' is* min, *then a state, t, which cannot be reached from s by a $(f_1 \wedge \neg f_2)$-path is not a closing state of any path in BestP(f,s).*
Proof: If t cannot be reached from s by a f_1-path, then by definition (semantics of Min-max CTL), t cannot be a closing state. If t can be reached from s by f_1-paths, but not by $f_1 \wedge \neg f_2$-paths, then every f_1-path from s through t has an intermediate state where f_2 is true. Since Q' is min, that intermediate state is the closing state. □

Lemma 3.2 *If Q' is* max, *then a state, t, which cannot be reached from s by a f_1-path is not a closing state of any path in BestP(f,s).*
Proof: If t cannot be reached from s by a f_1-path, then by definition (semantics of Min-max CTL), t cannot be a closing state. □

Lemma 3.3 *If Q' is* max *and t is a state which does not belong to any f_1-cycle, and for each successor t' of t, we have $t' \models A(f_1 U f_2)$, then t is not a closing state of any path in BestP(f,s).*
Proof: Since for each successor t' of t, $t' \models A(f_1 U f_2)$, it follows that any f_1-path from s through t in BestP(f,s) will also be an f_1-path from s through some state s' where f_2 holds, and t occurs earlier than s'. Since t does not belong to any f_1-cycle, $s' \neq t$. Since Q' is max, t cannot be a closing state. □

Lemma 3.4 *If Q' is* max *and t is a state such that there exists a f_1-path from s through t, $t \models f_2$, t belongs to a f_1-cycle, and for each successor t' of t, we have $t' \models A(f_1 U f_2)$, then t cannot be a closing state in any path having finite g-value, and there exists a path from s through t having g-value as ∞.*
Proof: Consider a path, π, having finite g-value. Then there exists a state s' which is the closing state of π. Clearly $s' \neq t$, since for each successor t' of t, $t' \models A(f_1 U f_2)$ and therefore every instance of t' is followed by some other candidate closing state.

Consider a f_1-path from s through t which repeatedly goes around in the f_1-cycle through t. In this path f_1 holds on all states and t occurs infinitely often. By definition, the g-value of such a path is ∞. $\Box$

Lemma 3.5 *If C-type is increasing, and Q-type is* min*, then any φ-path, P, from s through t which is longer than the shortest length φ-path, P^*, from s through t does not belong to BestP(f,s).*
Proof: Since C-type is increasing, the path cost of P^* is less than that of P. Since, Q-type is min, P^* is better than P and hence P cannot belong to BestP(f,s). $\Box$

Lemma 3.6 *If C-type is increasing, and Q-type is* max*, then any φ-path, P, from s through t which is shorter than the longest length φ-path, P^*, from s through t does not belong to BestP(f,s).*
Proof: Since C-type is increasing, the path cost of P^* is greater than that of P. Since, Q-type is max, P^* is better than P and hence P cannot belong to BestP(f,s). $\Box$

Lemma 3.7 *If C-type is decreasing, and Q-type is* min*, then any φ-path, P, from s through t which is shorter than the longest length φ-path, P^*, from s through t does not belong to BestP(f,s).*
Proof: Since C-type is decreasing, the path cost of P^* is less than that of P. Since, Q-type is min, P^* is better than P and hence P cannot belong to BestP(f,s). $\Box$

Lemma 3.8 *If C-type is decreasing, and Q-type is* max*, then any φ-path, P, from s through t which is longer than the shortest length φ-path, P^*, from s through t does not belong to BestP(f,s).*
Proof: Since C-type is decreasing, the path cost of P^* is greater than that of P. Since, Q-type is max, P^* is better than P and hence P cannot belong to BestP(f,s). $\Box$

Theorem 3.2 *Algorithm Evaluate correctly evaluates a Monotonic Min-max CTL formula at a state of a timed model.*
Proof: We establish the correctness of the algorithm for evaluating a Min-max CTL formula, $f = QE_C(f_1 U_{Q'} f_2)$, at a state, s, under the induction hypothesis that the algorithm correctly evaluates the subformulas f_1 and f_2 at all states of the model. Since evaluation for A formulas is exactly similar, the same proof applies to A formulas as well.

The algorithm determines the set of candidate closing states, and then proceeds to determine the g-value of the φ-path of appropriate type (longest or shortest) through the candidate closing states.

By Lemma 3.1 and Lemma 3.2, we have shown that a state can be a closing state only if it is reachable from s by a φ-path (where φ is defined in Step 2.2 of the algorithm). Therefore, in Step 2.3, we consider the set of states reachable from s by φ-paths.

By Lemma 3.3, we have shown that if Q' is max and t is a state which does not belong to any f_1-cycle, and for each successor t' of t, $t' \models A(f_1 U f_2)$, then t is not a closing state of any path in BestP(f,s). In Step 2.4.1 of the algorithm, we remove all such states from H.

By Lemma 3.4, we have shown that if Q' is max and t is a state which belongs to a f_1-cycle, and for each successor t' of t, $t' \models A(f_1 U f_2)$, then for every path (shortest or longest) through t, either the g-value of the path is ∞, or there is some other closing state. Further we have shown that there exists at least one path through such states with g-value as ∞.

Therefore, in Step 2.4.2 of the algorithm, we label such states as ∞, and in Step 2.5.1 we treat the g-values of paths through these states as ∞.

Lemma 3.5, Lemma 3.6, Lemma 3.7 and Lemma 3.8 establishes that only one path through each state in the set H needs to be considered for evaluation, and the path types are as shown in Table 1. In Step 2.5, for each state in H, the algorithm evaluates the cost $W(t)$ of the best path through t. Since these are the only candidate paths (by Lemmas 3.5-3.8), the cost of the best path among these is the desired value of $EvalS(f,s)$. In Step 2.6, the algorithm assigns the cost of the best path to $EvalS(f,s)$. □

Lemma 3.9 *The complexity of finding the length of a shortest f-path or a longest f-path from a state s to a state t in a timed model is $O(|\mathcal{R}|+|S|\log|S|)$, where $|S|$ is the number of states in the model and $|\mathcal{R}|$ is the size of the transition relation $\mathcal{R}$.*
Proof: Each f-path from s to t includes only states which are labeled f, and the state t. We first remove from the transition graph those states (except t) which are not labeled f and the set of transitions to and from these states. This can be done in $O(|\mathcal{R}|+|S|)$ time. All paths in the reduced transition graph are f-paths.

Finding the shortest path between a pair of nodes in a graph with non-negative edge costs requires $O(|\mathcal{R}|+|S|\log|S|)$ time where $|\mathcal{R}|$ denotes the number of edges in the graph[10].

For determining the longest path length, we require to consider the cycles in the graph. If we find a path from s to t through a state j which is self-reachable (that is, j belongs to a f-cycle), then the longest path length from s to t is ∞. Otherwise, we use the algorithm for acyclic graphs. This can be achieved in $O(|\mathcal{R}|+|S|)$ time. □

Theorem 3.3 *Algorithm Evaluate requires $O(|f|.|S|^2.(|\mathcal{R}|+|S|\log|S|))$ time to evaluate a Monotonic Min-max CTL formula f of length $|f|$ on a timed model $J=\langle\mathcal{AP},S,\mathcal{R},s_0,\mathcal{L}\rangle$*
Proof: Step 2.3 can be done by a single depth-first traversal in $O(|\mathcal{R}|+|S|)$ time. Step 2.4 requires us to determine whether states in H belong to any φ-cycle. Since the worst case number of states in H is $|S|$, this step can be completed in $O(|S|.(|\mathcal{R}|+|S|))$ time. By virtue of Lemma 3.9, the complexity of Step 2.5.2 is $O(|\mathcal{R}|+|S|\log|S|)$. Therefore, the total complexity of Step 2.2 to Step 2.6 is $O(|S|.(|\mathcal{R}|+|S|\log|S|))$. This is the complexity of evaluating the formula at one state when the Min-max values for the subformulas are given. The complexity of evaluating the formula at every state is given by $O(|S|^2.(|\mathcal{R}|+|S|\log|S|))$. By induction on the length of the formula, the complexity of Algorithm Evaluate is $O(|f|.|S|^2.(|\mathcal{R}|+|S|\log|S|))$. □

4 Examples

This section illustrates three problems of practical interest where Min-max CTL is useful to evaluate extremal timing properties. The first example relates to path planning, the second relates to reasoning about the controller of a motor, and the third relates to packet routing in a network of LANs. We present the models for each of these problems, and enumerate several Min-max properties of interest. We have a working prototype for a Min-max CTL evaluator, which has been used to present experimental results for larger instances of these problems showing the runtimes and memory requirements.

4.1 Travel planning

We are given the air maps, rail maps and road maps of a country. Each map shows the connections between the cities, as well as the time required on each connection. It is easy to see that the information may be represented by a three layered graph, where the layers correspond to the air, rail and road graphs respectively.

	φ_1		φ_2	
No. of Cities	Time (Sec)	Mem. (MB)	Time (Sec)	Mem. (MB)
100	0.46	0.10	0.29	0.09
1000	0.97	0.34	0.19	0.33
2000	220	0.69	0.56	0.66
3000	4.12	1.03	1.06	0.99
5000	8.58	1.72	2.73	1.66
7000	14.51	2.41	5.44	2.33
8000	17.98	2.76	6.86	2.66
10000	26.20	3.45	10.10	3.33
12000	35.73	4.14	15.00	3.99
15000	51.63	5.17	22.99	4.99
20000	79.32	6.90	38.59	6.66
25000	120.79	8.62	59.69	8.32
30000	160.16	10.35	84.23	9.99
40000	283.00	13.80	152.14	13.32
50000	421.83	17.25	232.67	16.65
75000	886.86	25.87	519.34	24.97
85000	1115.41	29.32	664.34	28.30
100000	1515.78	34.50	915.68	33.30

Table 2: Min-max Evaluator Results of Travel planner example

We can express several interesting Min-max queries for planning the mode of travel. We illustrate a few of them:

1. What is the minimum time to travel from city s to city z, when we travel by rail or road? We check the following query at s.

$$f = \min E_g((rail \vee road)\ U_{\min}\ z)$$

2. What is the minimum total time to travel from city s to city z, when we travel as far as possible by air, and then go by rail or road? We evaluate the following query at s.

$$\min E_{g+h}(air\ U_{\max}\ f)$$

   where f is the first query.

3. Suppose the cost of air travel for time t is ca(t) and the cost of rail travel for time t is cr(t). Then the minimum total cost to travel from city s to city z, when we travel as far as possible by air, and then go by rail is evaluated by:

$$\min E_{ca(g)+h}(air\ U_{\max}\ [\min E_{cr(g)}(rail\ U_{\min}\ z)])$$

Note that in this case the cost function $C(g,h)$ computes the travel cost as a function of time.

We ran our prototype Min-max evaluator on randomly generated maps over a large number of cities. The cities are numbered randomly. The following queries were evaluated on these models with the city c_i chosen randomly:

$$\begin{aligned}
\varphi_1 &= \min E_{(g+h)}(air\ U_{\min}\ rail \wedge \min E_{(g+h)}(rail\ U_{\min}\ road \wedge \min E_h(road\ U_{\min}\ c_i))) \\
\varphi_2 &= \min E_{(g+h)}(air\ U_{\max}\ rail \wedge \neg air \wedge \min E_h(rail\ U_{\min} c_i)))
\end{aligned}$$

Table 2 shows the results of evaluating these formulas at every state of the models. The Min-max evaluator was executed on a 550 Mhz Pentium-III machine with 64 MB RAM and Linux operating system.

4.2 Motor controller

Consider the controller of a motor which can operate in three different speeds. The motor can move from one operating speed to another by accelerating (or braking). The motor can also shift gears in one of its operating speeds, before accelerating or braking.

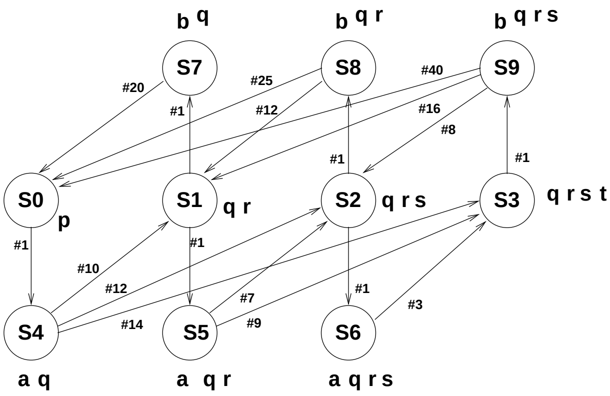

Figure 7: Transition Diagram of the Motor

The state transition diagram of the motor is shown in Fig 7. The states $S0$, $S1$, $S2$ and $S3$ are the four operating states of the motor. The motor is at rest in the state $S0$, and runs at rpms of 33, 75 and 125 respectively in the states $S1$, $S2$ and $S3$. The states $S4$, $S5$, and $S6$ are states at which the motor begins to accelerate to a higher operating speed, and the states $S7$, $S8$ and $S9$ are states at which the motor begins to decelerate to a lower operating speed. To shift gears, the motor must be brought to one of the operating speeds. Thus the path S0,S4,S1,S5,S3 denotes a sequence where the motor accelerates to the operating speed of 33 rpm (at state $S1$), shifts gears, and again accelerates to the operating speed of 125 rpm (at $S3$). The path S0,S4,S3 denotes a sequence where the motor accelerates directly to the operating speed of 125 rpm (without shifting gears). The edge delays indicate the time required for acceleration or deceleration. The time required to shift gears is 1 unit of time (shown beside the vertical edges).

We define the following atomic propositions on the states of the system.

p : the motor is at rest.
q : the motor is running.

r :	the motor is running at a rpm of 33 or more.
s :	the motor is running at a rpm of 75 or more.
t :	the motor is running at a rpm of 125.
a :	the motor is accelerating.
b :	the motor is decelerating (braking).

In Fig 7, the states are labeled by these atomic propositions. We assume that the states $S0$-$S9$ are the observable states of the motor, and our reasoning is based on the observable states only. We now enumerate several Min-max CTL queries on this model.

1. What is the quickest time by which the motor running at 75 rpm can be stopped? The following query may be evaluated at node $S2$.

$$\min E_g(F_{\min}\ p)$$

2. What is the quickest time by which the motor can be raised to a speed of 75 rpm or more and then brought to a stop? The following query may be evaluated at node $S0$.

$$\min E_{g+h}(F_{\min}\ (s \wedge \min E_g(F_{\min}\ p)))$$

3. What is the slowest time to raise the motor to a speed of 125 rpm (assuming that it does not brake in between)? To ensure that it does not brake, we pose the following Min-max CTL query at $S0$.

$$\max E_g(\neg b\ U_{\min}\ t)$$

4. What is the quickest time to raise the motor to a speed of 125 rpm if we shift gears at the operating speed of 33 rpm? We evaluate the following query at node $S0$.

$$\min E_{g+h}(F_{\min}\ ((r \wedge \neg s \wedge \neg b \wedge \neg a) \wedge \min E_g(F_{\min}\ t)))$$

5. In order to bring the motor to a stop from a speed of 125 rpm, what is the latest time when a brake may have to be applied (assuming that the motor does not accelerate in between). We evaluate the following query at state $S3$.

$$\max E_g(\neg a U_{\max} b)$$

	φ_1		φ_2		φ_3		φ_4		φ_5	
Number of Speed Levels	Time (Sec.)	Mem. (MB)	Time (Sec.)	Mem. (MB)	Time (Sec.)	Mem. (MB)	Time (Sec.)	Mem. (MB)	Time (Sec.)	Mem. (MB)
100	0.01	0.03	0.16	0.04	0.16	0.04	0.16	0.04	0.17	0.04
500	0.03	0.19	4.92	0.23	4.11	0.47	4.80	0.23	4.84	0.23
1000	0.09	0.38	19.94	0.47	16.54	1.50	20.60	0.47	20.85	0.47
1500	0.16	0.57	48.61	0.71	38.18	3.10	49.63	0.71	49.79	0.71
2000	0.26	0.76	90.35	0.95	74.46	5.26	95.99	0.95	91.93	0.95
2500	0.39	0.95	144.67	1.19	110.62	7.98	144.41	1.19	150.48	1.19

Table 3: Min-max Evaluator Results of the Motor example

We scaled up the model for the motor by increasing the number of speed levels. In practice, the discrete-time model of a motor controller can have a large number of speed levels depending on the accuracy with which it has to monitor the motor speed. We evaluated the following properties on the scaled up models. In these properties, *maxspeed* is an atomic proposition which labels the state having the maximum speed, and *middlespeed* is an atomic proposition which labels the state having half the maximum speed.

$$
\begin{aligned}
\varphi_1 &= \max E_g(\neg b\ U_{\min}\ maxspeed) \\
\varphi_2 &= \min E_{g+h}(F_{\min}\ middlespeed \wedge \min E_g(F_{\min}\ p)) \\
\varphi_3 &= \max E_g(\neg a\ U_{\max}\ b) \\
\varphi_4 &= \min E_g(F_{\min}\ p) \\
\varphi_5 &= \min E_{g+h}(F_{\min}\ middlespeed \wedge \max E_g(F_{\min}\ maxspeed))
\end{aligned}
$$

Table 3 shows the performance of the Min-max verifier for evaluating each of these queries *at every state* of the models. The first column shows the number of speed levels. The other columns show the runtimes and memory requirements for evaluating the above formulas on the models. The experimentation was done on a 550 Mhz Pentium-III machine with 64 MB RAM and Linux operating system.

4.3 Routing through a network of LANs

Fig 8 shows a network of three clusters (LANs), A, B, and C. Each cluster is represented by three switches. The hosts belonging to a given cluster (not shown in Fig 8) are connected to one of the switches in the cluster. For example, the hosts belonging to the cluster A are connected to either of the three switches $A1$, $A2$, or $A3$. Packet delays between the switches is also shown on the edges.

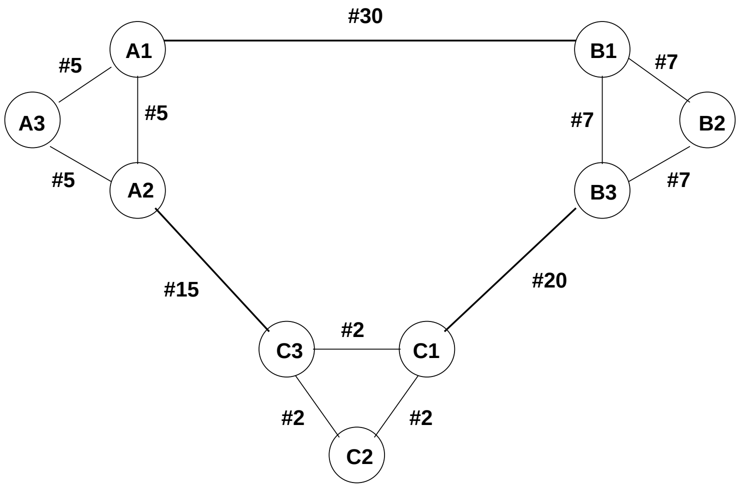

Figure 8: A Network of Clusters

The routing of packets through a network can be modeled by a non-deterministic state transition system as follows. Consider a packet with a destination d at a host a of the network. The routing tables of host a may have one or more next hosts to which packets with destination d may be forwarded. Table 4 shows the routing tables for host $A3$ and host $C1$ of the network[2] shown in Fig 8. In Table 4, DH denotes *destination host* and NH denotes the *next host* in the route path to this destination.

[2]It may be noted that the routing tables at a host may have a single entry for all hosts at a different cluster.

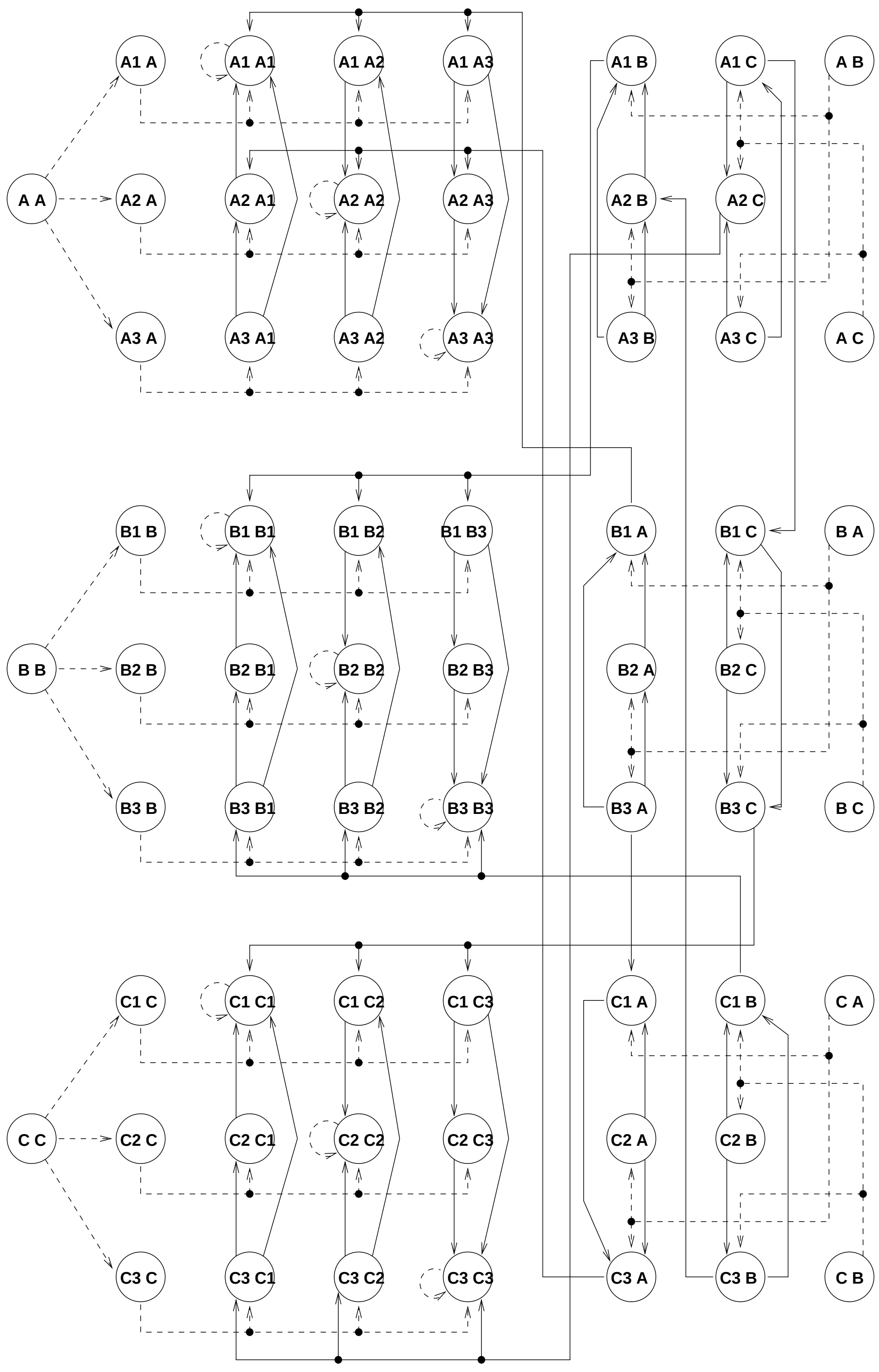

Figure 9: Transition system for packet switching

Routes at $A3$		Routes at $C1$	
DH	NH	DH	NH
A1	A1	C2	C2
A1	A2	C3	C3
A2	A2	C3	C2
A2	A1	A	C3
C	A1	B	B3
C	A2		
B	A1		
B	A2		

Table 4: Routing tables for hosts $A3$ and $C1$

The global transition relation which models the packet switching across the entire network is a collection of tuples (PH, DH, NH, τ), where PH denotes the *present host*, DH denotes the *destination host* of the packet, NH denotes the *next host*, and τ denotes the packet delay in the link from PH to NH. The tuple indicates that a packet with destination DH arriving at host PH may be forwarded to host NH at a delay of τ. Typically for a given destination, d, the transition relation will be acyclic, otherwise a packet with destination d may fall into this cycle and never reach the host d.

Fig 9 shows the transition graph of the packet switching network of Fig 8. Each node is represented by a pair (x, y), where x is the present host/cluster and y is the destination host/cluster. The source nodes of the transition graph are represented by *(x,y)* pairs (where $x, y \in \{A, B, C\}$), which indicate the originating and destination cluster of a packet. Sink nodes are nodes of the type *(y,y)*, (where $y \in \{A1, A2, A3, B1, B2, B3, C1, C2, C3\}$) that is, when a packet reaches its destination, it sinks into it. Dashed edges have zero delay[3]. A solid edge from a node (x, d) to a node (y, d) has a delay equal to the delay on the edge (x, y) in Fig 8.

A node (x, y) in the transition system has the label x. Further, a node (x, y) has the label $In(A)$ if x is a host in the cluster A of Fig 8. The node also has the label $Sink(A)$ if $x = y$ and (x, x) is a sink node in the cluster A. Labels for nodes in the clusters B and C are defined similarly.

We list several interesting Min-max CTL queries on this system.

1. What is the minimum delay to route a packet from host $B2$ to host $C3$? This query may be expressed in Min-max CTL as the following formula to be evaluated at node $(B2, C)$ of the transition system.

$$\min E_g(F_{\min}\ C3)$$

2. What is the worst case delay that a packet from a host in cluster A with a destination host in cluster C may face? To answer this query we evaluate the following Min-max CTL formula at the node (A, C) in Fig 9.

$$\max E_g(F_{\min}\ Sink(C))$$

[3]It is easy to model zero delay by a symbolic value ϵ while computing longest and shortest paths in such acyclic graphs, which can be dropped after computation.

3. What is the worst case time required for a broadcast from host $A2$ to hosts in the cluster C ? This is similar to the previous query except that we require to verify that all paths from the node $(A2, C)$ lead to sink nodes in the cluster C. We evaluate the following formula at node $(A2, C)$.
$$\max A_g(F_{\min}\ Sink(C))$$
4. If we use shortest path routing within cluster C, then what is the worst case delay that a packet from a host in cluster A with a destination host in cluster C may face? The following query is evaluated at node (A, C).
$$\max E_{g+h}(F_{\min}\ \min E_g(C\ U_{\min}\ Sink(C)))$$
5. What is the earliest delay by which a packet from host $A3$ can reach the host $B1$ if the packet is routed through cluster C? The following query is evaluated at node $(A3, B)$.
$$\min E_{g+h}(F_{\min}\ (C \wedge \min E_g(F_{\min}\ B1)))$$

We scaled up the network model to different network sizes. The network structure consists of multiple clusters of hosts, and a backbone network which connects the clusters. The connectivity within the clusters and the connectivity between clusters were randomly generated. Table 5 shows the results. The first two columns show the number of clusters and the number of hosts per cluster respectively. The third column shows the number of reachable states in the transition graph for packet routing (such as the one shown in Fig 9). The Min-max evaluator runs on this transition graph.

All states corresponding to the i^{th} cluster are labeled with cl_i. All states corresponding to the k^{th} host in a cluster are labeled with nd_k. Since the numbering of clusters and the numbering of hosts within a cluster are done randomly, there is no loss of generality when we test for the reachability of states labeled cl_j or nd_j for any given j. In the following properties, without loss of generality, we have chosen $i = n/4$, $j = n/2$, and $k = m/3$, where n is the number of clusters and m is the number of hosts per cluster.

$$\begin{aligned}
\varphi_1 &= \min E_g(F_{\min}\ (cl_i \wedge nd_k)) \\
\varphi_2 &= \max E_g(F_{\min}\ Sink(cl3)) \\
\varphi_3 &= \max E_{g+h}(F_{\min}\ \min E(cl_i\ U_{\min}\ Sink(cl_i))) \\
\varphi_4 &= \max E_{g+h}(F_{\min}\ (cl_i \wedge \min E_g(F_{\min}\ (cl_j \wedge nd_1))))
\end{aligned}$$

Table 5 shows the runtime and memory requirements for evaluating the above formulas at every state of the transition graph. The experimentation was carried out on a 550 Mhz Pentium-III machine with 64 MB RAM and Linux operating system.

5 Extensions

It is possible to extend Min-max CTL in several directions without sacrificing the computational efficiency of Min-max CTL evaluation. In this section, we describe two interesting extensions.

Adding on Conjunction and Disjunction: In Min-max CTL, we have not allowed the conjunction and disjunction of Min-max CTL state formulas. It is easy to extend Min-max CTL to allow conjunction and disjunction of Min-max CTL state formulas by

Network parameters			φ_1		φ_2		φ_3		φ_4	
No. of Clusters	Nodes/ Cluster	Reachable States	Time (Sec.)	Mem (MB)	Time (Sec.)	Mem (MB)	Time (Sec.)	Mem (MB)	Time (Sec.)	Mem (MB)
5	5	275	0.03	0.02	0.02	0.02	0.06	0.03	0.01	0.02
5	10	775	0.18	0.06	0.08	0.08	0.38	0.12	0.03	0.06
5	15	1525	0.55	0.12	0.19	0.17	1.19	0.28	0.08	0.12
5	20	2525	1.23	0.21	0.37	0.30	2.76	0.54	0.18	0.20
5	25	3775	2.42	0.31	0.69	0.49	5.42	0.93	0.35	0.31
5	30	5275	4.32	0.44	1.13	0.74	9.25	1.46	0.61	0.44
10	5	850	0.02	0.05	0.08	0.09	0.24	0.10	0.02	0.06
10	10	2100	0.07	0.15	0.47	0.32	1.53	0.35	0.12	0.17
10	15	3850	0.17	0.28	1.55	0.85	5.25	0.92	0.33	0.31
10	20	6100	0.37	0.45	3.85	1.87	13.53	1.98	0.73	0.50
10	25	8850	0.73	0.66	8.14	3.63	29.28	3.82	1.44	0.74
10	30	12100	1.25	0.91	15.00	6.45	58.04	6.72	2.53	1.02
15	5	1725	0.04	0.12	0.22	0.20	0.70	0.22	0.08	0.13
15	10	3975	0.16	0.29	1.14	0.69	3.76	0.75	0.32	0.32
15	15	6975	0.47	0.51	3.51	1.75	12.08	1.86	0.90	0.58
15	20	10725	0.98	0.80	8.14	3.70	29.80	3.90	1.90	0.90
15	25	15225	1.89	1.15	16.49	6.99	64.43	7.30	3.68	1.29
15	30	20475	3.35	1.56	29.53	12.10	122.53	12.55	6.48	1.75
20	5	2900	0.09	0.20	0.48	0.37	1.49	0.40	0.17	0.23
20	10	6400	0.36	0.47	2.26	1.25	7.84	1.34	0.69	0.53
20	15	10900	0.98	0.82	6.63	3.07	23.88	3.24	1.88	0.92
20	20	16400	2.05	1.25	14.95	6.32	56.23	6.62	4.14	1.40
20	25	22900	3.88	1.76	28.42	11.64	115.71	12.10	7.52	1.97
20	30	30400	6.75	2.36	49.94	19.75	209.09	20.41	13.05	2.64

Table 5: Min-max Evaluator Results for Network Example

appropriately defining the semantics of evaluation. For example, given two Min-max CTL formulas, f_1 and f_2, we can define the syntax of evaluation of $f = f_1 \wedge f_2$, with respect to a specified cost function C'' as:

$$EvalS(f, s) = C''\left(EvalS(f_1, s), EvalS(f_2, s)\right)$$

If $f = f_1 \vee f_2$, we can define $EvalS(f, s)$ in a similar way. However, in the second case, it is possible that $s \models f_1$ but $s \not\models f_2$. Since $s \models f_1 \vee f_2$, $EvalS(f, s)$ must return a value even though $EvalS(f_2, s)$ returns null. We believe that this difficulty can be overcome by defining appropriate semantics for evaluation with possibly null return values.

Min-max CTL *U* Min-max CTL: In Min-max CTL, we have allowed only CTL formulas for f_1 in every $f_1 U f_2$ formula, (that is, we have allowed only f_2 to be Min-max CTL). It is possible to extend Min-max CTL to allow f_1 to be Min-max CTL as well. In that case, we require to appropriately define the evaluation syntax over the set of states where f_1 holds until f_2 holds. This may require different interpretations of the same Min-max CTL sub-formula, f_1, depending on whether the given formula involves $f_1 U f_2$ or $f_2 U f_1$. We have not attempted to define any such semantics for simplicity. However there can be interesting properties which can be expressed in the extended language.

6 Conclusion

Model checking with temporal logics such as TCTL, TLTL and RTCTL which explicitly reason about timing properties of transition systems is known to be more complex than reasoning about untimed temporal logics such as CTL. Specifically, CTL is attractive because it can be checked in time polynomial in the size of the transition system. In this paper we have shown that quantitative reasoning about timing properties can often be done in polynomial time, specifically when we are interested in the extremal timing properties of the system. We have shown that the proposed logic, Min-max CTL, can be used to express interesting queries about the best case and worst case temporal behaviors of timed models. We have also shown that for many useful cost functions, Min-max CTL can be evaluated in polynomial time.

Acknowledgments

The authors thank the reviewers for their constructive comments which helped improve the presentation of the paper. Dr. Pallab Dasgupta acknowledges the support of the Indian National Science Academy for partial support of this work. Dr. P.P. Chakrabarti acknowledges the Dept. of Science & Technology, Govt. of India for partial support of this work.

References

[1] Alur, R., and Henzinger, T., Real time logics: Complexity and Expressiveness. *Information and Computation*, **104**, 1, 35-77, 1993.

[2] Alur, R., Courcoubetis, C., and Dill, D., Model checking in dense real-time. *Information and Computation*, **104**, 1, 2-34, 1993.

[3] Alur, R., and Henzinger, T.A., A really temporal logic. *JACM*, **41**, 1, 181-204, 1994.

[4] Alur, R., Timed Automata. Manuscript: www.cis.upenn.edu/~alur/Nato97.ps.gz, 1998.

[5] Burch, J.R., Clarke, E.M., Long, D.E., McMillan, K.L., and Dill, D.L., Symbolic model checking for sequential circuit verification. *IEEE Trans. on Computer Aided Design*, **13**, 4, 401-424, 1994.

[6] Campos, S., and Clarke, E.M., Real-time symbolic model checking for discrete time models. In *Theories and experiences for real-time system development*, AMAST series in computing, 1994.

[7] Campos, S.V., Clarke, E.M., Marrero, W., and Minea, M., Verus: A tool for quantitative analysis of finite-state real-time systems. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*, 1995.

[8] Clarke, E.M., Emerson, E.A., and Sistla, A.P., Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Program. Lang. & Systems*, **8**, 2, 244-263, 1986.

[9] Clarke, E.M., and Kurshan, R.P., Computer aided verification. *IEEE Spectrum*, **33**, 6, 61-67, 1996.

[10] Cormen, T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms*. The MIT Press, Cambridge and McGraw-Hill, 1990.

[11] Emerson, E.A., Mok, A.K., Sistla, A.P. and Srinivasan, J., Quantitative temporal reasoning, In *First Annual Workshop on Computer-Aided Verification*, France, 1989.

[12] Kropf, T., and Ruf, J., Using MTBDDs for discrete timed symbolic model checking. In *Proc. of ED&TC*, 182-187, 1997.

[13] Papadimitriou, C.H., *Computational Complexity*, Addison-Wesley, 1994.