Controlling a Random Population is EXPTIME-hard

Corto Mascle
ENS Paris-Saclay, FR
corto.mascle@ens-paris-saclay.fr

Mahsa Shirmohammadi
CNRS, IRIF, Université de Paris, FR
mahsa@irif.fr

Patrick Totzke
University of Liverpool, UK
totzke@liverpool.ac.uk

Abstract

Bertrand et al. [1] describe two-player zero-sum games in which one player tries to achieve a reachability objective in n games (on the same finite arena) simultaneously by broadcasting actions, and where the opponent has full control of resolving non-deterministic choices. They show EXPTIME completeness for the question if such games can be won for every number n of games.

We consider the *almost-sure* variant in which the opponent randomizes their actions, and where the player tries to achieve the reachability objective eventually with probability one. The lower bound construction in [1] does not directly carry over to this randomized setting.[1] In this note we show EXPTIME hardness for the almost-sure problem by reduction from Countdown Games.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Markov decision processes; Theory of computation $\rightarrow$ Network games

Keywords and phrases Markov Decision Processes, Synchronization

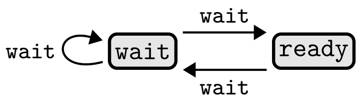
1 Definitions

Population Control. Write $\mathcal{M} = (Q, \Sigma, \delta)$ for a Markov Decision Process, where Q and Σ, are finite sets of *states* and *actions*, and $\delta : Q \times \Sigma \rightarrow \mathit{Dist}(Q)$ assigns to each state and action a probability distribution over states. A *successor* of q on action a is a state p with $\delta(q, a)(p) > 0$. The n-fold synchronized product of $\mathcal{M}$ is the MDP $\mathcal{M}^{(n)} = (Q^{(n)}, \Sigma, \delta)$ whose states, called *configurations* here, are n-dimensional vectors with components in Q, and δ is lifted to $Q^{(n)}$ in the natural way: $\delta(\mathbf{q}, a)(\mathbf{p}) = \prod_{i=0}^{n-1} \delta(\mathbf{q}(i), a)(\mathbf{p}(i))$ for all $\mathbf{q}, \mathbf{p} \in Q^{(n)}$ and $a \in \Sigma$. A *strategy* $\sigma : Q^{(n)} \rightarrow \Sigma$ assigns to every configuration $\mathbf{q}$ an action[2]. For every initial state $\mathbf{q} \in Q^{(n)}$, such a strategy induces a probability space $(\Omega, \mathcal{P}^{\sigma}_{\mathbf{q}})$ over all infinite sequences in $\Omega = \mathbf{q}(Q^{(n)})^{\omega}$ (see [3] for details).

In a configuration $\mathbf{q} \in Q^{(n)}$, we say m components *mark* state $p \in Q$ if the number of different indices $0 \leq i < n$ with $\mathbf{q}(i) = p$ is equal to m. In case $m > 0$ we simply call the state p *marked* in $\mathbf{q}$. Let $\mathbf{Start}, \mathbf{End} \in Q^{(n)}$ denote the configurations in which all n components mark a designated initial (or final, resp.) state of $\mathcal{M}$. A configuration $\mathbf{q} \in Q^{(n)}$ *can be synchronized* if there exists a strategy σ such that the probability $\mathcal{P}^{\sigma}_{\mathbf{q}}((Q^{(n)})^{*}\mathbf{End})$, of eventually visiting $\mathbf{End}$, is one. $\mathcal{M}^{(n)}$ *can be synchronized* if $\mathbf{Start}$ can be synchronized.

Given an MDP $\mathcal{M}$ equipped with initial and final states, the PopulationControl problem asks whether $\mathcal{M}^{(n)}$ can be synchronized for every $n \in \mathbb{N}$.

[1] The construction in Theorem 6.1 would allow spurious wins for the controller by alternating actions `init` and `reset` until an incorrect initialization of the ATM is reached. This must happen eventually with probability one and can subsequently be exploited by controller to win directly.
[2] We consider here only memoryless deterministic strategies as those suffice for almost-sure reachability problems [3].



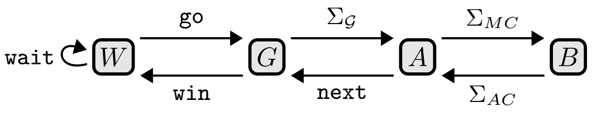


Figure 1 The waiting (on left) and the control gadgets (on right). Edges labelled by Σ_X are shorthand for several edges, one for each action in Σ_X. All but the depicted actions are daemonic.

Countdown Games. A *Countdown Game* is given by a directed graph $\mathcal{G} = (V, E)$, where edges carry positive integer weights, $E \subseteq (V \times \mathbb{N}_{>0} \times V)$. For an initial pair $(v, c_0) \in V \times \mathbb{N}$ of a vertex and a number, two opposing players (Player 1 and 2) alternatingly determine a sequence of such pairs as follows. In each round, from (v, c), Player 1 picks a number $d \leq c$ such that E contains at least one edge (v, d, v'); then Player 2 picks one such edge and the game continues from $(v', c - d)$. Player 1 wins the game iff the play reaches a pair in $V \times \{0\}$.

CountdownGame is the decision problem which asks if Player 1 has a strategy to win a given game for a given initial pair (v_0, c_0). All constants in the input are written in binary.

▶ **Proposition 1** (Thm. 4.5 in [2]). CountdownGame *is* EXPTIME-*complete.*

2 The Reduction

In order to reduce CountdownGame to PopulationControl we first observe that the number of turns in a Countdown Game cannot exceed the initial value of the counter, as the initial counter value decreases at each turn. Thus, if Player 2 has a winning strategy, choosing actions at random yields a positive probability of applying that strategy, hence a positive probability of winning. Therefore Player 1 wins the initial game if, and only if, she wins with probability one against a randomized adversary.

The main idea for our further construction is to require Player 1 to move components one-by-one away from a waiting state, first into the control graph of the Countdown Game, and ultimately into the goal. To avoid a loss in the intermediate phase she needs to win an instance of that game against a randomizing opponent. This is enforced using a combination of gadgets, including two binary counters that can effectively test for zero, be set to specific numbers, and that are set up so that they can decrement at the same rate. As a result, Player 1 has a winning strategy for the two-player Countdown Game if, and only if, the controller can synchronize the n-fold product of the constructed MDP for all n.

For a given Countdown Game $\mathcal{G}$ with an initial pair (v_0, c_0) we construct an MDP $\mathcal{M}$ as follows. We write that action a *takes state* s to *successor* t to mean that $\delta(s, a)(t) > 0$. The exact probability distributions δ do not matter in our construction so we let $\delta(s, a)$ be the uniform distribution over such successors.

Whenever action a takes state s only back to itself we say that s *ignores* a. There are states `Heaven` (the target) and `Hell` which ignore all actions. For a given state s, an action a is *angelic* if it takes s only to `Heaven`, and *daemonic* if it takes s to `Hell`. An action a is *safe* in a configuration if it is not daemonic for any marked state (in any gadget).

Besides the special states `Heaven` and `Hell`, $\mathcal{M}$ contains several gadgets described below.

Waiting. The waiting gadget has two states `Wait` and `Ready` which react to the action `wait` as depicted in Figure 1 (left). Whenever a configuration marks one of these states, a strategy that continuously plays `wait` will almost-surely reach a configuration in which exactly one component marks `Ready`.

A special action `go` (to indicate successful isolation of one component) takes `Ready` to the initial state v_0 of the game $\mathcal{G}$. All other actions (in gadgets described below) are ignored.

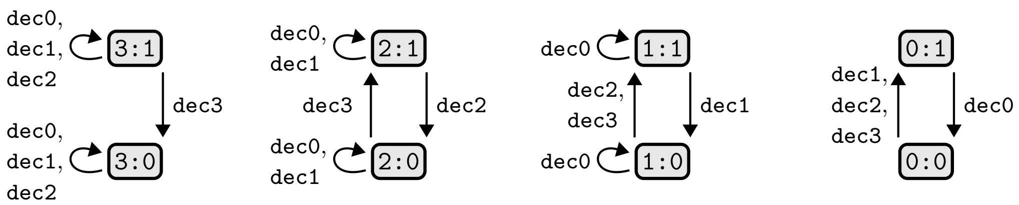


Figure 2 A (4-bit) Binary Counter. Not displayed are edges labelled by `(deci)` that make the respective actions daemonic for state `(i:0)`, and error actions `errori`, which are daemonic for `(i:0)` and `(i:1)`, for all bits $i \in \{0, 1, 2, 3\}$.

Game. The game $\mathcal{G} = (G, E)$ is directly interpreted as MDP: For every edge $(s, d, s') \in E$ there is an action (s, d) which takes s to s' and which is daemonic for all states $s' \neq s$.

The action `win` is is angelic for every state of G. All other actions are ignored.

Binary Counters. A (k-bit) Counter consists of states `(i:j)` for all $0 \leq i < k$ and $j \in \{0, 1\}$. For every bit i there is a decrement action `(deci)` which

- takes `(j:0)` only to `(j:1)` for all $0 \leq j < i$,
- takes `(i:1)` only to `(i:0)`,
- is daemonic for `(i:0)`, and
- is ignored by all `(j:l)`, for all $i < j$ and $l \in \{0, 1\}$.

We say that a configuration *holds* the number $c < 2^k$ in this counter if it marks those states that represent the binary expansion of c: for all $0 \leq i \leq k - 1$, state `(i:j)` is marked iff the ith bit in the binary expansion of c is j. An action a *sets* the counter to number d if for all $0 \leq i < k$, it takes `(i:0)` to only `(i:j)` where $j \in \{0, 1\}$ is the ith bit in the binary expansion of d, and is daemonic for all `(i:1)` (to ensure that the counter can only be set if it holds 0). Observe that if a counter holds c then there is a unique maximal sequence of safe decrement actions, that has length c and after which the counter holds 0.

Additionally, for every bit i the gadget has an error action `errori`, which is daemonic for `(i:0)` and `(i:1)`, and angelic for every other state (of $\mathcal{M}$). These actions can be used to quickly synchronize any configuration in which the counter is not correctly initialized, i.e., does not hold a number. See Figure 2 for a depiction of a 4-bit counter.

The MDP $\mathcal{M}$ will contain two distinct counter gadgets. A main counter MC has $\log_2(n_0)$ bits to hold possible counter values of the Countdown Game. An auxiliary counter AC has $\log_2(d_{\texttt{max}})$ many bits to hold the largest edge weight $d_{\texttt{max}}$ in $\mathcal{G}$. These have distinct sets of states and actions, so for clarity, we write $C.x$ to refer to state (or action) x in gadget C. We connect some new actions to these two counters as follows.

- The action `go` sets MC to n_0; this ensures that MC holds n_0 when starting to simulate $\mathcal{G}$.
- The action `win` is daemonic for every state $MC.$`(i:1)`. This enforces that the MC must hold 0 when a strategy claims Player 1 wins $\mathcal{G}$.
- Any action $(v, d) \in \Sigma_{\mathcal{G}}$ sets AC to d;
- The action `next` is daemonic for every state $AC.$`(i:1)`. This enforces that a strategy must first count down from d to 0 before it can simulate the next move in $\mathcal{G}$.

Control. The control gadget will enforce that a synchronizing strategy proposes actions in a proper order; see Figure 1. It consists of states W, G, A, B, and contains actions of all gadgets above (including `go`, `win`, `next`) and a new `error` action, which is angelic for all states except W, for which it is daemonic. All omitted edges in Figure 1 are daemonic.

Start/End. To complete the construction of $\mathcal{M}$, we introduce an initial state `Init` and actions `start` and `end`. The action `start` takes `Init` to `Wait` (Waiting gadget), W (Control gadget), and all `(i:0)` states of counters AC and MC. It is daemonic for every other state.

The action `end` is daemonic for `Wait` and `Ready`, and angelic for every other state in $\mathcal{M}$.

▶ **Lemma 2.** *$\mathcal{M}^{(n)}$ is synchronizable for all $n \in \mathbb{N}$ iff Player 1 wins $\mathcal{G}$.*

Proof. Suppose Player 1 wins the game $\mathcal{G}$. Fix n. Recall that in $\mathcal{M}^{(n)}$ all components of the initial configuration mark `Init`. A synchronizing strategy proceeds as follows:

- Play `start` to initialize the Waiting and Control gadgets, and to set AC and MC to 0. If any of the gadgets is not correctly initialized afterwards, play the respective error action to win directly. For instance, if W is unmarked, play `error` to synchronize.
- Reduce the number of components marking `Wait` one by one until a configuration is reached in which `Wait` is not marked. Once this is true, play `end` to synchronize.
- To reduce the number of components marking `Wait`, isolate one of them, and move it to `Heaven` by simulating the Countdown Game:
  1. Play `wait` until only a single component marks `Ready`, then play `go`. This will mark v_0 in the game gadget and sets MC to n_0. Recall that (v_0, n_0) is the initial pair of $\mathcal{G}$.
  2. Simulate rounds of the game $\mathcal{G}$: assume state v in the game gadget is marked and the counter MC holds c, then let d be the the number Player 1 plays to win from the pair (v, c) in $\mathcal{G}$. Play (v, d). This action will set AC to d. Alternate between (safe) decrement actions in AC and AB until they hold 0 and $c - d$, respectively. Play `next`.
  3. The above simulation of rounds in $\mathcal{G}$ is repeated until both AC and AB hold 0, by assumption that Player 1 wins $\mathcal{G}$ this is possible. At this point it is safe to play `win`.

Conversely, assume that Player 1 cannot win $\mathcal{G}$. Suppose that after the (only possible) initial move `start`, all gadgets are correctly initialized. Clearly, for every n, this event has strictly positive probability. We argue that no strategy can synchronize such a configuration. Indeed, a successful strategy had to play a sequence in $\texttt{wait}^* \cdot \texttt{go}$ first, followed by actions in $(\Sigma_{\mathcal{G}} \cdot (\Sigma_{AC} \cdot \Sigma_{MC} \cdot \texttt{next})^*)^*$, by construction of the control gadget. If after playing `go`, more than one component mark v_0, there is a non-zero chance that these will diverge, making subsequent actions in $\Sigma_{\mathcal{G}}$ unsafe. If exactly one component marks v_0 then the second sequence of actions (assuming all actions are safe) corresponds to a play of $\mathcal{G}$. This inevitably leads to a configuration in which counter MC holds 0 and the control enforces that the next action is in Σ_{MC}. But any such action will be daemonic for some state in MC and thus not be safe. We conclude that every strategy will lead to a configuration that at least one component marks `Hell` and thus cannot be synchronized. ◀

Our claim follows immediately from Proposition 1 and Lemma 2.

▶ **Theorem 3.** PopulationControl *is* EXPTIME-*hard.*

References

1 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Logical Methods in Computer Science*, 2019.

2 Marcin Jurdziński, François Laroussinie, and Jeremy Sproston. Model checking probabilistic timed automata with one or two clocks. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2007.

3 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* 1st edition, 1994.