One-Clock Priced Timed Games are PSPACE-hard

John Fearnley
Dept. of Computer Science,
University of Liverpool
John.Fearnley@liverpool.ac.uk

Rasmus Ibsen-Jensen
Dept. of Computer Science,
University of Liverpool
R.Ibsen-Jensen@liverpool.ac.uk

Rahul Savani
Dept. of Computer Science,
University of Liverpool
Rahul.Savani@liverpool.ac.uk

Abstract

The main result of this paper is that computing the value of a one-clock priced timed game (OCPTG) is PSPACE-hard. Along the way, we provide a family of OCPTGs that have an exponential number of event points. Both results hold even in very restricted classes of games such as DAGs with treewidth three. Finally, we provide a number of positive results, including polynomial-time algorithms for even more restricted classes of OCPTGs such as trees.

***CCS Concepts:* • Theory of computation** → Problems, reductions and completeness; Exact and approximate computation of equilibria.

Keywords: Priced timed games, computational complexity, game theory

ACM Reference Format:
John Fearnley, Rasmus Ibsen-Jensen, and Rahul Savani. 2020. One-Clock Priced Timed Games are PSPACE-hard. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3373718.3394772

1 Introduction

In this paper, we study *priced timed games* (PTG), which are two-player zero-sum games that are played on a graph. The defining feature of PTGs is that the game is played over time, with players accumulating costs both for spending time waiting in states, and for using edges. Ultimately, one of the players would like to reach a goal state while minimizing the cost, while the opponent would like to prevent the goal state from being reached, or if that is impossible, to maximize the cost of reaching the goal state.

Priced timed games have been studied extensively in the literature, starting with the work of La Torre, Mukhopadhyay, and Murano [18] who first studied games on DAGs, with the later paper of Bouyer, Cassez, Fleury, and Larsen [8] being the first to study the concept on general graphs. Since then, there has been a great deal of follow-up work on these games, e.g., [1, 3, 5–14, 16–18, 20], including work on practical applications in, for example, embedded systems, and also applications in other theoretical results.

One-clock priced timed games. In general, a PTG can have any number of *clocks*, which all increase at the same rate as time progresses, but which can be independently *reset* back to zero. The edges of the game can have *guards*, which only allow the edge to be used if the clock values satisfy the conditions of the guard.

In this paper, we focus on the case in which there is exactly one clock, and so we study *one-clock priced timed games* (OCPTG). It has been shown that one-clock priced timed games always have a value [11], and moreover algorithms have been proposed [11, 16, 20] for computing the value of these games. The current state of the art is the algorithm of Hansen, Ibsen-Jensen, and Miltersen [16], who give an algorithm that runs in $O(m \cdot \text{poly}(n) \cdot 12^n)$ time, where m is the number of edges and n is the number of vertices. This gives an exponential-time upper bound for the problem.

It has remained open, however, whether the problem can be solved in polynomial time. The running time of Hansen, Ibsen-Jensen, and Miltersen's algorithm [16] is polynomial in the number of *event points* in the game, which are the set of points at which the gradient of the value function changes. They showed that all OCPTGs have at most $24m(n+1)12^n$ event points, which directly leads to the running time of their algorithm mentioned above. They conjectured that the number of event points in a OCPTG is actually bounded by a polynomial [16], and if this conjecture were true, then their algorithm would always terminate in polynomial time.

1.1 Our contribution

Lower bounds. This paper shows that computing the value of a one-clock priced timed game is very unlikely to be solvable in polynomial time, by showing that the problem is actually PSPACE-hard. We begin by constructing a family of examples that have exponentially many event points. This explicitly disproves the conjecture of Hansen, Ibsen-Jensen, and Miltersen. We then use those examples as the foundation of our computational complexity reductions. We first show that the problem is both NP and coNP-hard, and we then combine the techniques from both those reductions to show

LICS '20, July 8–11, 2020, Saarbrücken, Germany
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7104-9/20/07...$15.00
https://doi.org/10.1145/3373718.3394772

hardness for the k-th level of the polynomial-time hierarchy, for all k, and finally PSPACE.

Remark. We consider, as is standard in computer science, the case when the numbers of the input (in particular, the edge weights) are given in binary. If, instead, the edge weights are given in unary, our construction is not exponential. It is an open question if there is a construction that is exponential with numbers given in unary.

Lower bounds for special cases. All of our lower bound constructions produce graphs with special structures. In particular they are all acyclic, planar, have in-degree and out-degree at most 2, and overall degree at most 3 (our figures show a simpler variant with overall degree at most 4). Also, the treewidth, cliquewidth, and rankwidth of our constructions are all 3.

Our results for the polynomial-time hierarchy give additional properties. To obtain hardness for the k-th level of the polynomial time hierarchy, we only need $k + 2$ distinct *holding rates*, which are the costs that the players incur by waiting in a particular state.

Another interesting feature is that, in a variant of the construction, which loses planarity, all but $k+1$ of the states, can be made *urgent*. A state is urgent if the player is not allowed to wait at the state. Urgent states are relevant because the results of [11, 20] are based on the technique of converting more and more states into urgent states, since it is easy to solve a game in which all states are urgent.

In particular, our NP- and coNP-hardness constructions have 3 distinct holding rates, namely $0, 1/2, 1$, and there is a variant in which all but two states can be made urgent.

Finally, members of our initial family that has exponentially many event points have the additional properties of having pathwidth 3, using only holding rates $\{0, 1\}$, and having only a single state that cannot be made urgent. Thus, the games may still have exponentially-many event points even for many of the most obvious special cases.

Upper bounds. Our hardness results essentially rule out finding polynomial-time algorithms for many questions in a large number of special cases, unless P = PSPACE. We are able to prove some upper bounds: we show that undirected graphs and trees have a polynomial number of event points, and so can be solved in polynomial time.

Finally, we show that OCPTGs on DAGs are in PSPACE by showing that a variant of the event-point iteration algorithm [16] can solve games on DAGS in polynomial space. Combined with our hardness results, we obtain a PSPACE-completeness result for OCPTGs played on DAGs. This result improves on an exponential-time algorithm by [1] that in turn improved on a double-exponential-time algorithm [18], both of which are designed for games with many clocks.

1.2 Related work

As shown in [7], building on a similar result in [12] for five clocks, some problems are undecidable in general for priced timed games with three clocks. This was extended to the value problem in [10]. The complexity of most problems for two clocks is still open.

Games with only a single player, called priced timed automata, have been studied extensively on their own, following their introduction in [2, 4]. They can be solved in NL for the one-clock case [19] and in PSPACE for the multiple clock case [6]. Games on DAGs are in EXP for any number of clocks [1], which improved on a previous 2EXP bound in [18]. Games with no costs and holding rates in $\{0, 1\}$ are called reachability timed games. They can be solved in polynomial time for one clock [16, 21] and in polynomial space for multiple clocks [17].

This result has been generalised by [14] to show a polynomial time algorithm for the decision question[1] for one-clock priced timed games with rates in $\{0, 1\}$ and integer costs. Previously, they also claimed that such games would have only a polynomial number of event points, implying that one could find the full value functions in polynomial time. This, however is incorrect: We show in [15] how to convert our examples with exponentially-many event points and two holding rates to have integer costs. Their result [14] does show a pseudo-polynomial number of event points for such games though.

More generally, [14] also give a pseudo-polynomial time algorithm for the special case with holding rates in $\{-d, 0, d\}$ for any number d (note that our paper otherwise does not discuss negative rates or costs).

2 Definitions

As shown by [16], every one-clock priced timed game can be reduced, in polynomial time, to a *simple* priced timed game (SPTG), which is an OCPTG in which there are no edge guards and no clock resets. Our hardness results will directly build SPTGs, and so we restrict our definitions to SPTGs in this section. Since every SPTG is a OCPTG, all of our hardness results directly apply to OCPTGs.

SPTGs. A simple priced timed game is a game played between two players called the minimizer and the maximizer. The game is formally defined by a 6-tuple: (V_1, V_2, G, E, c, r), where

- V_1 is the set of states belonging to the minimizer, V_2 is the set of states belonging to the maximizer, and G is a set of goal states. The set of all states is denoted as $V = V_1 \cup V_2 \cup G$, and we use n to denote the number of states.
- E is a set of directed *edges*, which is a subset of $V \times V$. We use m to denote the number of edges.

[1]I.e. given a game, a state, and a number, is the value of starting in that state at time 0 above the number?

- $c : E \rightarrow \mathbb{R}_{\geq 0}$ is a non-negative *cost function* for edges.
- $r : V \rightarrow \mathbb{R}_{\geq 0}$ is a non-negative *holding rate function* for states.

The game takes place over a period of time. At the start of the game, a pebble is placed on one of the states of the game. In each round of the game, we will be at some time $t \in [0, 1]$. The player who owns the state that holds the pebble, can choose to move the pebble along one of the outgoing edges of that state, or to delay until some future point in time. Moving along an edge e incurs the fixed one-time cost given by $c(e)$, while delaying for d time units at a state s incurs a cost of $r(s) \cdot d$.

The game starts at time 0, and either ends when a goal state is reached, or it never ends. If a goal state is not reached, then the minimizer loses the game, and receives payoff $-\infty$. Otherwise, the payoff is the total amount of cost that was incurred before the goal state was reached, which the maximizer wins, and the minimizer loses.

Strategies. Our players will use *time-positional strategies*, meaning that for each state and each point in time, the strategy chooses a fixed action that is executed irrespective of the history of the play. Formally, for each $j \in \{1, 2\}$, a time-positional strategy σ_j for player j is defined by a pair (W^j, S^j).

- W^j is a set of non-negative *change points*. That is, $W^j = \{0 = w^j_0 < w^j_1 < w^j_2 < \cdots < w^j_{k-1} < 1 = w^j_k\}$ gives a sequence of points in time at which the player changes their strategy. For notational convenience we define $w^j_{k+1} = \infty$.
- $S^j = \{S^j_0, S^j_1, \ldots S^j_k\}$ is a corresponding list of *strategy choices*, which defines what action the player chooses at each point in time. The player can either choose an outgoing edge, or choose to wait at the state, which we denote with the symbol δ. So, for each i, we have that $S^j_i : V_j \rightarrow E \cup \{\delta\}$ with the requirement that if $S^j_i(s) \in E$ then $S^j_i(s) = (s, s')$ for some state s'. At time 1, delay is not possible, so for all $s \in V_j$ we require that $S^j_k(s) \neq \delta$.

Plays. Given a pair of strategies σ_1, σ_2 for the minimizer and maximizer, respectively, the resulting *play* from a starting state s_0, and a starting time t_0 is denoted as $P(\sigma_1, \sigma_2, s_0, t_0)$, and is defined as follows. Initially, place a pebble on s_0 at time t_0. For each $j \in \{1, 2\}$ and i, whenever the pebble is placed on a state s_i in V_j at time t_i, let i' be the index such that $t_i \in [w^j_{i'}, w^j_{i'+1})$ and let $\ell \geq i'$ be the smallest index such that $e_i := S^j_\ell(s_i) = (s_i, s_{i+1}) \neq \delta$. Then, player j waits until time $t_{i+1} = \max(w^j_\ell, t_i)$, and then moves the pebble on to s_{i+1} at time t_{i+1}. We also define $\delta_i := t_{i+1} - t_i$ to be the delay that player i chooses at time t_i in s_i.

If $s_i \in G$, then the play is over and $|P(\sigma_1, \sigma_2, s_0, t_0)| = i$. If the play is never over, i.e. for all i, $s_i \notin G$, we have that $|P(\sigma_1, \sigma_2, s_0, t_0)| = \infty$.

Outcomes and values. The *outcome* $\text{val}(P)$ is defined to be ∞ if $|P| = \infty$, since no goal state is reached. Otherwise, the outcome is

$$\text{val}(P) := \sum_{t=0}^{|P|-1} (r(s_t) \cdot \delta_t + c(e_t)),$$

where $r(s_t) \cdot \delta_t$ is the cost for holding at the state s_t for δ_t time units, and $c(e_t)$ is the cost for using the edge e_t. Fix s to be a state, and t to be a time. The *lower value* is defined to be $\underline{\text{val}}(s, t) = \sup_{\sigma_2} \inf_{\sigma_1} \text{val}\left(P(\sigma_1, \sigma_2, s, t)\right)$, while the *upper value* is defined to be $\overline{\text{val}}(s, t) = \inf_{\sigma_1} \sup_{\sigma_2} \text{val}\left(P(\sigma_1, \sigma_2, s, t)\right)$.

By definition, $\underline{\text{val}}(s, t) \leq \overline{\text{val}}(s, t)$. It is implied by the proofs of [11], for a richer class of strategies, that $\underline{\text{val}}(s, t) = \overline{\text{val}}(s, t)$. It mostly follows from [11] (but formally, one also needs [16]) that this equality holds even when the minimizer is restricted to time-positional strategies in the definition of lower value and the maximizer is restricted to time-positional strategies in the definition of upper value. Therefore, the game is determined in time-positional strategies, and we use $\text{val}(s, t) := \underline{\text{val}}(s, t) = \overline{\text{val}}(s, t)$ to denote the value of the game starting at the state s, and time t.

Optimal and ϵ-optimal strategies. Given an $\epsilon \geq 0$, a strategy σ_1 is *ϵ-optimal* for the minimizer if $\text{val}(s, t) - \epsilon \leq \sup_{\sigma_2} \text{val}(P(\sigma_1, \sigma_2, s, t))$ for all s and t. A strategy is *optimal* if it is 0-optimal. The definitions for the maximizer are symmetric. As shown in [11], for all $\epsilon > 0$, ϵ-optimal strategies exist in OCPTGs, and [16] have shown that optimal strategies exist in SPTGs. Moreover, the function $\text{val}(s, t)$ is piecewise linear for OCPTGs [11], and additionally also continuous for SPTGs [16].

Event points. As mentioned, the value function of each state in an SPTG is piecewise linear. An *event point* is a point in time at which the value function of some state s changes from one linear function to another. The set of *event points* contains every event point for every state in the game.

As shown in [16], improving on [11, 20], the number of event points is less than 12^n for SPTGs and it is less than $m \cdot 12^n \cdot \text{poly}(n)$ for OCPTGs. The optimal strategy profiles for SPTGs constructed by [16] have the set of change points being equal to the set of event points. Conversely, it is clear that event points are a subset of the change points in any optimal strategy profile.

3 Exponentially many event points are required

We begin by constructing a family of simple priced timed games in which the number of event points in the optimal strategy is exponential. This serves two purposes. Firstly, it provides a negative answer to the question, posed in prior work [16], of whether the number of event points is polynomial. Secondly, this construction will be used in a fundamental way in the hardness results that we present in later sections.

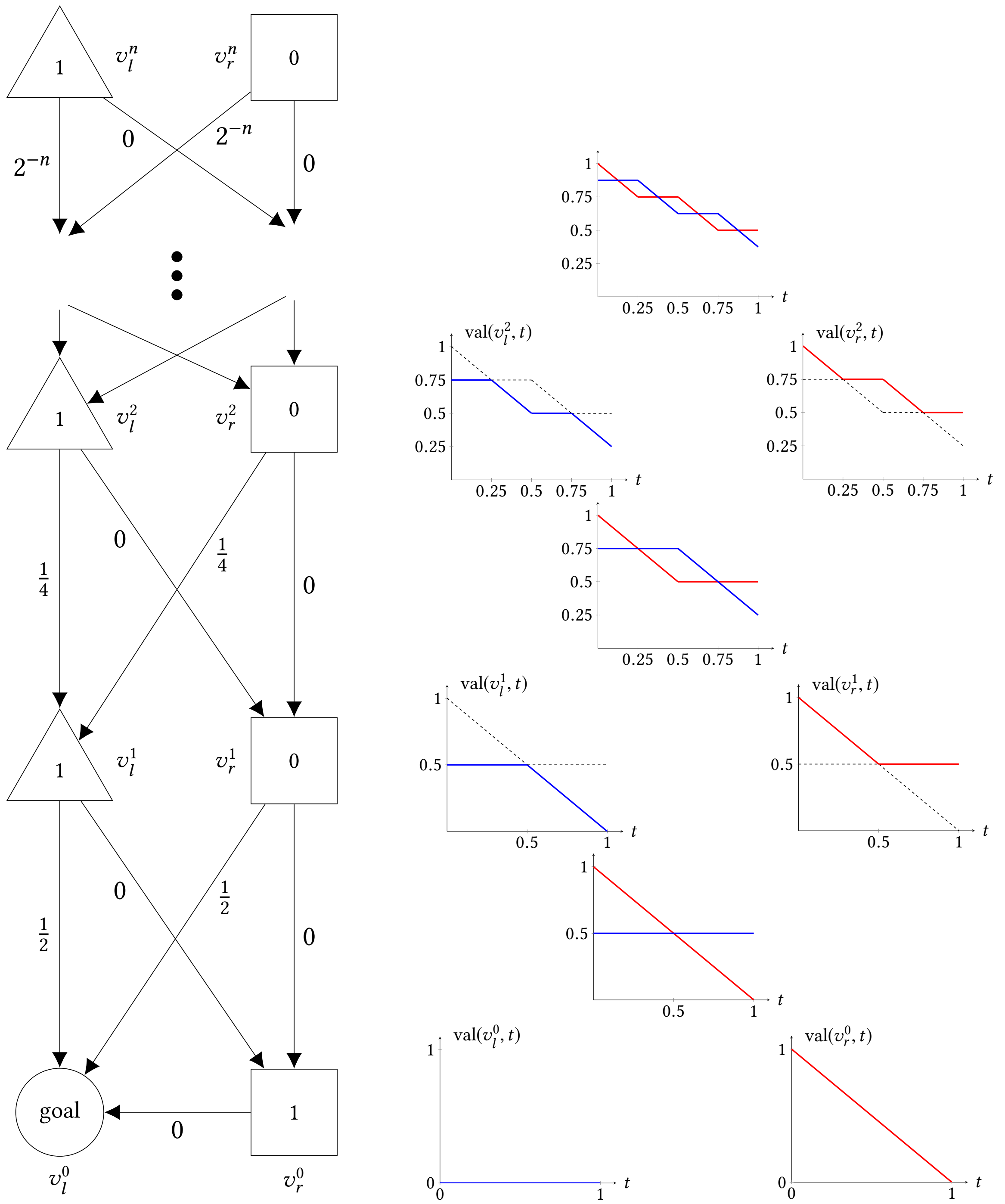


Figure 1. Event points lower bound construction. Values for left (minimizer) triangle states are formed as lower envelopes, and for right (maximizer) square states as upper envelopes, of intermediate "shift blue and overlay with red" diagrams.

The construction. The family of games is shown on the left-hand side in Figure 1. States belonging to the maximizer are drawn as squares, while states belonging to the minimizer are drawn as triangles. The number displayed on each state is the holding rate for that state, while the number affixed to each edge is the cost of using that edge.

The game is divided into levels, with each level containing two states, which we will call the *left state* (denoted as v_ℓ^i) and the *right state* (denoted as v_r^i). These names correspond to the positions at which these states are drawn in Figure 1 and are annotated on the figure.

At the bottom of the game, on level 0, the left state v_ℓ^0 is the goal state and the right state v_r^0 is a maximizer state with holding rate 1. The state v_r^0 has an edge to v_ℓ^0 with cost 0. For each level $i > 0$, the left state v_ℓ^i is a minimizer state with holding rate 1, and the right state v_r^i is a maximizer state with holding rate 0. Both states have the same outgoing edges: an edge to v_ℓ^{i-1} with cost 2^{-i}, and an edge to v_ℓ^{i-1} with cost 0.

Value diagrams. On the right-hand side in Figure 1, we show the value function for each state, represented as *value diagrams*. These show the value for each state at each point in time. The bottom-left diagram shows the value function of the goal state, which is zero at all points in time, since the game ends when the state is reached. The bottom-right diagram shows the value function of the state v_r^0 (the bottom-right state of the game). At this state, the maximizer will wait for as long as possible before moving to the goal, since this maximizes the cost generated from the holding rate of 1. Hence, the value of this state is $1 - x$ at time $0 \leq x \leq 1$, which is shown in the diagram.

For the states at level one of the game, first observe that there is no incentive for either player to wait. The left state has holding rate 1, which is the worst possible holding rate for the minimizer, and the right state has holding rate 0, which is the worst possible holding rate for the maximizer. Hence both players will move immediately to the lower level, and we must determine which state is chosen.

To do this, we use the value function diagrams of the lower level. Both players can move to the goal with an edge cost of 0.5, or move to v_r^0 with a cost of zero. So we shift the value function of the goal state up by 0.5, and then overlay it with the value function of v_r^0. This is displayed in the value diagram that lies between the two layers. The minimizer's value function is the *lower envelope* of these two functions, which minimizes the value, while the maximizer's value function is the *upper envelope*, which maximizes the value. This is shown in the value diagrams of the two states at level one.

This process repeats for each level. E.g. for level two, we overlay the two value diagrams from level one, after shifting the left-hand diagram up by the edge cost of 0.25, and then we take lower and upper envelopes for the respective players.

The exponential lower bound. To see that this game produces exponentially many event points, observe that the left-hand value diagram at level two contains two complete copies of the left-hand value diagram at level one, and that the same property holds for the right-hand value diagrams. This property generalizes, and we can show that the value diagrams for v_ℓ^n and v_r^n both contain 2^n distinct line segments. The following theorem is shown in [15].

Theorem 1. *There is a family of simple priced time games that have exponentially many event points.*

3.1 Inapproximability with few change points

We are also able to show that both players must use strategies with exponentially many change points in order to play close to optimally in our lower bound game. Specifically, we can show that if the game starts at the kth level of our game, that is, in the vertices v_ℓ^k or v_r^k, and if both players play ϵ-optimally for $\epsilon < 1/2^k$, then every interval of the form

$$\left[\frac{c}{2^{k-1}}, \frac{c+1}{2^{k-1}}\right)$$

for some integer c, must contain a change point. This is only possible if there are 2^{k-1} distinct change points.

We shall illustrate this for the case where $k = 3$, by showing that the minimizer must use four different change points at v_ℓ^3 to play an ϵ-optimal strategy with $\epsilon < 1/8$. The value diagram of v_ℓ^3 is the lower envelope of the value diagram at the top of Figure 1. Let us consider the interval $D = [c/4, (c+1)/4)$ for some integer c, and for the sake of contradiction, suppose that there are no change points in this interval.

Since the minimizer cannot change their strategy, they have only three options during D: always go to v_ℓ^2, always go to v_r^2, or wait at v_ℓ^3 until the end of D.

If the minimizer chooses to wait, then let us consider a play starting at time $c/4$. This play has a payoff of at least

$$1/4 + \text{val}(v_\ell^3, (c+1)/4),$$

because we wait with a holding rate of 1 for 1/4 time units, and then the best we can do at time $(c+1)/4$ is to follow the optimal strategy, which gives us a payoff of $\text{val}(v_\ell^3, (c+1)/4)$. On the other hand, $\text{val}(v_\ell^3, c/4)$ is such that

$$\text{val}(v_\ell^3, c/4) = \text{val}(v_\ell^3, (c+1)/4) + 1/8.$$

This can be seen from the value function for v_ℓ^3 in Figure 1: the first half of the value function during D is flat, while the second half falls at rate 1, hence the difference is 1/8. Since choosing to wait achieves a value that is 1/8 bigger than this, waiting cannot be ϵ-optimal for any $\epsilon < 1/8$.

For the other two options, the outcomes can be seen in the top value diagram in Figure 1. The red line gives the outcome for starting in v_ℓ^3 and always going to v_r^2, while the blue line gives the outcome for always going to v_ℓ^2, assuming that both players play optimally afterwards. The optimal strategy takes the lower envelope of the two lines.

There is a difference of 1/8 between the two lines at $c/4$ and $(x+1)/4$, but the lines cross in the middle of the interval, so the line that is part of the lower envelope at $c/4$ is not the line that is part of the lower envelope at $(c+1)/4$. Hence, choosing to go to v_r^2, or to v_ℓ^2 for the entire interval will cause a loss in value of up to 1/8, relative to the optimal strategy, which is the difference in height between the lines. The strategy is therefore not ϵ-optimal since $\epsilon < 1/8$.

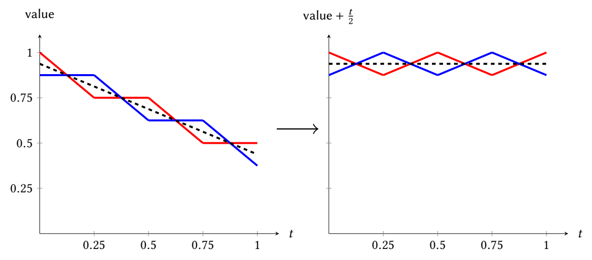


Figure 2. Our relative value diagramming convention.

This argument is generalized to all $k \geq 1$ and both players in [15]. Ultimately, the result is stated in the following lemma.

Lemma 1. *There is a family of simple priced time games in which every ϵ-optimal strategy with $\epsilon < 1/2^k$ uses 2^{k-1} change points.*

4 NP and coNP lower bounds

We now present NP-hardness and coNP-hardness results for computing the value of a simple priced timed game. This serves two purposes. Firstly, it introduces some of the key concepts that we will use in our PSPACE-hardness result. Secondly, these hardness results will hold for SPTGs that have only the holding rates $\{0, 1/2, 1\}$, which is not the case for our later results.

Our goal in this section is to show hardness results for the following decision problem: given a state s and a constant c, decide whether $\text{val}(s, 0) \geq c$. In other words, it is hard to determine the value of a particular state at time zero. The majority of this section will be used to describe the NP-hardness result, and the coNP-hardness will be derived by slightly altering the techniques that we develop.

Relative values. The family of games from Section 3 will be used as a basis for this result. We start by discussing a change in perspective that is helpful when dealing with value diagrams. Take, for example, the value diagram at the top of Figure 1. Observe that both of the value functions depicted in this diagram are weakly monotone. This will always be the case in an SPTG, since there are no guards, meaning that costs can only increase as the amount of time left in the game increases.

We will use values at specific points in time to encode information. But we will not use the absolute value, but rather the value *relative* to some monotone linear function. This is shown in Figure 2. On the right-hand side we have added the linear function time/2, which causes the value functions to become horizontal. The diagram shows the value functions increasing and decreasing relative to this linear function.

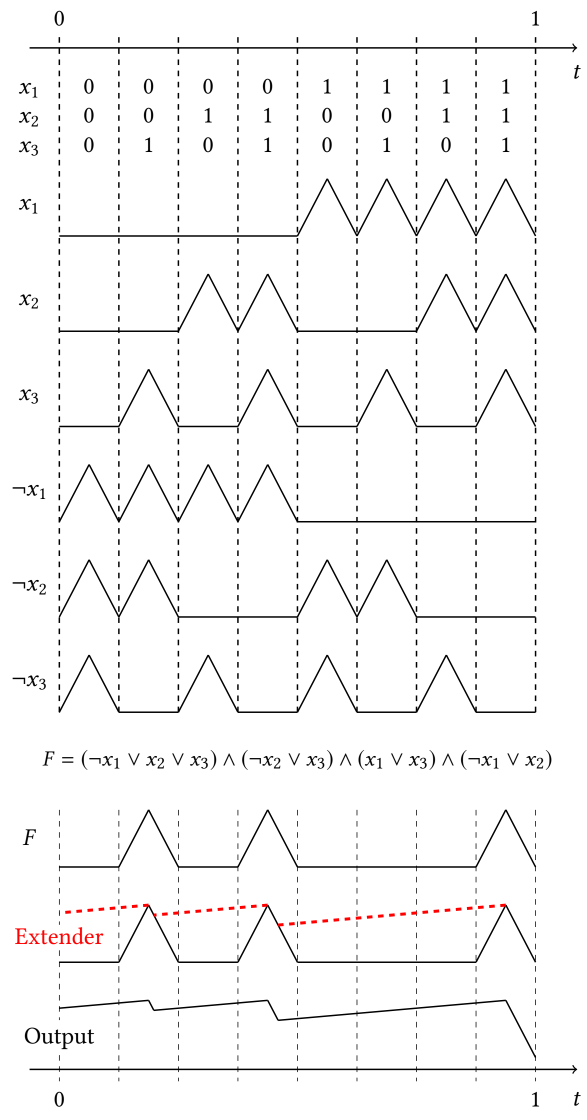


Figure 3. NP lower bound construction.

We will use relative values in our reduction, because it makes it easier to understand. It is worth pointing out, however, that this is only a change in perspective. The underlying absolute values are still always weakly monotone.

Enumerating bit strings. Our NP-hardness reduction will be a direct reduction from Boolean satisfiability. There are two steps to the reduction. First we build a set of gadgets that enumerate all possible n-bit strings over time, and then we build a gadget that tests whether a Boolean formula is true over this set of bit strings.

We start by describing the enumeration gadget. We denote the n bits of a bit string as x_1 through x_n. The enumeration gadget builds $2n$ states, corresponding to x_i and $\neg x_i$ for each index i. The top half of Figure 3 shows the relative value diagrams for these states.

The gadget divides time into 2^n intervals, with each interval corresponding to a particular bit string. Bit values of the bit-string are encoded using the relative value function of the states, using two fixed constants L and H that the relative value stays between.

- If a bit is zero for an interval, then the relative value of the state remains at L during the interval.
- If a bit is one for an interval, then the relative value of the state begins the interval at L, it increases during the interval to H, and then decreases back to L by the end of the interval. This forms the peaks shown in Figure 3.

The enumeration gadget produces these value functions by using several copies of the exponentially-many event point games from Section 3. From Figure 1, we can see that the value functions there are similar to what we want: the functions alternate between having high relative value and low relative value, and there are exactly 2^i alternations at level i. However, these value functions do not exactly match those shown in Figure 3. Specifically:

- The exponential lower bound functions are symmetric with respect to peaks and troughs, but we would like zeroes to be represented by the fixed constant L, and ones to be represented as peaks.
- The functions start at either peaks or troughs, but we would like to start in the middle of the waveform. So attempting to represent x_1 in Figure 3 using the value functions from the exponential lower bound would result in a bit-sequence like `1 1 0 0 0 0 1 1`, rather than `0 0 0 0 1 1 1 1`.
- When a state has a sequence of intervals that all encode one-bits, we would like each to contain a copy of the peaks shown in Figure 3. However, the exponentially-many event point game value functions would instead give us a single large peak during the whole interval.

To address these issues, we transform the exponentially-many event point game value functions so that they have these properties. This involves inserting a sequence of intermediate states, and the construction is described in detail in [15].

Evaluating a Boolean formula. Once we have constructed states that correspond to x_1 through x_n and $\neg x_1$ through $\neg x_n$, we can then design a gadget to evaluate an arbitrary Boolean formula F over every n-bit string. The output of this gadget is a state, that we will also call F, whose value is depicted in Figure 3. The output of the F state uses the same encoding as before: if F evaluates to false for a specific bit string, then the value of F remains at L for the entire interval, while if it evaluates to true, the value forms a peak that starts at L, increases to touch H, and then returns to L by the end of the interval.

To evaluate the formula, we first apply De Morgan's laws to ensure that all negations are applied to propositions, meaning that all internal operations of the formula consist only of $\wedge$ and $\vee$ operations. Next, we introduce a state in the game for each sub-formula $F' = x \oplus y$ of F, where $\oplus \in \{\wedge, \vee\}$. This state will have edges to the states corresponding to x and y with no edge costs, and

- if $\oplus = \vee$ then the state is a *maximizer* state with *holding rate* 0, while
- if $\oplus = \wedge$ then the state is a *minimizer* state with *holding rate* 1.

As in the exponentially-many event point games, the holding rates have been chosen so that neither player has an incentive to wait at these states. So the relative value of the state F'

- will be the maximum of the two input states for an $\vee$ gate, meaning that in any particular interval the relative value of F' will contain a peak if either of the two input states contains a peak,
- while for a $\wedge$ gate, the relative value will be the minimum of the two inputs, meaning that an interval will contain a peak only when both inputs contain peaks[2].

Hence this correctly simulates boolean logic, and the output of state F will encode the set of bit strings that satisfy the formula.

NP-hardness of computing values. Finally, we can turn this into our NP-hardness result. So far, we have shown how to evaluate the Boolean formula, but the outcome of the evaluation does not affect the values at time zero, because each evaluation is entirely contained within its interval.

To address this, we introduce one final state called the *extender*. The relative value function of the extender is shown at the bottom of Figure 3. Whenever the relative value of F peaks at the value H, the extender makes the relative value rise more gradually before the peak. This rate is carefully chosen so that the value will not have returned to L even after all 2^n intervals. Hence,

- if the relative value of F touches H at any point in time, the relative value of the extender at time zero will be strictly greater than L, while
- if the relative value of F is never more than L, then the relative value of the extender will be L at time zero.

This implies that the relative value (and hence absolute value) of the extender at time zero depends on the satisfiability of the formula F, which gives us our NP-hardness result.

The extender state is a maximizer state that has one outgoing edge to the state F with no edge cost, and a carefully chosen holding rate. The second to last relative value diagram in Figure 3 shows the effect of the holding rate of the extender. The idea is that the maximizer would like to wait

[2]An issue could arise if the peaks were located at different points in the intervals, but, as shown in [15], the peaks are always exactly in the middle.

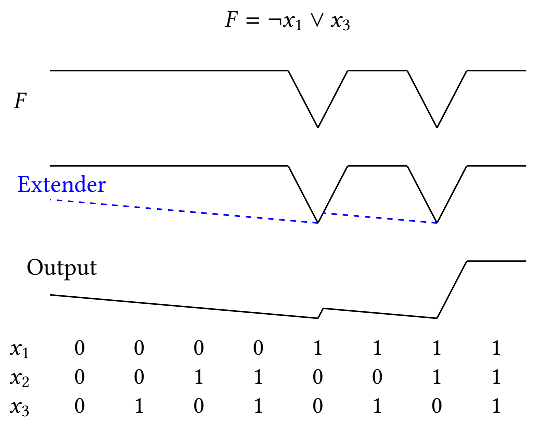


Figure 4. coNP lower bound construction. Troughs encode false assignments.

in the extender until the next interval in which the formula evaluates to true (if there is such an interval).

The holding rate at the extender determines the gradient of the red lines. For NP-hardness it is sufficient for this line to be horizontal[3], and never touch the relative value of L, but the ability for the extender state to decay back to L after a finite amount of time will later be used to show our PSPACE-hardness result.

One final thing to note is that this construction uses exactly three different holding rates. The exponentially-many event point games use the holding rates 0 and 1, and one extra holding rate (of 1/2) is introduced in the enumeration gadget. We get the following theorem. The full formal description of the construction, along with a proof of correctness, can be found in [15].

Theorem 2. *For an SPTG, deciding whether $v(s, 0) \geq c$ for a given state s and constant c is* NP*-hard, even if the game has only holding rates in* $\{0, 1/2, 1\}$.

coNP-hardness of computing values. To obtain coNP hardness, we use essentially the same technique, but with one important difference in our encoding of bits. In the NP-hardness result we used the constant L to encode a zero bit, and a peak that touches the constant H to encode a one bit. To prove coNP hardness, we flip that upside down.

- If a bit is one during an interval, then the relative value of the state will remain at H for the entire interval.
- If a bit is zero during an interval, then this is encoded as a trough, during which the relative value touches L.

We use this encoding, which we call the *reverse encoding* throughout the coNP-hardness construction: all of the states of the enumeration gadget use the reverse encoding, and the formula evaluation is also done in reverse encoding. We end up with a state whose relative value encodes F in reverse encoding, as shown in Figure 4.

With the reverse encoding, if F is always true, then the relative value of the state will be H. If there exists an input that makes F false, then this will be encoded as a trough. We can extend this back to time zero using an extender state with a carefully chosen holding rate[4], though this time the extender state must be a minimizer state, since we want the extender player to obtain a lower value by waiting until F is not satisfied.

The end result is that the relative value at time 0 is H if F is always true, and it is strictly less than H if there exists an assignment to variables that makes F false. Again, this construction uses only three holding rates, so we obtain the following theorem.

Theorem 3. *For an SPTG, deciding whether $v(s, 0) \geq c$ for a given state s and constant c is* coNP*-hard, even if the game has only holding rates in* $\{0, 1/2, 1\}$.

The proof of this theorem appears in [15]. Since the NP and coNP-hardness proofs are very similar, we prove them both at the same time in the appendix.

5 PSPACE lower bound

We now move on to our main result, and show that computing the value of a particular state at time zero is PSPACE-hard. We will reduce directly from TQBF, which is the problem of deciding whether a quantified Boolean formula is true. The high level idea is to make use of the techniques from our NP-hardness reduction to deal with existential quantifiers, and the techniques from our coNP-hardness reduction to deal with universal quantifiers.

As a running example, we will use the formula

$$F = (x_1 \land x_4 \land x_5) \lor (x_1 \land \neg x_4 \land \neg x_5),$$

and we will apply the reduction to the TQBF instance

$$\forall x_1 \; \exists x_4 x_5 \cdot F(x_1, x_4, x_5).$$

The slightly odd choice of variable indices will be explained shortly.

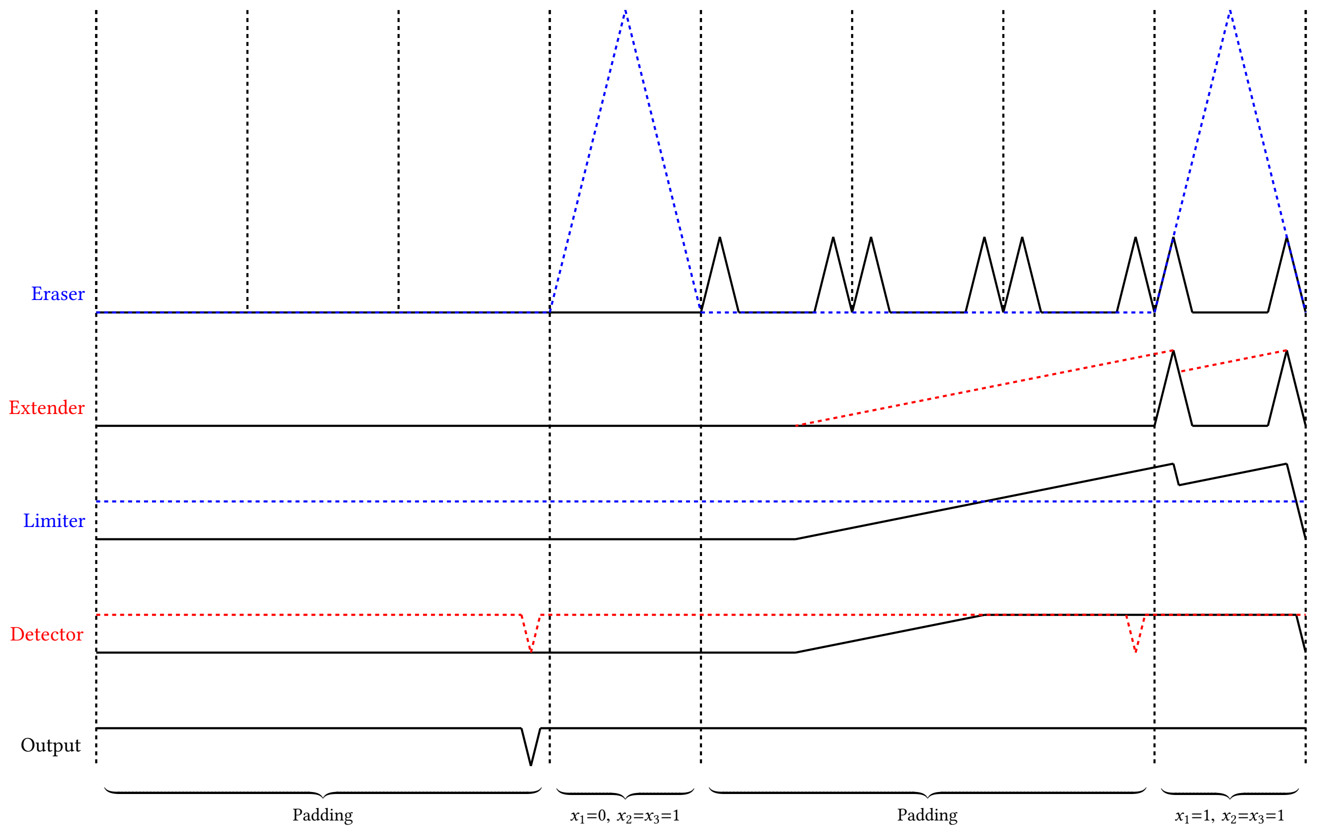
Overview. As in previous reductions, we will divide the time period into intervals, and we will associate each interval with a bit string, and evaluate the formula on each of those bit strings. However, in this setting we must now deal with both types of quantifiers.

Our solution is shown in Figure 5. We use the quantifiers to divide the bit strings into blocks, and place padding between the blocks. For our running example, we have two blocks, which correspond to the case where $x_1 = 0$ and the case where $x_1 = 1$. So we have split the bit strings according to the universal quantifier in the formula. We will refer to the two sub-instances as $F'(x_1) := \exists x_4 x_5 \cdot F(x_1, x_4, x_5)$.

[3]Horizontal in our relative value diagrams means a holding rate of 1/2 in the actual game with absolute values.

[4]As for NP-hardness, this holding rate can be 1/2.

$$F(x_1, x_4, x_5) \;:=\; \forall x_1 \; \exists x_4 x_5 \cdot (x_1 \wedge x_4 \wedge x_5) \vee (x_1 \wedge \neg x_4 \wedge \neg x_5)$$



x_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
x_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 5. PSPACE lower bound construction.

The idea is to evaluate the two blocks separately using the method from the NP-hardness reduction. So we will determine whether $F'(x_1)$ holds when $x_1 = 0$, and independently determine whether it holds when $x_1 = 1$. This then leaves us with the problem of deciding whether $\forall x_1 \cdot F'(x_1)$ is true, which will be evaluated using methods from the coNP-hardness reduction. We do this by turning the output of the two independent evaluations of F' into a reverse encoded input for the coNP problem.

The padding. The padding between the blocks is used to ensure that the two evaluations of F' are independent. The padding is implemented by inserting extra dummy variables into the formula. In our running example, we add the extra dummy variables x_2 and x_3, but we do not modify the formula itself in any way. As shown in the first line of Figure 5, this leads to each block being repeated four times, since we enumerate all four possible settings for x_2 and x_3, but none of these change the output of the formula.

The first step is to take the minimum of this relative value function with a state that we call the *eraser*, whose relative value function is shown in blue in the first line of Figure 5. This value function peaks during the block[5] that we would like to keep, but stays at the value L during the blocks that we would like to erase. So by taking the minimum, we keep the right-most copy, and erase the other three, which gives us the padding between the blocks.

Recall from the NP-hardness reduction that the extender state is used to detect whether the relative value has peaked

[5]We do so by encoding the formula $(x_2 \wedge x_3)$ using our previous constructions for formulas but only over these two variables, which results in the very high peak.

during a block. Furthermore, the relative value of the extender decays over time, and that the rate at which this happens is controlled by the holding rate of the extender.

In the PSPACE-hardness reduction, we choose the decay rate so that the value will always decay back to L during the padding before the next block starts. This can be seen in the extender state in the second line of Figure 5. In the right-hand block, there are two assignments that make the formula true, and this information is carried to the left-hand edge of the block by the extender. The padding provides enough space for the extender to decrease back to L before the left-hand block begins.

Changing the encoding. So far we have independently evaluated $F'(x_1)$ for both possible settings of x_1, and this is encoded in the second value function of Figure 5. The rest of the steps in that figure show how we then turn this into a reverse encoding of $\forall x_1 \cdot F'(x_1)$.

The overall goal is to detect whether the extender is above L at the left-hand boundary of each block. In fact, we choose the decay rate of the extender to be slow enough to guarantee that if there was a peak during the block the value of the extender is above $(H+L)/2$ at the left-hand edge of the block.

The first step is to take the minimum of the relative value function with a *limiter* state, shown in blue in the third line of Figure 5, whose relative value is constant at $(H+L)/2$. This effectively chops off the top half of the function. We then construct a state, known as the *detector*, shown in red in the fourth line of Figure 5. This state has a relative value function that remains at $(H+L)/2$ throughout, except at the left-hand edge of each block, where there is a trough that touches L. We do this by encoding the formula $(\neg x_2 \vee x_3 \vee \neg x_4 \vee \neg x_5)$ in reverse encoding.

We take the maximum of the value function with the detector. This does the following.

- If there was a peak during the block, the value of the extender will be above $(H+L)/2$, and so the trough in the detector will be eliminated. The limiter ensures that the relative value does not exceed $(H+L)/2$ in this case.
- If there was no peak during the block, the value of the extender will be L, and so the trough in the detector will not be eliminated.

The end result is that we have a trough in the final value function if and only if $F'(x_1)$ was false for the corresponding block.

Observe that this is a valid reverse encoding of the problem $\forall x_1 \cdot F'(x_1)$, with the only change being that the relative function ranges between L and $(H+L)/2$ rather than L and H. So we can apply the techniques from the coNP-hardness reduction to determine whether $\forall x_1 \cdot F'(x_1)$ is true.

PSPACE-hardness. So far, we have seen how to deal with alternations of the form $\forall x \exists y$, but the same techniques can also deal with alternations of the form $\exists y \forall x$. The only difference is that we must turn a reverse encoded output into the normal encoding, which can again be done with appropriately constructed limiter and detector states.

The full PSPACE-hardness result applies the two techniques inductively. Every alternation of quantifiers in the formula is handled by turning one encoding into the other, ready to be evaluated by the next level of quantifiers. The full details can be found in [15], where we prove the following result.

Theorem 4. *For an SPTG, deciding whether $v(s,0) \geq c$ for a given state s and constant c is PSPACE-hard.*

It is also worth noting that if the formula only has k alternations, then the resulting game uses $k+2$ distinct holding rates. The holding rates 0 and 1 are already used by the exponential lower bound game. Each level of alternation uses an extender state with a distinct holding rate, which accounts for the other k holding rates. Hence, we also get the following result.

Theorem 5. *For an SPTG with $k+2$ distinct holding rates, deciding whether $v(s,0) \geq c$ for a given state s and constant c is hard for the k-th level of the polynomial-time hierarchy.*

5.1 Other decision problems

All of our results so far have shown hardness of deciding whether $v(s,0) \geq c$ for some state s, and some constant c. In this section, we point out that our construction can also prove hardness for other, related, decision problems.

As in the NP- and coNP-hardness section, we can let the outer-most extender state produce a horizontal line, rather than a decaying one. This ensures that we can pick two constants H' and L' such that $\text{val}(v,0) = H'$ if the formula is true and $\text{val}(v,0) = L'$ if the formula is false. Thus, all our hardness proofs for each of NP-, coNP- and PSPACE-hardness, and hardness for the k-th level of the polynomial time hierarchy, apply to the following promise problem.

PromiseSPTG:. Given an SPTG, a state v and two numbers $c > c'$, with the promise that $\text{val}(v,0) \in \{c, c'\}$, is $\text{val}(v,0) = c$?

This problem can be reduced in polynomial time to each of the following problems.

1. DecisionSPTG: Given an SPTG G, a state v and a value c, is $\text{val}(v,0) \geq c$?
2. ApproxAdditiveSPTG: Given an SPTG G, a state v and an $\epsilon > 0$, find a number c such that $\text{val}(v,0) \in [c-\epsilon, c+\epsilon]$?
3. ApproxMultiplicativeSPTG: Given an SPTG G, a state v and an $\epsilon > 0$, find a number c such that $\text{val}(v,0) \in [c(1-\epsilon), c(1+\epsilon)]$?
4. EqualDecisionSPTG: Given an SPTG G, a state v and a value c, is $\text{val}(v,0) = c$?
5. ϵ-StrategySPTG: Given an SPTG G, a state v, an $\epsilon > 0$ and an action a is there an ϵ-optimal strategy that uses a at time 0?

6. StrategySPTG: Given an SPTG G, a state v and an action a is there an optimal strategy that uses a at time 0?
7. AllOptimalStrategiesSPTG: Given an SPTG G, a state v and an action a do all optimal strategies use a at time 0?

The reduction is trivial for (1)-(4), since we have just removed the promise (and pick $0 < \epsilon < \frac{c-c'}{2}$, e.g. $\epsilon = \frac{c-c'}{3}$).

For (5), (6) and (7), fix some $\epsilon < \frac{c-c'}{2}$. We add another minimizer state v' to the game with holding rate $M + 1$, where M is the largest holding rate in the rest of the game. The state v' has an edge to v and an edge to a goal state. The edge to v has cost 0 and the edge to the goal state has cost $\frac{c+c'}{2}$.

It is clear that $\text{val}(v', 0) = c'$ if and only if $\text{val}(v, 0) = c'$. Also, if $\text{val}(v, 0) = c'$, then no ϵ-optimal strategy can use the edge to v at time 0, so no optimal strategy can do this either. Similarly, if $\text{val}(v, 0) = c$, then no ϵ-optimal strategy can use the edge to the goal state. This proves hardness for (5) and (6). Also, since the holding rate is larger than M in the above construction, the minimizer will not wait in v' under an optimal strategy and therefore he must use an edge immediately, which proves hardness for (7).

That said, it is ϵ-optimal, for any $\epsilon > 0$, to wait for a duration of $\frac{\epsilon}{M}$ in v' and then make the optimal choice in either cases, when starting in v' at time 0. This explains why the ϵ-optimal variant of AllOptimalStrategiesSPTG does not appear in our list.

Note that parametrising the problems with time t, instead of always using time 0, trivially makes the questions even harder. Also, using techniques similar to what we use for shifting in our construction allow us to show hardness for any of these problems for a given time $t \in (0, 1)$. Finally, as shown by [16], finding $\text{val}(v, 1)$ and the optimal and ϵ-optimal choice at time 1 can be solved in time $O(m + n \log n)$ and is thus in P.

6 Properties of our hard instances

The instances that we have constructed actually lie in a very restricted class of graphs, which we describe in this section.

The exponential-many event point games. In Section 3 the family of games that we constructed are all DAGS with degree four, as seen in Figure 1. In [15], we show that by slightly modifying the graph, this can actually be reduced to a DAG with degree three. Furthermore, there are only two distinct holding rates namely the ones in $\{0, 1\}$.

The game also has a planar graph. This can be seen by redrawing Figure 1 in the following way. The crossing of edges in the middle of each level can be eliminated by taking each edge (v_r^i, v_ℓ^{i-1}) and making it "wrap-around" under the structure by passing v_r^0 on the right before going to the left side and moving up.

While proving an upper bound on the number of event points in a class of games similar, but more general than SPTGs, the authors of [11, 20] use a technique based on adding more and more *urgent* states to the game. A state is urgent if the owner is not allowed to wait in it. In our construction with exponentially-many event points, the minimizer would not want to wait in a state with rate 1, and the maximizer would not want to wait in a state with rate 0, because in both cases this is the worst possible rate for them. So the optimal strategies only wait in the state v_r^0. Therefore, making any number of states, besides v_r^0, urgent does not change the value functions. Hence, our results show that, while games with no non-urgent states are easy (because they can be solved as a priced game) games with a *single* non-urgent state still may have an exponential number of event points.

In [15] we give a more in depth argument for this and also argue that each member of the family have pathwidth, treewidth, cliquewidth and rankwidth three.

The PSPACE-hard games. The PSPACE-hard games add several extra gadgets to the exponentially-many event point games. These gadgets essentially form a directed tree structure, whose leafs have outgoing edges to a unique copy of one of our exponential lower bound games. Hence, the games continue to be DAGs and planar (because no edge goes "over" the top states in our exponentially-many event point games), and the gadgets can also be constructed so that the games continue to have degree three. In [15], we give a more in depth argument for this and also argue that each member of the family have treewidth, cliquewidth and rankwidth three. We lose bounded pathwidth as a property, which is caused by the large tree of states that we add to construct our gadgets.

For the NP, coNP, and polynomial time hierarchy results, we show in [15] that a variant of our constructions (that is not planar and where the treewidth, cliquewidth and rankwidth is 4 instead of 3) has the property that for NP and coNP hard instances there are only 2 states that cannot be made urgent. Each alternation adds one extra state that cannot be made urgent. Hence, it is NP-hard to solve games with 2 non-urgent states and hard for the k-th level of the polynomial time hierachy to solve games with $k + 1$ non-urgent states.

7 Upper bounds for undirected graphs, trees and DAGs

In Section 3 and Section 6, we showed that there are an exponential number of event points for SPTGs belonging to even very restrictive graph classes. In this section we show that there are some classes of games in which there is at most a polynomial number of event points. Specifically, this holds for undirected graphs and trees. It then follows by [16] that the event point iteration algorithm runs in polynomial-time for these problems.

Secondly, we show that SPTGs on DAGs are in PSPACE. The result extends to OCPTGs because of a reduction by [16].

Our main result implies that they are PSPACE-hard and thus, this shows that they are PSPACE-complete.

Undirected graphs. The trick is to consider that whenever play goes to a maximizer state v at some time t from some other state v', the maximizer can choose to send the play immediately back to state v'. Because our strategies are time-positional, if the maximizer follows this strategy, and the play then ever goes to a maximizer state v from some other state v', the play will continue going back and forth between v and v' forever, and therefore never reach a goal state. The outcome is therefore ∞, which is the best possible for the maximizer, and so we can assume that he will adopt this strategy.

As shown in [16], if $\text{val}(v, t') = \infty$ for some t', then $\text{val}(v, t) = \infty$ for all t and $\text{val}(v, 1)$ can be found in $O(m + n \log n)$. In the remaining states, we can assume that maximizer states cannot be entered. This allows us to solve the minimizer and goal states as a sub-game first (which can be done in polynomial time since it is a priced timed automata [16, 19]). The remaining maximizer states are also easy to solve in polynomial time once this has been done. Full details of the argument can be found in [15].

Trees. The argument for trees is also fairly straightforward, in that the following lemma (see [15]) can easily be shown, using structural induction and how value functions are computed by the value iteration algorithm.

We will say that a line segment L is covered by a line or line segment L' if $L \subseteq L'$ and also extend the notion to sets, i.e. a set S is covered by a set S' if each element $L \in S$ is covered by some element $L' \in S'$ (which may depend on L).

Lemma 2. *Consider a state s which is the root of a tree with k leaves, for some number k. Then, let L_s be the line segments of $\text{val}(s, t)$. There exists a set L_k of k lines that covers L_s.*

Because there can be at most $\frac{k(k-1)}{2}$ intersections of k lines, that is also a bound on the number of line segments of $\text{val}(v, t)$. This, in turn, means that there are at most $O(n^3) = O(nk^2)$ many line segments for $\text{val}(v, t)$ over all n states of the graph. Because an event point is the time coordinate of an end point of some line segment of $\text{val}(v, t)$ for some state v, we therefore have at most $O(n^3)$ event points.

DAGs. To show that DAGs are in PSPACE, we will first argue that the denominator of each event point t^* and number $\text{val}(v, t^*)$ for all v can be expressed in polynomial space. For a natural number c, we say that a fraction x is *c-expressible* if the denominator d of x is such that $d \cdot k = c$ for some natural number k. In [15], we show the following lemma.

Lemma 3. *Consider a SPTG on a DAG of depth h with integer holding rates. Let $R = \Pi_{v_1, v_2 \in V | r(v_1) \neq r(v_2)} |r(v_1) - r(v_2)|$. Let v be some state at depth h_v and (x, y) some end point of a line segment of $\text{val}(v, t)$. If $y = \infty$, then $\text{val}(v, t) = \infty$ and otherwise, if $y \neq \infty$, the numbers x and y are R^{h-h_v}-expressible.*

We can find the set of states for which $\text{val}(v, t) = \infty$ in time $O(m + n \log n)$ by using techniques from [16]. Specifically, that paper shows that if $\text{val}(v, t') = \infty$ for some t' then $\text{val}(v, t) = \infty$ for all t, and that paper also shows how to find $\text{val}(v, 1)$ in time $O(m + n \log n)$ for all v.

For the remaining states, note that R^{h-1} can be described in polynomial space, since it is a product of $\leq hn^2$ numbers, each of which are bounded by the largest holding rate. In turn, we can also bound the numerators as using at most polynomial space and thus all the numbers.

This, in turn, means that a variant of the event point iteration algorithm given in [16] (that does not store the end points of line segments of $\text{val}(v, t)$, which is only used for the output) runs in polynomial space (see [15] for pseudocode for the event point iteration algorithm), because it then stores only t^* and $\text{val}(v, t^*)$ for all v at any one point for some event point t^*. That can then find $\text{val}(v, t')$ for some given v, t' by finding the value $\text{val}(v, t^*)$ for the smallest event point $t^* > t$ and how $\text{val}(v, t)$ behaves between t^* and the next smaller event point (which is how the algorithm iterates over the event points). Thus, it can solve the decision question we are interested in. We give more details of this argument in [15].

***Games with holding rates* $\{0, 1\}$.** In [14], it was previously claimed (fixed in arXiv:1404.5894v6) that an SPTG with holding rates $\{0, 1\}$ and integer costs can be solved in polynomial time because, it was claimed, such games would have only polynomially-many event points. Our results, however, show that this claim is incorrect: We show in [15] how to convert our examples with exponentially-many event points and holding rates $\{0, 1\}$ to have integer costs.

8 Open questions

While our results show that one-clock priced timed games and many special cases are PSPACE-hard, there are still a number of open questions.

The biggest open question for priced timed games is likely the complexity of two-clock priced timed games. That said, a number of other models related to priced timed games have been considered and there is often a jump in complexity when going from one clock to two or more clocks in those models, as we mentioned in the introduction. Also, many questions related to three or more clocks for priced timed games are undecidable [7, 10, 12]. This suggests that similar questions for the case of two clocks are also undecidable.

Besides that, we show that the complexity of priced timed games is PSPACE-hard. Previous work have shown them to be solvable in exponential time [16, 20], which does leave a gap. A possible way to resolve the question is to show a conjecture by [11]. If, as conjectured by [11], the number of iterations of the value iteration algorithm is polynomial, the problem is PSPACE-complete, since DAGs are in PSPACE, as we show, and the value iteration algorithm in essence turns the game into a DAG with states polynomial in the number of iterations and the number of states of the game.

Let ℓ be the number of event points. We show $\ell \geq 2^{n/2}/2$. Previous work [16] has shown that $\ell \leq 12^n$ for SPTGs. This means that $\ell = 2^{\Theta(n)}$, but this is quite a wide gap, and one could work on making it smaller.

We have shown that priced timed games on DAGs with one clock are PSPACE-complete, but the best result for DAGS with more clocks [1] is exponential. For DAGs the results for more clocks seems similar to the one clock case though: The value iteration algorithm runs in exponential time (see [1] for the upper bound on more clocks, and we show the lower bound for one clock). There is an exponential number of areas (called event points for one clock) where the strategy should change (see [1] for the bounds for more clocks, the upper bound extending to one clock, and we have the lower bound for one clock). Does our PSPACE upper bound generalise to more clocks?

While we show PSPACE-hardness for several special cases of one-clock priced timed games, a number of special cases may still have polynomial time algorithms:

- *Constant pathwidth.* We show that each member of our family that has exponentially many event points has pathwidth 3, but no computational-complexity hardness is shown and it is plausible that they are easier than the general case.
- *Pseudo-polynomial time algorithm for costs.* Our constructions use costs that double as we double the number of event points. To avoid the lower bounds, one could consider pseudo-polynomial time algorithms.
- *A player with few states.* Our PSPACE-hard construction has a nearly equal number minimizer and maximizer states. Conversely, for automata (i.e., when only one player has states), the corresponding problem is in NL [19]. Can one design fast algorithms for the case where one player only has a few states? Here, few could mean either constant or one could do a parametrized analysis.
- *Very limited graph width.* We show hardness for games with treewidth, cliquewidth and rankwidth three, but the cases of lower treewidths, cliquewidths and rankwidths are still open (apart from trees, which we have shown in Section 7 can be solved in polynomial time).

References

[1] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. 2004. Optimal Reachability for Weighted Timed Games. In *Proc. of ICALP*. 122–133.

[2] Rajeev Alur, Salvatore La Torre, and George J. Pappas. 2001. Optimal Paths in Weighted Timed Automata. In *Proc. of HSCC*. 49–62.

[3] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. 2007. UPPAAL-Tiga: Time for Playing Games!. In *Proc. of CAV*. 121–125.

[4] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. 2001. Minimum-Cost Reachability for Priced Time Automata. In *Proc. of HSCC*. 147–161.

[5] Patricia Bouyer. 2006. Weighted Timed Automata: Model-Checking and Games. *Electronic Notes in Theoretical Computer Science* 158 (2006), 3 – 17.

[6] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. 2007. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design* 31, 2 (2007), 135–175.

[7] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. 2006. Improved undecidability results on weighted timed automata. *Inform. Process. Lett.* 98, 5 (2006), 188 – 194.

[8] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. 2004. Optimal Strategies in Priced Timed Game Automata. In *Proc. of FSTTCS*. 148–160.

[9] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. 2005. Synthesis of Optimal Strategies Using HyTech. *Electronic Notes in Theoretical Computer Science* 119, 1 (2005), 11 – 31.

[10] Patricia Bouyer, Samy Jaziri, and Nicolas Markey. 2015. On the Value Problem in Weighted Timed Games. In *Proc. of CONCUR*. 311–324.

[11] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. 2006. Almost Optimal Strategies in One Clock Priced Timed Games. In *Proc. of FSTTCS*. 345–356.

[12] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. 2005. On Optimal Timed Strategies. In *Proc. of FORMATS*. 49–64.

[13] Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. 2015. Simple Priced Timed Games are not That Simple. In *Proc. of FSTTCS*. 278–292.

[14] Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. 2014. Adding Negative Prices to Priced Timed Games. In *Proc. of CONCUR*. 560–575.

[15] John Fearnley, Rasmus Ibsen-Jensen, and Rahul Savani. 2020. One-Clock Priced Timed Games are PSPACE-hard. *CoRR* abs/2001.04458 (2020). https://arxiv.org/abs/2001.04458

[16] Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. 2013. A Faster Algorithm for Solving One-Clock Priced Timed Games. In *Proc. of CONCUR*. 531–545.

[17] Marcin Jurdziński and Ashutosh Trivedi. 2007. Reachability-Time Games on Timed Automata. In *Proc. of ICALP*. 838–849.

[18] Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. 2002. Optimal-Reachability and Control for Acyclic Weighted Timed Automata. In *Proc. of IFIP 17th World Computer Congress — TC1 Stream*. 485–497.

[19] F. Laroussinie, N. Markey, and Ph. Schnoebelen. 2004. Model Checking Timed Automata with One or Two Clocks. In *Proc. of CONCUR*. 387–401.

[20] Michal Rutkowski. 2011. Two-Player Reachability-Price Games on Single-Clock Timed Automata. In *Proc. of QAPL*. 31–46.

[21] Ashutosh Trivedi. 2011. *Competitive Optimisation on Timed Automata*. Ph.D. Dissertation. University of Warwick.