Controlling a population

Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, Adwait Godbole

Logical Methods in Computer Science
Volume 15, Issue 3, 2019, pp. 6:1–6:30
https://lmcs.episciences.org/

Submitted Jul. 04, 2018
Published Jul. 29, 2019

CONTROLLING A POPULATION

NATHALIE BERTRAND, MIHEER DEWASKAR, BLAISE GENEST, HUGO GIMBERT, AND ADWAIT AMIT GODBOLE

Univ Rennes, Inria & IRISA, France
e-mail address: nathalie.bertrand@inria.fr

University of North Carolina at Chapel Hill, USA
e-mail address: miheer@live.unc.edu

Univ Rennes, CNRS, IRISA, France
e-mail address: blaise.genest@irisa.fr

CNRS & LaBRI, France
e-mail address: hugo.gimbert@labri.fr

IIT Bombay, India
e-mail address: godbole15@gmail.com

Abstract. We introduce a new setting where a population of agents, each modelled by a finite-state system, are controlled uniformly: the controller applies the same action to every agent. The framework is largely inspired by the control of a biological system, namely a population of yeasts, where the controller may only change the environment common to all cells. We study a synchronisation problem for such populations: no matter how individual agents react to the actions of the controller, the controller aims at driving all agents synchronously to a target state. The agents are naturally represented by a non-deterministic finite state automaton (NFA), the same for every agent, and the whole system is encoded as a 2-player game. The first player (Controller) chooses actions, and the second player (Agents) resolves non-determinism for each agent. The game with m agents is called the m-population game. This gives rise to a parameterized control problem (where control refers to 2 player games), namely the *population control problem*: can Controller control the m-population game for all $m \in \mathbb{N}$ whatever Agents does?

In this paper, we prove that the population control problem is decidable, and it is a EXPTIME-complete problem. As far as we know, this is one of the first results on the control of parameterized systems. Our algorithm, which is not based on cut-off techniques, produces winning strategies which are symbolic, that is, they do not need to count precisely how the population is spread between states. The winning strategies produced by our algorithm are optimal with respect to the synchronisation time: the maximal number of steps before synchronisation of all agents in the target state is at most polynomial in the number of agents m, and exponential in the size of the NFA. We also show that if there is no winning strategy, then there is a population size M such that Controller wins the m-population game if and only if $m \leq M$. Surprisingly, M can be doubly exponential in the number of states of the NFA, with tight upper and lower bounds.

Key words and phrases: Logic and verification, control, parametric systems.

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.23638/LMCS-15(3:6)2019

1. Introduction

Finite-state controllers, implemented by software, find applications in many different domains: telecommunication, aeronautics, etc. Many theoretical studies from the model-checking community showed that, in idealised settings, finite-state controllers suffice. Games are an elegant formalism to model control problems [5]: players represent the controller and the system; the precise setting (number of players, their abilities, and their observations) depends on the context.

Recently, finite-state controllers have been used to control living organisms, such as a population of yeasts [23]. In this application, microscopy is used to monitor the fluorescence level of a population of yeasts, reflecting the concentration of some molecule, which differs from cell to cell. Finite-state systems can model a discretisation of the population of yeasts [23, 3]. The frequency and duration of injections of a sorbitol solution can be controlled, being injected uniformly into a solution in which the yeast population is immersed. However, the response of each cell to the osmotic stress induced by sorbitol varies, influencing the concentration of the fluorescent molecule. The objective is to control the population to drive it through a sequence of predetermined fluorescence states.

Taking inspiration from this biological control problem, we propose in this paper, an *idealised* problem for the population of yeasts: the (perfectly-informed) controller aims at leading synchronously all agents to a given fluorescence state. We introduce the *m-population game*, where a population of m identical agents is controlled uniformly. Each agent is modeled as a nondeterministic finite-state automaton (NFA), the same for each agent. The first player, called Controller, applies the same action, a letter from the NFA alphabet, to every agent. Its opponent, called Agents, chooses the reaction of each individual agent, that is their successor state upon that action. These reactions can differ due to non-determinism. The objective for Controller is to gather all agents synchronously in the target state, and Agents seeks the opposite objective. Our idealised setting may not be entirely satisfactory, yet it constitutes a first step towards more realistic formalisations of the yeast population control problem.

Dealing with large populations *explicitly* is in general intractable due to the state-space explosion problem. We therefore consider the associated *symbolic parameterized control problem*, that requires to synchronise all agents, independently of the population size. Interestingly, this population control problem does not fit traditional game frameworks from the model-checking community. While *parameterized verification* received recently quite some attention (see the related work below), to the best of our knowledge, our framework is among the first ones in *parameterized control*.

Our results. We first show that considering an infinite population is not equivalent to the parameterized control problem: there are simple cases where Controller cannot control an infinite population but can control every finite population. Solving the ∞-population game reduces to checking a reachability property on the support graph [21], which can be easily done in PSPACE. On the other hand, solving the parameterized control problem requires new proof techniques, data structures and algorithms.

We easily obtain that when the answer to the population control problem is negative, there exists a population size M, called the *cut-off*, such that Controller wins the m-population game if and only if $m \leq M$. Surprisingly, we obtain a lower-bound on the cut-off doubly exponential in the number of states of the NFA. Exploiting this cut-off naively would thus yield an inefficient algorithm of least doubly exponential time complexity.

Fortunately, developing new proof techniques (*not* based on cut-off), we manage to obtain a better complexity: we prove the population control problem to be EXPTIME-complete. As a byproduct, we obtain a doubly exponential upper-bound for the cut-off, matching the lower-bound. Our techniques are based on a reduction to a parity game with exponentially many states and a polynomial number of priorities. The constructed parity game, and associated winning strategies, gives insight on the winning strategies of Controller in the m-population games, for all values of m. Controller selects actions based on a polynomial number of *transfer graphs*, describing the trajectory of agents before reaching a given state. If Controller wins this parity game then he can uniformly apply his winning strategy to all m-population games, just keeping track of these transfer graphs, independently of the exact count in each state. If Agents wins the parity game then he also has a uniform winning strategy in m-population games, for m large enough, which consists in splitting the agents evenly among all transitions of the transfer graphs.

Last, we obtain that when the answer to the population control problem is positive, the controller built by our algorithm takes at most a polynomial number of steps to synchronize all agents in the winning state, where the polynomial is of order the number of agents power the number of states of the NFA. We show that our algorithm is optimal, as there are systems which require at least this order of steps to synchronize all agents.

Related work. Parameterized verification of systems with many identical components started with the seminal work of German and Sistla in the early nineties [16], and received recently quite some attention. The decidability and complexity of these problems typically depend on the communication means, and on whether the system contains a leader (following a different template) as exposed in the recent survey [13]. This framework has been extended to timed automata templates [2, 1] and probabilistic systems with Markov decision processes templates [6, 7]. Another line of work considers population protocols [4, 15, 8]. Close in spirit, are broadcast protocols [14], in which one action may move an arbitrary number of agents from one state to another. Our model can be modeled as a subclass of broadcast protocols, where broadcasts emissions are self loops at a unique state, and no other synchronisation allowed. The parameterized reachability question considered for broadcast protocols is trivial in our framework, while our parameterized control question would be undecidable for broadcast protocols. In these different works, components interact directly, while in our work, the interaction is indirect via the common action of the controller. Further, the problems considered in related work are verification questions, and do not tackle the difficult issue we address of synthesizing a controller for all instances of a parameterized system.

There are very few contributions pertaining to parameterized games with more than one player. The most related is [20], which proves decidability of control of mutual exclusion-like protocols in the presence of an unbounded number of agents. Another contribution in that domain is the one of broadcast networks of identical parity games [7]. However, the game is used to solve a verification (reachability) question rather than a parameterized control problem as in our case. Also the roles of the two players are quite different.

The winning condition we are considering is close to *synchronising words*. The original synchronising word problem asks for the existence of a word w and a state q of a *deterministic* finite state automaton, such that no matter the initial state s, reading w from s would lead to state q (see [24] for a survey). Lately, synchronising words have been extended to NFAs [21]. Compared to our settings, the author assumes a possibly infinite population of agents. The setting is thus not parameterized, and a usual support arena suffices to

obtain a PSPACE algorithm. Synchronisation for probabilistic models [11, 12] have also been considered: the population of agents is not finite nor discrete, but rather continuous, represented as a distribution. The distribution evolves deterministically with the choice of the controller (the probability mass is split according to the probabilities of the transitions), while in our setting, each agent moves nondeterministically. In [11], the controller needs to apply the same action whatever the state the agents are in (similarly to our setting), and then the existence of a controller is undecidable. In [12], the controller can choose the action depending on the state each agent is in (unlike our setting), and the existence of a controller reaching uniformly a set of states is PSPACE-complete.

Last, our parameterized control problem can be encoded as a 2-player game on VASS [9], with one counter per state of the NFA: the opponent gets to choose the population size (a counter value), and the move of each agent corresponds to decrementing a counter and incrementing another. Such a reduction yields a symmetrical game on VASS in which both players are allowed to modify the counter values, in order to check that the other player did not cheat. Symmetrical games on VASS are undecidable [9], and their asymmetric variant (only one player is allowed to change the counter values) are decidable in 2EXPTIME [19], thus with higher complexity than our specific parameterized control problem.

An extended abstract of this work appeared in the proceedings of the conference CONCUR 2017. In comparison, we provide here full proofs of our results, and added more intuitions and examples to better explain the various concepts introduced in this paper. Regarding contributions, we augmented our results with the study of the maximal time to synchronisation, for which we show that the controller built by our algorithm is optimal.

Outline. In Section 2 we define the population control problem and announce our main results. Section 3 introduces the capacity game, and shows its equivalence with the population control problem problem. Section 4 details the resolution of the capacity game in EXPTIME, relying on a clever encoding into a parity game. It also proves a doubly exponential bound on the cut-off. Section 5 studies the maximal time to synchronisation. Section 6 provides matching lower bounds on the complexity and on the cut-off. The paper ends with a discussion in Section 7.

2. The population control problem

2.1. **The m-population game.** A nondeterministic finite automaton (NFA for short) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ with Q a finite set of states, Σ a finite alphabet, $q_0 \in Q$ an initial state, and $\Delta \subseteq Q \times \Sigma \times Q$ the transition relation. We assume throughout the paper that NFAs are complete, that is, $\forall q \in Q, a \in \Sigma\ , \exists p \in Q :\ (q, a, p) \in \Delta$. In the following, incomplete NFAs, especially in figures, have to be understood as completed with a sink state.

For every integer m, we consider a system $\mathcal{A}^m$ with m identical agents $\mathcal{A}_1, \ldots, \mathcal{A}_m$ of the NFA $\mathcal{A}$. The system $\mathcal{A}^m$ is itself an NFA $(Q^m, \Sigma, q_0^m, \Delta^m)$ defined as follows. Formally, states of $\mathcal{A}^m$ are called configurations, and they are tuples $\mathbf{q} = (q_1, \ldots, q_m) \in Q^m$ describing the current state of each agent in the population. We use the shorthand $\mathbf{q}_0[m]$, or simply $\mathbf{q}_0$ when m is clear from context, to denote the initial configuration $(q_0, \ldots, q_0)$ of $\mathcal{A}^m$. Given a target state $f \in Q$, the f-synchronizing configuration is $f^m = (f, \ldots, f)$ in which each agent is in the target state.

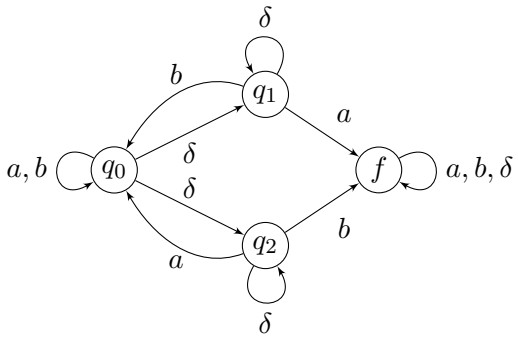


Figure 1: An example of NFA: The splitting gadget $\mathcal{A}_{\mathsf{split}}$.

The intuitive semantics of $\mathcal{A}^m$ is that at each step, the same action from Σ applies to all agents. The effect of the action however may not be uniform given the nondeterminism present in $\mathcal{A}$: we have $((q_1, \ldots, q_m), a, (q'_1, \ldots, q'_m)) \in \Delta^m$ iff $(q_j, a, q'_j) \in \Delta$ for all $j \leq m$. A (finite or infinite) play in $\mathcal{A}^m$ is an alternating sequence of configurations and actions, starting in the initial configuration: $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \cdots$ such that $(\mathbf{q}_i, a_i, \mathbf{q}_{i+1}) \in \Delta^m$ for all i.

This is the *m-population game* between Controller and Agents, where Controller chooses the actions and Agents chooses how to resolve non-determinism. The objective for Controller is to gather all agents synchronously in f while Agents seeks the opposite objective.

Our parameterized control problem asks whether Controller can win the m-population game for every $m \in \mathbb{N}$. A strategy of Controller in the m-population game is a function mapping finite plays to actions, $\sigma : (Q^m \times \Sigma)^* \times Q^m \to \Sigma$. A play $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \mathbf{q}_2 \cdots$ is said to *respect* σ, or is a *play under* σ, if it satisfies $a_i = \sigma(\mathbf{q}_0 a_0 \mathbf{q}_1 \cdots \mathbf{q}_i)$ for all $i \in \mathbb{N}$. A play $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \mathbf{q}_2 \cdots$ is *winning* if it hits the f-synchronizing configuration, that is $\mathbf{q}_j = f^m$ for some $j \in \mathbb{N}$. Controller wins the m-population game if he has a strategy such that all plays under this strategy are winning. One can assume without loss of generality that f is a sink state. If not, it suffices to add a new action leading tokens from f to the new target sink state ☺ and tokens from other states to a losing sink state ☹. The goal of this paper is to study the following parameterized control problem:

Population control problem
Input: An NFA $\mathcal{A} = (Q, q_0, \Sigma, \Delta)$ and a target state $f \in Q$.
Output: Yes iff for every integer m Controller wins the m-population game.

For a fixed m, the winner of the m-population game can be determined by solving the underlying reachability game with $|Q|^m$ states, which is intractable for large values of m. On the other hand, the answer to the population control problem gives the winner of the m-population game for arbitrary large values of m. To obtain a decision procedure for this parameterized problem, new data structures and algorithmic tools need to be developed, much more elaborate than the standard algorithm solving reachability games.

Example 2.1. We illustrate the population control problem with the example $\mathcal{A}_{\mathsf{split}}$ on action alphabet $\Sigma = \{a, b, \delta\}$ in Figure 1. Here, to represent a configuration $\mathbf{q}$, we use a counting abstraction, and identify $\mathbf{q}$ with the vector (n_0, n_1, n_2, n_3), where n_0 is the number of agents in state q_0, etc, and n_3 is the number of agents in f. Controller has a way to gather all agents synchronously to f. We can give a symbolic representation of a memoryless winning strategy

σ: $\forall k_0, k_1 > 0,\ \forall k_2, k_3 \geq 0,\ \sigma(k_0, 0, 0, k_3) = \delta,\ \sigma(0, k_1, k_2, k_3) = a,\ \sigma(0, 0, k_2, k_3) = b$. Under this strategy indeed, the number of agents outside f decreases by at least one at every other step. The properties of this example will be detailed later and play a part in proving a lower bound (see Proposition 6.3).

Example 2.2. We provide another illustrating example, requiring a more involved strategy. Consider the NFA from Figure 2, with $\Sigma = \{\texttt{try}, \texttt{retry}, \texttt{top}, \texttt{bot}, \texttt{keep}, \texttt{restart}\}$. This NFA is again a positive instance of the population control problem. Yet, in contrast with the previous example, there are unsafe moves for Controller. Indeed, after playing `try` from q_0, playing `bot` is losing if there are agents in $q_\top$, and playing `top` is losing if there are agents in $q_\bot$ (recall that unspecified transitions lead to a sink losing state). However, alternating `try` and `keep` until either $q_\bot$ becomes empty - allowing to play `top` or $q_\top$ is empty - allowing to play `bot`, and then `restart`, yields a configuration with less agents in q_0, and at least one in f. Continuing in the same way provides a winning strategy for Controller. This example will be used again in Section 5, regarding the worst-case time to synchronisation.

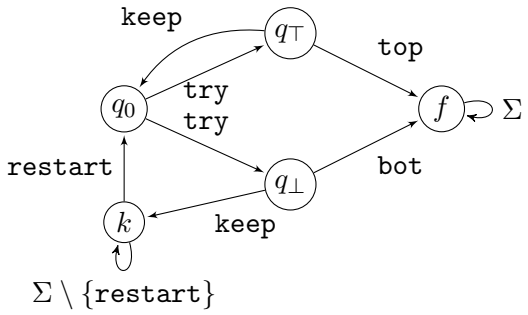


Figure 2: A second example of NFA for the population control problem: $\mathcal{A}_{\text{time}}$.

2.2. **Parameterized control and cut-off.** A first observation for the population control problem is that $\mathbf{q}_0[m]$, f^m and Q^m are stable under a permutation of coordinates. A consequence is that the m-population game is also symmetric under permutation, and thus the set of winning configurations is symmetric and the winning strategy can be chosen uniformly from symmetric winning configurations. Therefore, if Controller wins the m-population game then he has a positional winning strategy which only counts the number of agents in each state of $\mathcal{A}$ (the counting abstraction used in Example 2.1).

Proposition 2.3. *Let $m \in \mathbb{N}$. If Controller wins the m-population game, then he wins the m'-population game for every $m' \leq m$.*

Proof. Let $m \in \mathbb{N}$, and assume σ is a winning strategy for Controller in $\mathcal{A}^m$. For $m' \leq m$ we define σ' as a strategy on $\mathcal{A}^{m'}$, inductively on the length of finite plays. Initially, σ' chooses the same first action as σ: $\sigma'(q_0^{m'}) = \sigma(q_0^m)$. We then arbitrarily choose that the missing $m - m'$ agents would behave similarly as the first agent. This is indeed a possible move for the adversary in $\mathcal{A}^m$. Then, for any finite play under σ' in $\mathcal{A}^{m'}$, say $\pi' = \mathbf{q}_0^{m'} a_0 \mathbf{q}_1^{m'} a_1 \mathbf{q}_2^{m'} \cdots \mathbf{q}_n^{m'}$, there must exist an extension π of π' obtained by adding $m - m'$ agents, all behaving as the first agent in $\mathcal{A}^{m'}$, that is consistent with σ. Then, we let $\sigma'(\pi') = \sigma(\pi)$. Obviously, since σ is winning in $\mathcal{A}^m$, σ' is also winning in $\mathcal{A}^{m'}$. □

Hence, when the answer to the population control problem is negative, there exists a *cut-off*, that is a value $M \in \mathbb{N}$ such that for every $m < M$, Controller has a winning strategy in $\mathcal{A}^m$, and for every $m \geq M$, he has no winning strategy.

Example 2.4. To illustrate the notion of cut-off, consider the NFA on alphabet $\Sigma = A \cup \{b\}$ from Figure 3. Here again, unspecified transitions lead to a sink losing state ☹.

Let us prove that the cut-off is $M = |Q| - 2$ in this case. On the one hand, for $m < M$, there is a winning strategy σ_m in $\mathcal{A}^m$ to reach f^m, in just two steps. It first plays b, and because $m < M$, in the next configuration, there is at least one state q_i such that no agent is in q_i. It then suffices to play a_i to win. On the other hand, if $m \geq M$, there is no winning strategy to synchronize in f, since after the first b, agents can be spread so that there is at least one agent in each state q_i. From there, Controller can either play action b and restart the whole game, or play any action a_i, leading at least one agent to the sink state ☹.

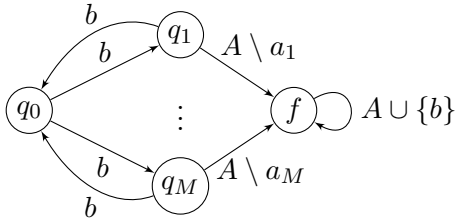


Figure 3: An NFA with a linear cut-off.

2.3. **Main results.** We are now in a position to state the contributions of this paper. Most importantly, we establish the decidability and complexity of the population control problem, with matching upper and lower bounds on complexity:

Theorem 2.5. *The population control problem is* EXPTIME*-complete.*

To prove Theorem 2.5, we proceed as follows. First, Theorem 3.7 states the equivalence of the population control problem with an involved but non-parametric control problem, called the capacity game. A simple yet suboptimal 2EXPTIME upper bound derives from this equivalence. In Theorem 4.5, we reduce the capacity game to an exponential-size parity game with polynomially many parities, yielding an EXPTIME upper bound. The matching EXPTIME-hard lower bound is proved in Theorem 6.1.

For positive instances of the population control problem, our decision algorithm computes a symbolic strategy σ, applicable to all instances $\mathcal{A}^m$, which, in particular, does not rely on the number of agents in each state. This symbolic strategy requires exponential memory. Further, it is optimal with respect to the synchronisation time, *i.e.* the maximal number of steps before synchronisation, which is polynomial in the number of agents.

Theorem 2.6. *The synchronisation time under the winning strategy σ is polynomial in the number of agents (and exponential in the size of $\mathcal{A}$). There is a family of NFA $(\mathcal{A}_n)$ with n states, such that $m^{\frac{n-2}{2}}$ steps are needed by any strategy to synchronise m agents.*

The upper bound is stated in Theorem 5.4, and the lower bound in Corollary 5.3.

For negative instances to the population control problem, the cut-off is at most doubly exponential, which is asymptotically tight.

Theorem 2.7. *In case the answer to the population control problem is negative, the cut-off is at most* $\leq 2^{2^{O(|Q|^4)}}$. *There is a family of NFA* $(\mathcal{A}_n)$ *of size* $O(n)$ *and whose cut-off is* 2^{2^n}.

Concerning the cut-off, the upper bound derives from results of Theorem 4.5 (about the size of Agents' winning strategy) combined with Proposition 3.10. The lower bound is stated in Proposition 6.3

3. The capacity game

The objective of this section is to show that the population control problem is equivalent to solving a game called the *capacity game*. To introduce useful notations, we first recall the population game with infinitely many agents, as studied in [21] (see also [22] p.81).

3.1. The ∞-population game.

To study the ∞-population game, the behaviour of infinitely many agents is abstracted into *supports* which keep track of the set of states in which at least one agent is. We thus introduce the *support game*, which relies on the notion of *transfer graphs*. Formally, a transfer graph is a subset of $Q \times Q$ describing how agents are moved during one step. The domain of a transfer graph G is $\mathrm{Dom}(G) = \{q \in Q \mid \exists (q,r) \in G\}$ and its image is $\mathrm{Im}(G) = \{r \in Q \mid \exists (q,r) \in G\}$. Given an NFA $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ and $a \in \Sigma$, the transfer graph G is compatible with a if for every edge (q,r) of G, $(q,a,r) \in \Delta$. We write $\mathcal{G}$ for the set of transfer graphs.

The *support game* of an NFA $\mathcal{A}$ is a two-player reachability game played by Controller and Agents on the *support arena* as follows. States are supports, *i.e.*, non-empty subsets of Q and the play starts in $\{q_0\}$. The goal support is $\{f\}$. From a support S, first Controller chooses a letter $a \in \Sigma$, then Agents chooses a transfer graph G compatible with a and such that $\mathrm{Dom}(G) = S$, and the next support is $\mathrm{Im}(G)$. A play in the support arena is described by the sequence $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ of supports and actions (letters and transfer graphs) of the players. Here, Agents' best strategy is to play the maximal graph possible, and we obtain a PSPACE algorithm [21], and problem is PSPACE-complete:

Proposition 3.1. *Controller wins the* ∞*-population game iff he wins the support game.*

Proof. Let $\pi = \mathbf{q}_0 a_1 \mathbf{q}_1 \dots \mathbf{q}_{n-1} a_n \mathbf{q}_n \dots$ be an infinite (or a finite) play of the ∞-population game: agent $i \in \mathbb{N}$ is in state $\mathbf{q}_k[i]$ at step k. By only observing the support of the states and the transfer graphs, we can project this play onto the support arena. More precisely, denoting $S_k = \{\mathbf{q}_k[i] \mid i \in \mathbb{N}\}$ and $G_{k+1} = \{(\mathbf{q}_k[i], \mathbf{q}_{k+1}[i]) \mid i \in \mathbb{N}\}$ for every k, we have $\Phi(\pi) = S_0 \xrightarrow{a_1,G_1} S_1 \cdots S_{n-1} \xrightarrow{a_n,G_n} S_n \cdots$ is a valid play in the *support arena*.

Hence if Controller can win the support game with strategy σ, then Controller can use the strategy $\sigma \circ \Phi$ in the ∞-population game. This is a winning strategy since the projection in the support arena should eventually reach $\{f\}$.

On the other hand if Controller doesn't have a winning strategy in the support game, then by determinacy of reachability games, Agents has a strategy in the support game to avoid reaching $\{f\}$. This strategy can be extended to a strategy in the ∞-population game by sending infinitely many agents along each edge of the chosen transfer graph. This can always be done because, inductively, there are infinitely many agents in each state. □

Perhaps surprisingly, when it comes to finite populations, the support game cannot be exploited to solve the population control problem. Indeed, Controller might win every m-population game (with $m < \infty$) and at the same time lose the ∞-population game. The example from Figure 1 witnesses this situation. As already shown, Controller wins any m-population game with $m < \infty$. However, Agents can win the ∞-population game by splitting agents from q_0 to both q_1 and q_2 each time Controller plays δ. This way, the sequence of supports is $\{q_0\}\{q_1, q_2\}(\{q_0, f\}\{q_1, q_2, f\})^*$, which never hits $\{f\}$.

3.2. **Realisable plays.** Plays of the m-population game (for $m < \infty$) can be abstracted as plays in the support game, forgetting the identity of agents and keeping only track of edges that are used by at least one agent. Formally, given a play $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \mathbf{q}_2 \cdots$ of the m-population game, define for every integer n, $S_n = \{\mathbf{q}_n[i] \mid 1 \leq i \leq m\}$ and $G_{n+1} = \{(\mathbf{q}_n[i], \mathbf{q}_{n+1}[i]) \mid 1 \leq i \leq m\}$. We denote $\Phi_m(\pi)$ the play $S_0 \xrightarrow{a_1, G_1} S_1 \xrightarrow{a_2, G_2} \dots$ in the support arena, called the projection of π.

Not every play in the support arena can be obtained by projection. This is the reason for introducing the notion of realisable plays:

Definition 3.2 (Realisable plays)**.** A play of the support game is *realisable* if there exists $m < \infty$ such that it is the projection by Φ_m of a play in the m-population game.

To characterise realisability, we introduce entries of accumulators:

Definition 3.3. Let $\rho = S_0 \xrightarrow{a_1, G_1} S_1 \xrightarrow{a_2, G_2} \dots$ be a play in the support arena. An *accumulator* of ρ is a sequence $T = (T_j)_{j \in \mathbb{N}}$ such that for every integer j, $T_j \subseteq S_j$, and which is *successor-closed i.e.*, for every $j \in \mathbb{N}$, $(s \in T_j \wedge (s,t) \in G_{j+1}) \implies t \in T_{j+1}$. For every $j \in \mathbb{N}$, an edge $(s,t) \in G_{j+1}$ is an *entry* to T if $s \notin T_j$ and $t \in T_{j+1}$; such an index j is called an *entry time*.

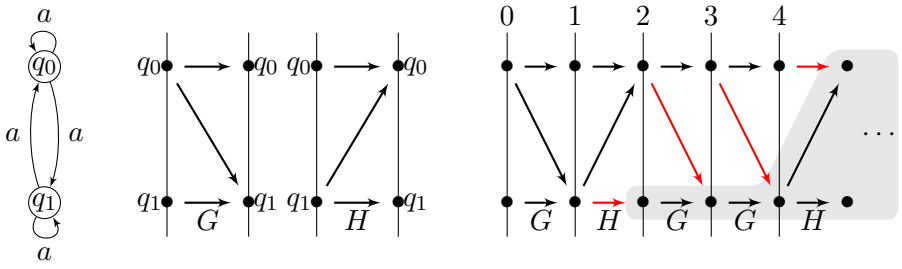


Figure 4: An NFA, two transfer graphs, and a play with finite yet unbounded capacity.

Figure 4 illustrates the notions we just introduced: it contains an NFA (left), two transfer graphs G and H (middle), and $\rho = GHG^2HG^3 \cdots$ a play in the support arena (right). The grey zone is an accumulator defined by $T_0 = T_1 = \emptyset, T_2 = T_3 = T_4 = \{q_1\}$ and $T_n = \{q_0, q_1\}$ for all $n \geq 5$.

Definition 3.4 (Plays with finite and bounded capacity)**.** A play has *finite capacity* if all its accumulators have finitely many entries (or entry times), *infinite capacity* otherwise, and *bounded capacity* if the number of entries (or entry times) of its accumulators is bounded.

Continuing with the example of Figure 4, entries of the accumulator are depicted in red. The play $\rho = GHG^2HG^3\cdots$ is not realisable in any m-population game, since at least n agents are needed to realise n transfer graphs G in a row: at each G step, at least one agent moves from q_0 to q_1, and no new agent enters q_0. Moreover, let us argue that ρ has unbounded capacity. A simple analysis shows that there are only two kinds of non-trivial accumulators $(T_j)_{j\in\mathbb{N}}$ depending on whether their first non-empty T_j is $\{q_0\}$ or $\{q_1\}$. We call these top and bottom accumulators, respectively. All accumulators have finitely many entries, thus the play has finite capacity. However, for every $n \in \mathbb{N}$ there is a bottom accumulator with $2n$ entries. Therefore, ρ has unbounded capacity, and it is not realisable.

We show that in general, realisability is equivalent to *bounded* capacity:

Lemma 3.5. *A play is realisable iff it has bounded capacity.*

Proof. Let $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \cdots$ be a realisable play in the support arena and $\pi = \mathbf{q}_0 a_1 \mathbf{q}_1 a_2 \mathbf{q}_2 \cdots$ a play in the m-population game for some m, such that $\Phi_m(\pi) = \rho$. For any accumulator $T = (T_j)_{j\in\mathbb{N}}$ of ρ, let us show that T has less than m entries. For every $j \in \mathbb{N}$, we define $n_j = \mid \{1 \le k \le m \mid \mathbf{q}_j(k) \in T_j\} \mid$ as the number of agents in the accumulator at index j. By definition of the projection, every edge (s,t) in G_j corresponds to the move of at least one agent from state s in $\mathbf{q}_j$ to state t in $\mathbf{q}_{j+1}$. Thus, since the accumulator is successor-closed, the sequence $(n_j)_{j\in\mathbb{N}}$ is non-decreasing and it increases at each entry time. The number of entry times is thus bounded by m the number of agents.

Conversely, assume that a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \cdots$ has bounded capacity, and let m be an upper bound on the number of entry times of its accumulators. Let us show that ρ is the projection of a play $\pi = \mathbf{q}_0 a_1 \mathbf{q}_1 a_2 \mathbf{q}_2 \cdots$ in the $(|S_0||Q|^{m+1})$-population game. In the initial configuration $\mathbf{q}_0$, every state in S_0 contains $|Q|^{m+1}$ agents. Then, configuration $\mathbf{q}_{n+1}$ is obtained from $\mathbf{q}_n$ by spreading the agents evenly among all edges of G_{n+1}. As a consequence, for every edge $(s,t) \in G_{n+1}$ at least a fraction $\frac{1}{|Q|}$ of the agents in state s in $\mathbf{q}_n$ moves to state t in $\mathbf{q}_{n+1}$. By induction, $\pi = \mathbf{q}_0 a_1 \mathbf{q}_1 a_2 \mathbf{q}_2 \cdots$ projects to some play $\rho' = S'_0 \xrightarrow{a_1,G'_1} S'_1 \xrightarrow{a_2,G'_2} \cdots$ such that for every $n \in \mathbb{N}$, $S'_n \subseteq S_n$ and $G'_n \subseteq G_n$. To prove that $\rho' = \rho$, we show that for every $n \in \mathbb{N}$ and state $t \in S_n$, at least $|Q|$ agents are in state t in $\mathbf{q}_n$. For that let $(U_j)_{j\in\{0\dots n\}}$ be the sequence of subsets of Q defined by $U_n = \{t\}$, and for $0 < j < n$,

$$U_{j-1} = \{s \in Q \mid \exists t' \in U_j, (s,t') \in G_j\} \ .$$

In particular, $U_0 = S_0$. Let $(T_j)_{j\in\mathbb{N}}$ be the sequence of subsets of states defined by $T_j = S_j \setminus U_j$ if $j \le n$ and $T_j = S_j$ otherwise. Then $(T_j)_{j\in\mathbb{N}}$ is an accumulator: if $s \notin U_j$ and $(s,s') \in G_j$ then $s' \notin U_{j+1}$. As a consequence, $(T_j)_{j\in\mathbb{N}}$ has at most m entry times, thus, there are at most m indices $j \in \{0\dots n-1\}$ such that some agents in the states of $S_j \setminus T_j = U_j$ in configuration $\mathbf{q}_j$ may move to states of T_{j+1} in configuration $\mathbf{q}_{j+1}$. In other words, if we denote M_j the number of agents in the states of U_j in configuration $\mathbf{q}_j$ then there are at most m indices where the sequence $(M_j)_{j\in\{0\dots n\}}$ decreases. By definition of π, even when $M_j > M_{j+1}$, at least a fraction $\frac{1}{|Q|}$ of the agents moves from U_j to U_{j+1} along the edges of G_{j+1}, thus $M_{j+1} \ge \frac{M_j}{|Q|}$. Finally, the number of agents M_n in state t in $\mathbf{q}_n$ satisfies $M_n \ge \frac{|S_0||Q|^{m+1}}{|Q|^m} \ge |Q|$. Hence ρ and ρ' coincide, so that ρ is realisable. □

3.3. **The capacity game.** An idea to obtain a game on the support arena equivalent with the population control problem is to make Agents lose whenever the play is not realisable, *i.e.* whenever the play has unbounded capacity. One issue with (un)bounded capacity is however that it is not a regular property for runs. Hence, it is not easy to use it as a winning condition. On the contrary, *finite* capacity is a regular property. We thus relax (un)bounded capacity by using (in)finite capacity and define the corresponding abstraction of the population game:

Definition 3.6 (Capacity game)**.** The *capacity game* is the game played on the support arena, where Controller wins a play iff either the play reaches $\{f\}$ or the play has infinite capacity. A player *wins the capacity game* if he has a winning strategy in this game.

We show that this relaxation can be used to decide the population control problem.

Theorem 3.7. *The answer to the population control problem is positive iff Controller wins the capacity game.*

This theorem is a direct corollary of the following propositions (3.8 - 3.10):

Proposition 3.8. *Either Controller or Agents wins the capacity game, and the winner has a winning strategy with* finite memory*.*

Proof. Whether a play has infinite capacity can be verified by a non-deterministic Büchi automaton of size $2^{|Q|}$ on the alphabet of transfer graphs, which guesses an accumulator on the fly and checks that it has infinitely many entries. This Büchi automaton can be determinised into a parity automaton (*e.g.* using Safra's construction) with state space M of size $\mathcal{O}\left(2^{2^{|Q|}}\right)$. The synchronized product of this deterministic parity automaton with the support game produces a parity game which is equivalent with the capacity game, in the sense that, up to unambiguous synchronization with the deterministic automaton, plays and strategies in both games are the same and the synchronization preserves winning plays and strategies. Since parity games are determined and positional [25], either Controller or Agents has a positional winning strategy in the parity game, thus either Controller or Agents has a winning strategy with finite memory M in the capacity game. □

Proposition 3.9. *If Controller wins the capacity game, then Controller has a winning strategy in the m-population game for all m.*

Proof. Assuming that Controller wins the capacity game with a strategy σ, he can win any m-population game, $m < \infty$, with the strategy $\sigma_m = \sigma \circ \Phi_m$. The projection $\Phi_m(\pi)$ of every infinite play π respecting σ_m is realisable, thus $\Phi_m(\pi)$ has bounded, hence finite, capacity (Lemma 3.5). Moreover $\Phi_m(\pi)$ respects σ, and since σ wins the capacity game, $\Phi_m(\pi)$ reaches $\{f\}$. Thus π reaches f^m and σ_m is winning. □

We now prove the more challenging reverse implication. Recall by Proposition 3.8 that if Agents has a winning strategy in the capacity game, then he has a *finite-memory* strategy.

Proposition 3.10. *If Agents has a winning strategy in the capacity game using* finite memory *M, then he has a winning strategy in the $|Q|^{1+|\mathsf{M}|\cdot 4^{|Q|}}$-population game.*

Proof. Let τ be a winning strategy for Agents in the capacity game with finite-memory M. First we show that any play $\pi = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ compatible with τ should have capacity (*i.e.* count of entry times of any of its accumulator) bounded by $B = |\mathsf{M}| \times 4^{|Q|}$.

Let $\{T_i\}_{i\in\mathbb{N}}$ be any accumulator of π. If there are two integers $0 \leq i < j \leq n$ such that at times i and j:

- the memory state of τ coincide: $\mathsf{m}_i = \mathsf{m}_j$;
- the supports coincide: $S_i = S_j$; and
- the supports in the accumulator T coincide: $T_i = T_j$.

then we show that there is no entry in the accumulator between indices i and j. The play π_* identical to π up to date i and which repeats ad infinitum the subplay of π between times i and j, is consistent with τ, because $\mathsf{m}_i = \mathsf{m}_j$ and $S_i = S_j$. The corresponding sequence of transfer graphs is $G_0, \ldots, G_{i-1}(G_i, \ldots, G_{j-1})^\omega$, and $T_0, \ldots, T_{i-1}(T_i \ldots T_{j-1})^\omega$ is a "periodic" accumulator of π_*. By periodicity, this accumulator has either no entry or infinitely many entries after date $i-1$. Since τ is winning, π_* has finite capacity, thus the periodic accumulator has no entry after date $i-1$, and hence there are no entries in the accumulator $(T_j)_{j\in\mathbb{N}}$ between indices i and j.

Let I be the set of entry times for the accumulator $(T_j)_{j\in\mathbb{N}}$. According to the above, for all pairs of distinct indices (i,j) in I, we have $m_i \neq m_j \vee S_i \neq S_j \vee T_i \neq T_j$. As a consequence,

$$|I| \leq B = |\mathsf{M}| \cdot 4^{|Q|} \ .$$

Now following the proof of Lemma 3.5, for $m = |Q|^{B+1}$, Agents has a strategy τ_m in the m-population game of following the transfer graphs suggested by τ. In other words, when it is Agents's turn to play in the m-population game, and the play so far $\pi = \mathbf{q}_0 \xrightarrow{a_1} \mathbf{q}_1 \cdots \mathbf{q}_n \xrightarrow{a_{n+1}}$ is projected via Φ_m to a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \cdots S_n \xrightarrow{a_{n+1}}$ in the capacity game, let $G_{n+1} = \tau(\rho)$ be the decision of Agents at this point in the capacity game. Then, to determine $\mathbf{q}_{n+1}$, τ_m splits evenly the agents in $\mathbf{q}_n$ along every edge of G_{n+1}. Since the capacity of ρ is bounded by B, the argument in the proof of Lemma 3.5 shows that $\mathbf{q}_n$ has at least $|Q|$ agents in each state and thus $\{(\mathbf{q}_n[i], \mathbf{q}_{n+1}[i]) \mid 1 \leq i \leq m\} = G_{n+1}$. This means that the projected play $\Phi_m(\pi\mathbf{q}_{n+1})$ continues to be consistent with τ and its support will never reach $\{f\}$. Thus τ_m guarantees that not all agents will be in target state m simultaneously, and hence is a winning strategy for Agents in the m-population game. □

As consequence of Proposition 3.8, the population control problem can be decided by explicitly computing the parity game and solving it, in 2EXPTIME. In the next section we will improve this complexity bound to EXPTIME.

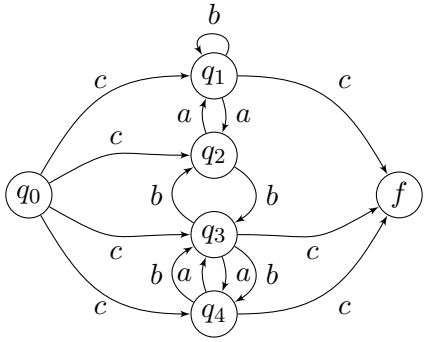


Figure 5: NFA where Controller needs memory to win the associated capacity game.

We conclude with an example showing that, in general, positional strategies are not sufficient to win the capacity game. Consider the example of Figure 5, where the only way for Controller to win is to reach a support without q_2 and play c. With a memoryless strategy, Controller cannot win the capacity game. There are only two memoryless strategies from support $S = \{q_1, q_2, q_3, q_4\}$. If Controller only plays a from S, the support remains S and the play has bounded capacity. If he only plays b's from S, then Agents can split agents from q_3 to both q_2, q_4 and the play remains in support S, with bounded capacity. In both cases, the play has finite capacity and Controller loses.

However, Controller can win the capacity game. His (finite-memory) winning strategy σ consists in first playing c, and then playing alternatively a and b, until the support does not contain $\{q_2\}$, in which case he plays c to win. Two consecutive steps ab send q_2 to q_1, q_1 to q_3, q_3 to q_3, and q_4 to either q_4 or q_2. To prevent Controller from playing c and win, Agents needs to spread from q_4 to both q_4 and q_2 every time ab is played. Consider the accumulator T defined by $T_{2i} = \{q_1, q_2, q_3\}$ and $T_{2i-1} = \{q_1, q_2, q_4\}$ for every $i > 0$. It has an infinite number of entries (from q_4 to T_{2i}). Hence Controller wins if this play is executed. Else, Agents eventually keeps all agents from q_4 in q_4 when ab is played, implying the next support does not contain q_2. Strategy σ is thus a winning strategy for Controller.

4. Solving the capacity game in EXPTIME

To solve efficiently the capacity game, we build an equivalent exponential size parity game with a polynomial number of parities. To do so, we enrich the support arena with a *tracking list* responsible of checking whether the play has finite capacity. The tracking list is a list of transfer graphs, which are used to detect certain patterns called *leaks*.

4.1. Leaking graphs.

In order to detect whether a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ has finite capacity, it is enough to detect *leaking* graphs (characterising entries of accumulators). Further, leaking graphs have special *separation* properties which will allow us to track a small number of graphs. For G, H two graphs, we denote $(a, b) \in G \cdot H$ iff there exists z with $(a, z) \in G$, and $(z, b) \in H$.

Definition 4.1 (Leaks and separations)**.** Let G, H be two transfer graphs. We say that G *leaks at* H if there exist states q, x, y with $(q, y) \in G \cdot H$, $(x, y) \in H$ and $(q, x) \notin G$. We say that G *separates* a pair of states (r, t) if there exists $q \in Q$ with $(q, r) \in G$ and $(q, t) \notin G$. Denote by $\mathsf{Sep}(G)$ the set of all pairs (r, t) which are separated by G.

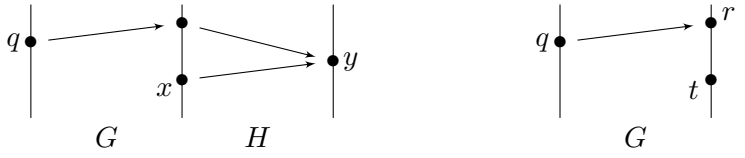
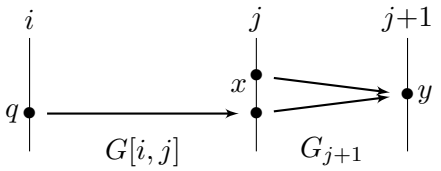


Figure 6: Left: G leaks at H; Right: G separates (r, t).

The tracking list will be composed of concatenated graphs *tracking* i of the form $G[i, j] = G_{i+1} \cdots G_j$ relating S_i with S_j: $(s_i, s_j) \in G[i, j]$ if there exists $(s_k)_{i<k<j}$ with $(s_k, s_{k+1}) \in G_{k+1}$ for all $i \leq k < j$. Infinite capacity relates to leaks in the following way:

Lemma 4.2. *A play has infinite capacity iff there exists an index i such that $G[i,j]$ leaks at G_{j+1} for infinitely many indices j.*

Proof. To prove the right-to-left implication, assume that there exists an index i such that $G[i,j]$ leaks at G_{j+1} for an infinite number of indices j. As the number of states is finite, there exists a state q with an infinite number of indices j such that we have some $(x_j, y_{j+1}) \in G_{j+1}$ with $(q, y_{j+1}) \in G[i, j+1]$, $(q, x_j) \notin G[i,j]$. The accumulator generated by $T_i = \{q\}$ has an infinite number of entries, and we are done with this direction.



For the left-to-right implication, assume that there is an accumulator $(T_j)_{j\geq 0}$ with an infinite number of entries.
For $X = (X_n)_{n\in\mathbb{N}}$ an accumulator, we denote $|X_n|$ the number of states in X_n, and we define the *width* of X as $\text{width}(X) = \limsup_n |X_n|$. We first prove the following property:

(†) If $\emptyset \neq Y \subseteq X$ and $Z \subseteq X$ are two disjoint accumulators, then $\text{width}(Z) < \text{width}(X)$.

Let us prove property (†). Let r be the minimal index s.t. $Y_r \neq \emptyset$. Thus, for every $n \geq r$, Y_n contains at least one vertex. Because Y and Z are disjoint, we derive $|Z_n| + 1 \leq |X_n|$. Taking the limsup of this inequality we obtain (†).

We pick X an accumulator of minimal width with infinitely many incoming edges. Let r minimal such that $X_r \neq \emptyset$. Let $v \in X_r$. We denote Y^v the smallest accumulator containing v. We have $Y^v \neq \emptyset$. Let us show that Y^v has infinitely many incoming edges. Define $Y^v \subseteq T^v = (T^v_n)_{n\in\mathbb{N}}$ the set of predecessors of vertices in Y^v. We let $Z_n = X_n \setminus T^v_n$ for all n. We have $Z = (Z_n)$ is an accumulator, because T^v is predecessor-closed and X is successor-closed. Applying property (†) to $0 \neq Y^v \subseteq X$ and $Z \subseteq X$, we obtain $\text{width}(Z) < \text{width}(X)$. By width minimality of X among successor-closed sets with infinitely many incoming edges, $Z = X \setminus T^v$ must have finitely many incoming edges only. Since X has infinitely many incoming edges, then T^v has infinitely many incoming edges. Thus there are infinitely many edges connecting a vertex outside Y^v to a vertex of Y^v, so that Y^v has infinitely many incoming edges. We have just shown that $G[r,j]$ leaks at infinitely many indices j. □

Indices i such that $G[i,j]$ leaks at G_{j+1} for infinitely many indices j are said to *leak infinitely often*. Note that if G separates (r,t), and r, t have a common successor in H, then G leaks at H. To link leaks with separations, we consider for each index k, the pairs of states that have a common successor, in possibly several steps, as expressed by the symmetric relation R_k: $(r,t) \in R_k$ iff there exists $j \geq k+1$ and $y \in Q$ such that $(r,y) \in G[k,j]$ and $(t,y) \in G[k,j]$.

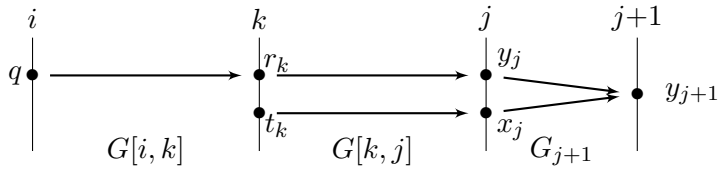
Lemma 4.3. *For $i < n$ two indices, the following properties hold:*

(1) *If $G[i,n]$ separates $(r,t) \in R_n$, then there exists $m \geq n$ such that $G[i,m]$ leaks at G_{m+1}.*
(2) *If index i does not leak infinitely often, then the number of indices j such that $G[i,j]$ separates some $(r,t) \in R_j$ is finite.*
(3) *If index i leaks infinitely often, then for all $j > i$, $G[i,j]$ separates some $(r,t) \in R_j$.*
(4) *If $i < j < n$ then $\mathsf{Sep}(G[i,n]) \subseteq \mathsf{Sep}(G[j,n])$.*

Proof. We start with the proof of the first item. Assume that $G[i,n]$ separates a pair $(r,t) \in R_n$. Hence there exists q such that $(q,r) \in G[i,n]$, $(q,t) \notin G[i,n]$. Since $(r,t) \in R_n$, there is an index $k > n$ and a state y such that $(r,y) \in G[n,k]$ and $(t,y) \in G[n,k]$. Hence, there exists a path $(t_j)_{n \leq j \leq k}$ with $t_n = t$, $t_k = y$, and $(t_j, t_{j+1}) \in G_{j+1}$ for all $n \leq j < k$. Moreover, there is a path from q to y because there are paths from q to r and from r to y. Let $\ell \leq k$ be the minimum index such that there is a path from q to t_ℓ. As there is no path from q to $t_n = t$, necessarily $\ell \geq n+1$. Obviously, $(t_{\ell-1}, t_\ell) \in G_\ell$, and by definition and minimality of ℓ, $(q, t_{\ell-1}) \notin G[i, \ell-1]$ and $(q,t_\ell) \in G[i,\ell]$. That is, $G[i,\ell-1]$ leaks at G_ℓ.

Let us now prove the second item, using the first one. Assume that i does not leak infinitely often, and towards a contradiction suppose that there are infinitely many j's such that $G[i,j]$ separates some $(r,t) \in R_j$. To each of these separations, we can apply item ***1.*** to obtain infinitely many indices m such that $G[i,m]$ leaks at G_{m+1}, a contradiction.

We now prove the third item. Since there are finitely many states in Q, there exists $q \in Q$ and an infinite set J of indices such that for every $j \in J$, $(q, y_{j+1}) \in G[i, j+1]$, $(q, x_j) \notin G[i,j]$, and $(x_j, y_{j+1}) \in G_{j+1}$ for some x_j, y_{j+1}. The path from q to y_{j+1} implies the existence of y_j with $(q,y_j) \in G[i,j]$, and $(y_j, y_{j+1}) \in G_{j+1}$. We have thus found separated pairs $(x_j, y_j) \in R_j$ for every $j \in J$. To exhibit separations at other indices $k > j$ with $k \notin J$, the natural idea is to consider predecessors of the x_j's and y_j's.



We define sequences $(r_k, t_k)_{k \geq i}$ inductively as follows. To define r_k, we take a $j \geq k+1$ such that $j \in J$; this is always possible as J is infinite. There exists a state r_k such that $(q, r_k) \in G[i,k]$ and $(r_k, y_j) \in G[k,j]$.
Also, as x_j belongs to $\mathrm{Im}(G[1,j])$, there must exist a state t_k such that $(t_k, x_j) \in G[k,j]$. Clearly, $(q,t_k) \notin G[i,k]$, else $(q,x_j) \in G[i,j]$, which is not true. Last, y_{j+1} is a common successor of t_k and r_k, that is $(t_k, y_{j+1}) \in G[k, j+1]$ and $(r_k, y_{j+1}) \in G[k, j+1]$. Hence $G[i,k]$ separates $(r_k, t_k) \in R_k$.

For the last item, let $(r,t) \in \mathsf{Sep}(G[i,n])$ and pick $q \in Q$ such that $(q,r) \in G[i,n]$ but $(q,t) \notin G[i,n]$. Since $(q,r) \in G[i,n]$ there exists $q' \in Q$ so that $(q,q') \in G[i,j]$ and $(q',r) \in G[j,n]$. It also follows that $(q',t) \notin G[j,n]$ since $(q,q') \in G[i,j]$ but $(q,t) \notin G[i,n]$. Thus we have shown $(r,t) \in \mathsf{Sep}(G[j,n])$. □

4.2. **The tracking list.** Given a fixed index i, figuring out whether i is leaking or not can be done using a deterministic automaton. However, when one wants to decide the existence of *some* index that leaks, naively, one would have to keep track of runs starting from all possible indices $i \in \mathbb{N}$. The tracking list will allow us to track only quadratically many indices at once. The *tracking list* exploits the relationship between leaks and separations. It is a list of transfer graphs which altogether separate all possible pairs of states[1], and are sufficient to detect when leaks occur.

[1]It is sufficient to consider pairs in R_j. However, as R_j is not known *a priori*, we consider all pairs in Q^2.

By item (4) in Lemma 4.2, for any n, $\mathsf{Sep}(G[1,n]) \subseteq \mathsf{Sep}(G[2,n]) \subseteq \cdots \mathsf{Sep}(G[n,n])$. The *exact* tracking list $\mathcal{L}_n$ at step n is defined as a list of $k \leq |Q|^2$ graphs $G[i_1,n], \cdots, G[i_k,n]$, where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ is the list of indices for which $\mathsf{Sep}(G[i-1,n]) \neq \mathsf{Sep}(G[i,n])$ (with the convention that $\mathsf{Sep}(G[0,n]) = \emptyset$).

Consider the sequence of graphs in Figure 7, obtained from alternating `try` and `retry` in the example from Figure 2 where Agents splits agents whenever possible.

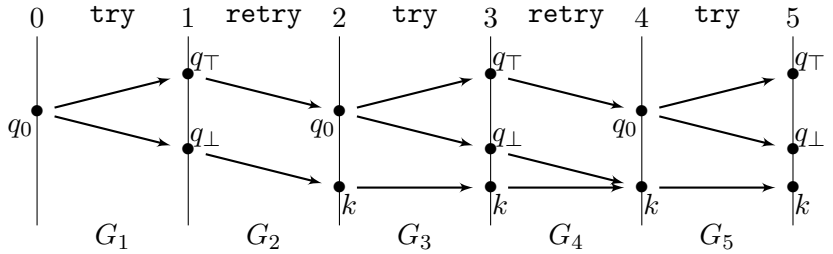


Figure 7: Sequence of graphs associated with a run.

Let us compute $\mathsf{Sep}(G[i,5])$ on that example for $0 \leq i \leq 4$. The graph $G[0,5] = G_1 \cdots G_5$ has the following edges: $(q_0, q_\top), (q_0, q_\bot), (q_0, k)$, and thus $\mathsf{Sep}(G[0,5]) = \emptyset$. In comparison, $\mathsf{Sep}(G[1,5]) = \{(k, q_\top), (k, q_\bot)\}$ because $(q_\bot, k)$ is an edge of $G[1,5]$ and $(q_\bot, q_\top)$ and $(q_\bot, q_\bot)$ are not. Also, $\mathsf{Sep}(G[2,5]) = \{(k, q_\top), (k, q_\bot)\}$. Finally $\mathsf{Sep}(G[3,5]) = \mathsf{Sep}(G[4,5]) = \{(k, q_\top), (k, q_\bot), (q_\top, k), (q_\bot, k)\}$. Thus, $\mathcal{L}_5 = (G[1,5]; G[3,5])$. Notice that $G[1,5] = \{(q_\top, q_\top), (q_\top, q_\bot), (q_\top, k), (q_\bot, k)\}$ and $G[3,5] = \{(q_\top, q_\top), (q_\top, q_\bot), (q_\bot, k), (k, k)\}$.

The *exact* tracking list $\mathcal{L}_n$ allows one to test for infinite leaks, but computing it with polynomial memory seems hard. Instead, we propose to approximate the *exact* tracking list into a list, namely the tracking list $\overline{\mathcal{L}}_n$, which needs only polynomial memory to be computed, and which is sufficient for finding infinite leaks.

The tracking list $\overline{\mathcal{L}}_n$ is also of the form $\{G[i_1,n], G[i_2,n], \ldots G[i_k,n]\}$ where $0 \leq i_1 < i_2 \ldots i_k < n$ with $\emptyset \neq \mathsf{Sep}(G[i_r,n]) \subsetneq \mathsf{Sep}(G[i_{r+1},n])$. It is is computed inductively in the following way: $\overline{\mathcal{L}}_0$ is the empty list. For $n > 0$, the list $\overline{\mathcal{L}}_n$ is computed from $\overline{\mathcal{L}}_{n-1}$ and G_n in three stages by the following update_list algorithm:

(1) First, every graph $G[i, n-1]$ in the list $\overline{\mathcal{L}}_{n-1}$ is concatenated with G_n, yielding $G[i,n]$.
(2) Second, $G_n = G[n-1,n]$ is added at the end of the list.
(3) Lastly, the list is filtered: a graph H is kept if and only if it separates a pair of states $(p,q) \in Q^2$ which is not separated by any graph that appears earlier in the list.[2]

Under this definition, $\overline{\mathcal{L}}_n = \{G[i_j,n] \mid 1 \leq j \leq k, \mathsf{Sep}(G[i_{j-1},n]) \neq \mathsf{Sep}(G[i_j,n])\}$, with the convention that $\mathsf{Sep}(G[i_0,n]) = \emptyset$.

Notice that the tracking list $\overline{\mathcal{L}}_n$ may differ from the exact tracking list $\mathcal{L}_n$, as shown with the example on Figure 8. We have $\mathcal{L}_3 = (G[1,3], G[2,3])$, as $G[0,3] = G_1 \cdots G_3$ does not separate any pair of states, $G[1,3] = G_2 \cdot G_3$ separates (q_1, q_2) and $G[2,3] = G_3$ separates (q_1, q_2) and (q_2, q_1). However, $\overline{\mathcal{L}}_3 = (G[2,3]) \neq \mathcal{L}_3$. Indeed, $\mathcal{L}_2 = \overline{\mathcal{L}}_2 = (G[0,2])$ as $G[0,2] = G_1 \cdot G_2$ and $G[1,2] = G_2$ separate exactly the same pairs $(q_1,q_2); (q_1,q_3); (q_2,q_3); (q_3,q_2)$. Applying the update_list algorithm, we obtain the intermediate list (G[0,3],G[2,3]) after

[2]This algorithm can be performed without knowing the indices $(i_j)_{j \leq k}$, but just the graphs $(G[i_j,n])_{j \leq k}$.

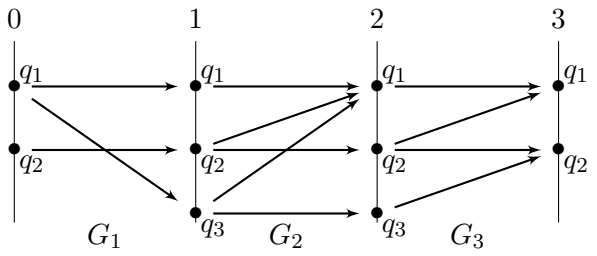


Figure 8: Example where the tracking list $\overline{\mathcal{L}}_3$ differs from the exact tracking list $\mathcal{L}_3$.

stage 2. As $G[0,3]$ separates no pair of states, it is filtered out in stage 3. We obtain $\overline{\mathcal{L}}_3 = (G[2,3]) \neq \mathcal{L}_3$.

Let $\overline{\mathcal{L}}_n = \{H_1, \cdots, H_\ell\}$ be the tracking list at step n. Each transfer graph $H_r \in \overline{\mathcal{L}}_n$ is of the form $H_r = G[t_r, n]$. We say that r is the *level* of H_r, and t_r the *index tracked* by H_r. Observe that the lower the level of a graph in the list, the smaller the index it tracks.

When we consider the sequence of tracking lists $(\overline{\mathcal{L}}_n)_{n\in\mathbb{N}}$, for every index i, either it eventually stops to be tracked or it is tracked forever from step i, *i.e.* for every $n \geq i$, $G[i,n]$ is not filtered out from $\overline{\mathcal{L}}_n$. In the latter case, i is said to be *remanent* (it will never disappear).

Lemma 4.4. *A play has infinite capacity iff there exists an index i such that i is remanent and leaks infinitely often.*

Proof. Because of Lemma 4.2 we only need to show that if there is an index i that leaks infinitely often, then there is an index which is remanent and leaks infinitely often.

Let i be the smallest index that leaks infinitely often. By Lemma 4.3 (2) there is an $N > i$ so that whenever $j < i < N \leq n$, $\mathsf{Sep}(G[j,n]) \cap R_n = \emptyset$. Similarly, by Lemma 4.3 (3), for every $n > i$, $\mathsf{Sep}(G[i,n]) \cap R_n \neq \emptyset$. Combined with Lemma 4.3 (4), this implies that there is some index $j^* \geq i$ which is remanent. Indeed, let $\overline{\mathcal{L}}_N = \{G[i_1, N], G[i_2, N] \dots G[i_k, N]\}$ and let $j^* = \min\{i_r \mid i_r \geq i\}$. Index j^* exists because for any $i_r < i$, $\mathsf{Sep}(G[i_r, N]) \subsetneq \mathsf{Sep}(G[i, N]) \subseteq \mathsf{Sep}(G[N-1, N])$. The strict inequality arises because $\mathsf{Sep}(G[i_r, N]) \cap R_N = \emptyset$ but $\mathsf{Sep}(G[i, N]) \cap R_N \neq \emptyset$. By the same argument, for every $n \geq N$ and $i_r < i$, $\mathsf{Sep}(G[i_r, n]) \cap R_n = \emptyset$ and $\mathsf{Sep}(G[j^*, n]) \cap R_n \supseteq \mathsf{Sep}(G[i, n]) \cap R_n \neq \emptyset$. This shows that j^* is remanent. By Lemma 4.3 (2), index j^* also leaks infinitely often. □

4.3. **The parity game.** We now describe a parity game $\mathcal{PG}$, which extends the support arena with on-the-fly computation of the tracking list.

Priorities. By convention, lowest priorities are the most important and the odd parity is good for Controller, so Controller wins iff the lim inf of the priorities is odd. With each level $1 \leq r \leq |Q|^2$ of the tracking list are associated two priorities $2r$ (graph $G[i_r, n]$ non-remanent) and $2r+1$ (graph $G[i_r, n]$ leaking), and on top of that are added priorities 1 (goal reached) and $2|Q|^2 + 2$ (nothing), hence the set of all priorities is $\{1, \dots, 2|Q|^2 + 2\}$.

When Agents chooses a transition labelled by a transfer graph G, the tracking list is updated with G and the priority of the transition is determined as the smallest among: priority 1 if the support $\{f\}$ has ever been visited, priority $2r+1$ for the smallest r such

that H_r (from level r) leaks at G, priority $2r$ for the smallest level r where graph H_r was removed from $\mathcal{L}$, and in all other cases priority $2|Q|^2+2$.

States and transitions. $\mathcal{G}^{\leq |Q|^2}$ denotes the set of list of at most $|Q|^2$ transfer graphs.

- States of $\mathcal{PG}$ form a subset of $\{0,1\} \times 2^Q \times \mathcal{G}^{\leq |Q|^2}$, each state being of the form $(b, S, H_1, \ldots, H_\ell)$ with $b \in \{0,1\}$ a bit indicating whether the support $\{f\}$ has been seen, S the current support and $(H_1, \ldots, H_\ell)$ the tracking list. The initial state is $(0, \{q_0\}, \varepsilon)$, with ε the empty list.
- Transitions in $\mathcal{PG}$ are all $(b, S, H_1, \ldots, H_\ell) \xrightarrow{\mathbf{p},a,G} (b', S', H'_1, \ldots, H'_{\ell'})$ where $\mathbf{p}$ is the priority, and such that $S \xrightarrow{a,G} S'$ is a transition of the support arena, and
  (1) $(H'_1, \ldots, H'_{\ell'})$ is the tracking list obtained by updating the tracking list $(H_1, \ldots, H_\ell)$ with G, as explained in subsection 4.2;
  (2) if $b = 1$ or if $S' = \{f\}$, then $\mathbf{p} = 1$ and $b' = 1$;
  (3) otherwise $b' = 0$. In order to compute the priority $\mathbf{p}$, we let $\mathbf{p}'$ be the smallest level $1 \leq r \leq \ell$ such that H_r leaks at G and $\mathbf{p}' = \ell + 1$ if there is no such level, and we also let $\mathbf{p}''$ as the minimal level $1 \leq r \leq \ell$ such that $H'_r \neq H_r \cdot G$ and $\mathbf{p}'' = \ell + 1$ if there is no such level. Then $\mathbf{p} = \min(2\mathbf{p}' + 1, 2\mathbf{p}'')$.

We are ready to state the main result of this paper, which yields an EXPTIME complexity for the population control problem. This entails the first statement of Theorem 2.5, and together with Proposition 3.10, also the first statement of Theorem 2.7.

Theorem 4.5. *Controller wins the game $\mathcal{PG}$ if and only if Controller wins the capacity game. Solving these games can be done in time $O(2^{(1+|Q|+|Q|^4)(2|Q|^2+2)})$. Strategies with $2^{|Q|^4}$ memory states are sufficient to both Controller and Agents.*

Proof. The state space of parity game $\mathcal{PG}$ is the product of the set of supports with a deterministic automaton computing the tracking list. There is a natural correspondence between plays and strategies in the parity game $\mathcal{PG}$ and in the capacity game.

Controller can win the parity game $\mathcal{PG}$ in two ways: either the play visits the support $\{f\}$, or the priority of the play is $2r+1$ for some level $1 \leq r \leq |Q|^2$. By design of $\mathcal{PG}$, this second possibility occurs iff r is remanent and leaks infinitely often. According to Lemma 4.4, this occurs if and only if the corresponding play of the capacity game has infinite capacity. Thus Controller wins $\mathcal{PG}$ iff he wins the capacity game.

In the parity game $\mathcal{PG}$, there are at most $2^{1+|Q|}\left(2^{|Q|^2}\right)^{|Q|^2} = 2^{1+|Q|+|Q|^4}$ states and $2|Q|^2+2$ priorities, implying the complexity bound using state-of-the-art algorithms [18]. Actually the complexity is even quasi-polynomial according to the algorithms in [10]. Notice however that this has little impact on the complexity of the population control problem, as the number of priorities is logarithmic in the number of states of our parity game.

Further, it is well known that the winner of a parity game has a positional winning strategy [18]. A *positional* winning strategy σ in the game $\mathcal{PG}$ corresponds to a *finite-memory* winning strategy σ' in the capacity game, whose memory states are the states of $\mathcal{PG}$. Actually in order to play σ', it is enough to remember the tracking list, *i.e.* the third component of the state space of $\mathcal{PG}$. Indeed, the second component, in 2^Q, is redundant with the actual state of the capacity game and the bit in the first component is set to 1 when the play visits $\{f\}$ but in this case the capacity game is won by Controller whatever is played afterwards. Since there are at most $2^{|Q|^4}$ different tracking lists, we get the upper bound on the memory. □

5. Number of steps before Synchronization

In this section, we will restrict ourselves to *controllable NFAs*, that is positive instances of the population control problem. To be useful in the presence of many agents, the controller should be able to gather all agents in the target state in a reasonable time (counted as the number of actions played before synchronization). Notice that this notion is similar to the termination time used for population protocols in [8].

5.1. **Dependency with respect to the number of agents.** We first show that there are controllable NFAs for which Controller requires a quadratic number of steps (in the number of agents) against the best strategy of Agents. Consider again the NFA $\mathcal{A}_{\text{time}}$ from Figure 2 (see also Figure 9, left). Recall that $\mathcal{A}_{\text{time}}$ is controllable.

Lemma 5.1. *For the NFA $\mathcal{A}_{time}$, Controller requires $\Theta(m^2)$ steps to win in the worst case.*

Proof. A winning strategy σ for Controller is to play `try` followed by `keep` until only one of $q_\top$ or $q_\bot$ is filled, in which case action `top` or `bottom` can be played to move the associated agents to the target state.This will eventually happen, as the number of agents in $q_\top$ and in $q_\bot$ is decreasing while the number of agents in state k increases upon (`try`;`keep`) and when $q_\bot$ is not empty. This is the strategy generated from our algorithm.

We now argue on the number of steps σ needed to send all agents to the target state. Observe that the only non-deterministic action is `try` from state q_0. Clearly enough, regarding the number of steps before synchronisation, Agents' best answer is to move one agent in $q_\bot$ and the remaining agents in $q_\top$ upon each `try` action. Letting m be the number of agents, the run associated with σ and the best counterstrategy for Agents is

$$(\texttt{try};\texttt{keep})^{m-1};\texttt{bot};\texttt{restart};(\texttt{try};\texttt{keep})^{m-2};\texttt{bot};\texttt{restart}\cdots\texttt{try};\texttt{keep};\texttt{bot}$$

and its size is $\sum_{i=1}^{m-1}(2i+2)-1 = O(m^2)$. The system thus requires a quadratic number of steps before synchronisation, and this is the worst case. □

Notice that the above result only needs a fixed number of states, namely 6.

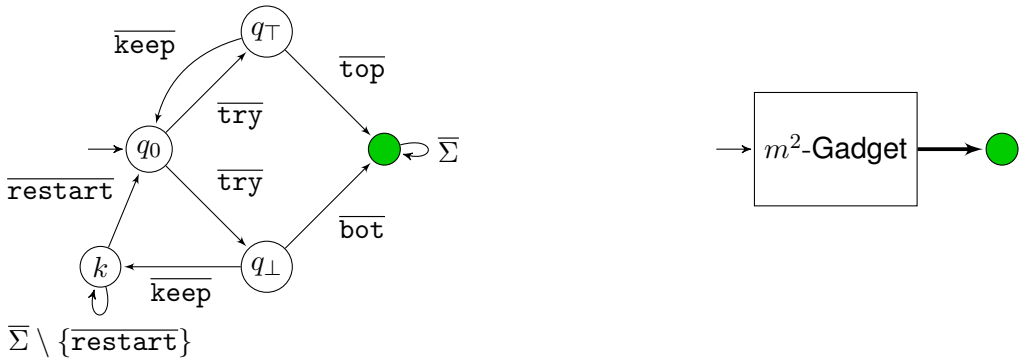


Figure 9: NFA $\overline{\mathcal{A}_{\text{time}}}$ on alphabet $\overline{\Sigma}$ and its abstraction into the gadget m^2-Gadget.

5.2. **Polynomial bound on the number of steps for synchronization.** We now show that we can build an NFA with n states such that the system requires order $m^{0(n)}$ steps before synchronisation. For that, we turn the system $\mathcal{A}_{\text{time}}$ into a gadget, as shown on Figure 9. This gadget will be used in an inductive manner to obtain an NFA for which $m^{0(n)}$ steps before synchronisation are required against the best strategy of Agents.

First, we construct an NFA which requires $O(m^4)$ steps before synchronisation. Essentially we replace the edge from q_0 to $q_\top$ in the NFA $\mathcal{A}_{\text{time}}$ by the m^2-Gadget to obtain the NFA $\mathcal{A}_4$ on Figure 10. The alphabet $\overline{\Sigma}$ of actions of the m^2-Gadget is a disjoint copy of the alphabet Σ of actions of $\mathcal{A}_{\text{time}}$. In particular, playing any action of Σ when any token is in the m^2-Gadget leads to the losing sink state ☹.

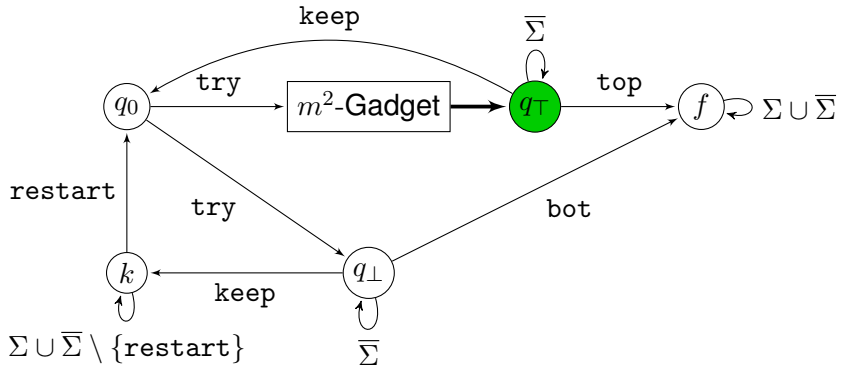


Figure 10: The NFA $\mathcal{A}_4$ which requires $\Theta(m^4)$ steps to synchronize all agents into the final state. State $q_\top$ is the output state of the m^2-Gadget.

Consider the strategy of Agents which is to place 1 agent in $q_\perp$ (resp. $\overline{q_\perp}$) and the rest to $q_\top$ (resp. $\overline{q_\top}$) when action `try` (resp. $\overline{\texttt{try}}$) is played. Relying on Lemma 5.1, any strategy of Controller needs m^2 steps to place 1 agent in k, then $(m-1)^2$ steps for the next agent, etc. Thus, any strategy needs $O(m^3)$ steps to place 1 agent in the target state f. Finally, any strategy needs $O(m^4)$ steps to place all the agents in the target state f. We thus obtain:

Lemma 5.2. *For the NFA $\mathcal{A}_4$, Controller requires $\Theta(m^4)$ steps to win in the worst case.*

One can repeat this construction, nesting copies of the m^2-Gadget. At each new gadget, the number of states in the NFA increases by a constant amount, namely 4. The ℓ-layered NFA, consisting of $\ell - 1$ nested gadgets, has $4\ell + 2$ states and requires $\Theta(m^{2\ell})$ steps before synchronisation. We thus derive the following upper-bound on the time to synchronisation:

Corollary 5.3. *There exist NFAs with $|Q|$ states such that $m^{\frac{|Q|-2}{2}}$ steps are required by any strategy to synchronise m agents.*

5.3. **Optimality of the winning strategy.** In this subsection, we show that the winning strategy built by our algorithm is optimal, in the sense that it never requires more than $m^{|Q|^{O(1)}}$ steps to synchronize m agents. For $\mathcal{A}$ a controllable NFA, we write $\sigma_{\mathcal{A}}$ the winning strategy built by our algorithm.

Theorem 5.4. *For $\mathcal{A}$ with $|Q|$ states, $\sigma_{\mathcal{A}}$ needs at most $m^{|Q|} \times 2^{|Q|^4}$ steps to synchronise m agents in the target state f.*

To prove Theorem 5.4, an essential ingredient is the following:

Lemma 5.5. *Let* m *be a memory state of* $\sigma_{\mathcal{A}}$*,* $S \neq \{f\}$ *a support and* H *a transfer graph such that (i)* (m, S) *is reachable under* $\sigma_{\mathcal{A}}$*, (ii)* H *is compatible with* $\sigma_{\mathcal{A}}$ *from* (m, S)*, and (iii)* H *is a loop around* (m, S)*. Then, there exists a partition* $S = T \uplus U$ *such that* $H(U) \subseteq U$ *and* $H(T) \cap U \neq \emptyset$*, where* $H(X) = \{q \in S \mid \exists p \in X,\ (p, q) \in H\}$*.*

Intuitively, letting G a run according to $\sigma_{\mathcal{A}}$ reaching (m, S), the run GH^ω is also according to $\sigma_{\mathcal{A}}$. As $\sigma_{\mathcal{A}}$ is winning, GH^ω must be non realisable, that is GH^ω needs to have an accumulator with an infinite number of entries. We prove in this lemma that for a repeated graph H, there is a structural characterisation of accumulators with an infinite number of entries: S can be partitioned into $U \uplus T$, where U corresponds to a structural accumulator, and there is one entry from T to U for each H.

Proof. As Controller is winning, the play GH^ω must have infinite capacity. By Lemma 4.2, there exist an index i and an infinite sequence of indices $j > i$ such that H^{j-i} leaks at H. Because the same graph H is repeated, we can assume wlog that $i = 0$. As the number of states is finite, there exists a triple (q, x, y) such that $(q, y) \in H^{j+1}$, $(x, y) \in H$ and $(q, x) \notin H^j$ for all j's in an infinite set J of indices.

Let ℓ be the number of steps in G. We consider H atomic, that is, as if it is a single step. For every state $s \in S$, define $A^s = (A^s_n)_{n\in\mathbb{N}}$ the smallest accumulator with $A^s_\ell = \{s\}$.

If any state $s \in S$ is such that $s \notin A^s_n$ for any $n > \ell$, then we are done as we can set $U = \bigcup_{n>\ell} A^s_n$, and $T = S \setminus U$. Indeed, $s \notin U$, and s has a successor in U (any state in $A^s_{\ell+1}$). We can thus assume that $s \in \bigcup_{n>\ell} A^s_n$ for all $s \in S$.

Let $U = \bigcup_{n>\ell} A^y_n = \bigcup_{n\geq\ell} A^y_n$ as $y \in \bigcup_{n>\ell} A^y_n$ and $A^y_\ell = \{y\}$. Let $T = S \setminus U$. We will show that this choice of (T, U) satisfies the condition of the statement. In particular, we will show that $x \notin U$, and as $(x, y) \in H$ and $y \in U$, we are done.

Let $k_x, k_y > 0$ such that $(x, x) \in H^{k_x}$, and $(y, y) \in H^{k_y}$. Let $k = k_x \times k_y$. Partition J into sets $J_r = \{j \mid j = r(\mod k)\}$, for $r < k$. As J is infinite, one of J_r must be infinite. Taking two indices $j, j' \in J_r$, we have that $j' - j$ is a multiple of k. In particular, we have $(x, x) \in H^{j'-j}$ and $(y, y) \in H^{j'-j}$. We also know that $(y, x) \notin H^{j'-j-1}$ as $(q, x) \notin H^{j'}$ and $(q, y) \in H^{j+1}$.

For any state $s \in S$, let $width(s) = max_n(|A^s_n|)$. Easily, for all $s \in U$, $width(s) \leq width(y)$, as s can be reached by y, let say in v steps, and hence $A^s_n \subseteq A^y_{n+v}$. We now show that $width(x) > width(y)$, which implies that $x \notin U$, and we are done. For all n, we have $A^y_n \subseteq A^x_{n+1}$, as $(x, y) \in H$. Now, there are two indices $j, j' \in J_r$ such that $j' - j > n + 1$, as J_r is infinite. Let z such that $(x, z) \in H^{n+1}$ and $(z, x) \in H^{j'-j-n-1}$, which must exists as $(x, x) \in H^{j'-j}$. As $(y, x) \notin H^{j'-j-1}$, we have $(y, z) \notin H^n$. That is, $z \in A^x_{n+1} \setminus A^y_n$. Thus, $|A^x_{n+1}| \geq |A^y_n| + 1$, and thus $width(x) > width(y)$. □

We can now prove Theorem 5.4:

Proof. Assume by contradiction that there is a run consistent with $\sigma_{\mathcal{A}}$ lasting more than $m^{|Q|} \times 2^{|Q|^4}$ steps before synchronisation. Because there are no more than $2^{|Q|^4}$ different memory states, there is one memory state m which is repeated at least $m^{|Q|}$ times. Let us decompose the path as $G_1 G_2 \cdots G_{m^{|Q|}}$, such that $G_i \dots G_j$ is a loop around m, for all $1 \leq i < j \leq m^{|Q|}$. We write S for the support associated with m.

By Lemma 5.5 applied with $H = G_1 G_2 \cdots G_{m^{|Q|}}$, there exists a partition $S = T \uplus U$, such that $H(U) \subseteq U$ and $H(T) \cap U \neq \emptyset$. We define a sequence $U_1, \ldots, U_{m^{|Q|}}$ of supports inductively as follows:

- $U_{m^{|Q|}} = U$, and
- for $i = m^{|Q|}$ to $i = 1$, $U_{i-1} = \{s \in S \mid G_i(\{s\}) \subseteq U_i\}$.

We further write $T_i = S \setminus U_i$. We have $U \subseteq U_0 \subsetneq S$, and $U_i \neq \emptyset$ as $G_i(U_{i-1}) \subseteq U_i$ for all i. In the same way, $U_i \neq S$ for all i as otherwise we would have $U_{m^{|Q|}} = S$, a contradiction with $U_{m^{|Q|}} = U \neq S$. Hence $T_i \neq \emptyset$ for all i.

Consider now the set K of indices i such that $G_i(T_{i-1}) \cap U_i \neq \emptyset$. By Lemma 5.5, K is nonempty. Moreover, since there are m agents, $|K| < m$, otherwise, T_0 would contain at least m more agents than $T_{m^{|Q|}}$. Thus $1 \leq |K| < m$. Hence, there are two indices $i < j \in K$ that are far enough, *i.e.* such that $j - i > m^{|Q|-1}$, with $k \notin K$ for all $i < k < j$. Therefore, for every k between i and j, $G_k(T_{k-1}) = T_k$ and $G_k(U_{k-1}) = U_k$: no agent is transferred from $(T_k)_{i \leq k \leq j}$ to $(U_k)_{i \leq k \leq j}$ in the fragment $G_i \cdots G_j$. We say that $(T_k)_{i \leq k \leq j}$ and $(U_k)_{i \leq k \leq j}$ do not communicate. In particular, we have 2 non-empty disjoint subsets T_i, U_i of Q. We will inductively partition at each step at least one of the sequences into 2 non-communicating subsequences. Eventually, we obtain $|Q| + 1$ non-communicating subsequences, and in particular $|Q| + 1$ non-empty disjoint subsets of Q, a contradiction.

Applying Lemma 5.5 again on $H' = G_i \cdots G_j$ yields a partition $S' = T' \uplus U'$. We define in the same way $U'_j = U'$ and inductively U'_k for $k = j - 1, \ldots, i$. As above, there are less than m indices $i' \in [i, j]$ such that $G_{i'}(T'_{i'-1}) \cap U_{i'} \neq \emptyset$. We can thus find an interval $[i', j'] \subsetneq [i, j]$ such that $j' - i' > m^{|Q|-2}$ and $(T'_k)_{i' \leq k \leq j'}$ and $(U'_k)_{i' \leq k \leq j'}$ do not communicate.

To sum up, (any pair of) the four following sequences do not communicate together:

- $(T_k \cap T'_k)_{i' \leq k \leq j'}$, $(T_k \cap U'_k)_{i' \leq k \leq j'}$,
- $(U_k \cap T'_k)_{i' \leq k \leq j'}$ and $(U_k \cap U'_k)_{i' \leq k \leq j'}$.

For any of these four sequences $(X_k)_{i' \leq k \leq j'}$, if $X_k \neq \emptyset$ for some k, then $X_k \neq \emptyset$ for all k (as these 4 sequences do not communicate together and they partition S), in which case we say that the sequence is *non-empty*. Now some of these sequences may be empty. Yet, we argue that at least 3 of them are non-empty.

Indeed, for at least one index $i \leq k < j$, we have $G_k(T'_{k-1}) \cap U'_k \neq \emptyset$. As there is no communication between $(T_k)_{i \leq k \leq j}$ and $(U_k)_{i \leq k \leq j}$, at least one of T_k, U_k, let say T_k, contains at least one state from T'_k *and* one state from U'_k. Assuming $|j' - i'|$ maximal, we can choose $k = i'$ (else we have a contradiction with $|j' - i'|$ maximal, unless $i' = i$, in which case we can choose $k = j' + 1$). That is, $T_{i'} \cap T'_{i'} \neq \emptyset$ and $T_{i'} \cap U'_{i'} \neq \emptyset$, and both sequences are non-empty. Obviously at least one of $U_{i'} \cap T'_{i'}$ and $U_{i'} \cap U'_{i'}$ should be non-empty as $U_{i'} = (U_{i'} \cap T'_{i'}) \cup (U_{i'} \cap U'_{i'})$ is nonempty, and this gives us the third non-empty sequence. Hence, we have three non-empty sequences such that no pair of these sequences communicate between i' and j'.

We can iterate once more to obtain $i'' < j''$ with $j'' - i'' > m^{|Q|-3}$, and four non-empty sequences, such that no pair of these sequences communicate between i'' and j''. This is because 1 non-empty sequence contains states from both U'' and T'', giving 2 non-empty sequences, and the 2 other non-empty sequences gives at least 2 non-empty sequences. Obviously, this operation can be made at most $|Q|$ times, as it would result into $|Q| + 1$ non-empty and pairwise disjoint subsets of Q. We thus obtain a contradiction with the number of steps being more than $m^{|Q|} \times 2^{|Q|^4}$. □

6. Lower bounds

The proofs of Theorems 2.5 and 2.7 are concluded by the proofs of lower bounds.

Theorem 6.1. *The population control problem is* EXPTIME*-hard.*

Proof. We first prove PSPACE-hardness of the population control problem, reducing from the halting problem for polynomial space Turing machines. We then extend the result to obtain the EXPTIME-hardness, by reducing from the halting problem for polynomial space *alternating* Turing machines. Let $\mathcal{M} = (S, \Gamma, T, s_0, s_f)$ be a Turing machine with $\Gamma = \{0, 1\}$ as tape alphabet. By assumption, there exists a polynomial P such that, on initial configuration $x \in \{0,1\}^n$, $\mathcal{M}$ uses at most $P(n)$ tape cells. A transition $t \in T$ is of the form $t = (s, s', b, b', d)$, where s and s' are, respectively, the source and the target control states, b and b' are, respectively, the symbols read from and written on the tape, and $d \in \{\leftarrow, \rightarrow, -\}$ indicates the move of the tape head. From $\mathcal{M}$ and x, we build an NFA $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ with a distinguished state $\mathbf{Acc}$ such that, $\mathcal{M}$ terminates in s_f on input x if and only if $(\mathcal{A}, \mathbf{Acc})$ is a positive instance of the population control problem.

Now we describe the states of NFA $\mathcal{A}$. They are given by:

$$Q = Q_{\texttt{cells}} \cup Q_{\texttt{pos}} \cup Q_{\texttt{cont}} \cup \{q_0, \mathbf{Acc}, ☹\}$$

where

- $Q_{\texttt{cells}} = \bigcup_{i=1}^{P(n)} \{0_i, 1_i\}$ are the states for the cells contents of $\mathcal{M}$, one per bit and per position;
- $Q_{\texttt{pos}} = \{p_i \mid 1 \leq i \leq P(n)\}$ are the states for the position of tape head of $\mathcal{M}$;
- $Q_{\texttt{cont}} = S$ are the states for the control state of $\mathcal{M}$;
- q_0 is the initial state of $\mathcal{A}$, $\mathbf{Acc}$ is a sink winning state and ☹ is a sink losing state.

A configuration of the Turing machine, of the form $(q, p, x) \in S \times [P(n)] \times \{0,1\}^{P(n)}$, is represented by any configuration of $\mathcal{A}^m$ such that the set of states with at least one agent is $\{q, p\} \cup \{0_i \mid x_i = 0\} \cup \{1_i \mid x_i = 1\}$.

With each transition $t = (s, s', b, b', d)$ in the Turing machine and each position p of the tape, we associate an action $a_{t,p}$ in $\mathcal{A}$, which simulates the effect of transition t when the head position is p. For instance, Fig. 11 represents the transitions associated with action $a_{t,k}$, for the transition $t = (q_i, q_j, 0, 1, \rightarrow)$ of the Turing Machine and position k on the tape. Note that if agents are in the states representing a configuration of the Turing machine, then the only action Controller can take to avoid ☹ is to play $a_{t,p}$ where p is the current head position and t is the next allowed transition. Moreover on doing this, the next state in $\mathcal{A}^m$ exactly represents the next configuration of the Turing machine.

There are also winning actions called $\texttt{check}(Q')$ for certain subsets $Q' \subseteq Q$. Controller should only play these when no agents are in Q'. One of them is for $Q' = Q_{\texttt{cont}} \setminus \{s_f\}$ which can effectively only be played when the Turing machine reaches s_f, indicating that $\mathcal{M}$ has accepted the input x. $\texttt{check}(Q')$ for other subsets Q' are used to ensure that Agents sets up the initial configuration of the Turing machine correctly (see the formal definition of the transitions at the end of the construction).

Let us now describe the transitions of $\mathcal{A}$ in more detail. The actions are

- $\Sigma = \Sigma_{\texttt{trans}} \cup \Sigma_{\texttt{check}} \cup \{\texttt{start}\}$, with
  - $\Sigma_{\texttt{trans}} = T \times \{1, \ldots, P(n)\}$
  - $\Sigma_{\texttt{check}} = \left\{\texttt{check}(Q') \mid Q' \in \left\{\{0_i, 1_i\}_{1 \leq i \leq P(n)}, Q_{\texttt{cont}}, Q_{\texttt{cont}} \setminus \{s_f\}, Q_{\texttt{pos}}\right\}\right\}$

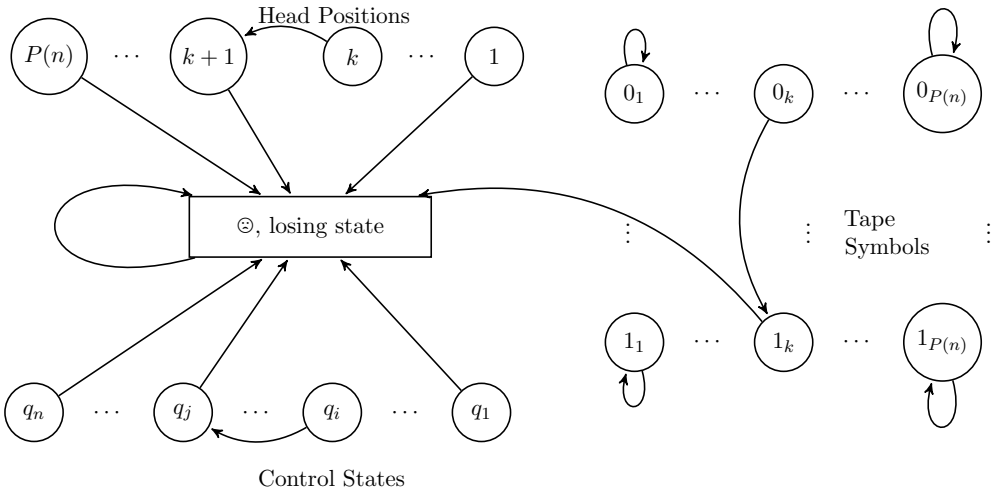


Figure 11: Transitions associated with action $a_{t,k}$ for $t = (q_i, q_j, 0, 1, \rightarrow)$.

To describe the effect of actions from Σ, we use the following terminology: a state $q \in Q$ is called an α-sink for $\alpha \in \Sigma$ if $\Delta(q, \alpha) = \{q\}$, and it is a sink if it is an α-sink for every α. Only the initial `start` action is nondeterministic: for every $\alpha \in \Sigma \setminus \{\texttt{start}\}$, and every $q \in Q$, $\Delta(q, \alpha)$ is a singleton. A consequence is that in the games $\mathcal{A}^m$, the only decision Player 2 makes is in the first step.

The two distinguished states **Acc** and ☹ are sinks. Moreover, any state but q_0 is a `start`-sink. From the initial state q_0, the effect of `start` aims at representing the initial configuration of the Turing machine: $\Delta(q_0, \texttt{start}) = \{s_0, p_1\} \cup \{0_k \mid x_k = 0 \text{ or } k \geq n\} \cup \{1_k \mid x_k = 1\}$. Then, the actions from $\Sigma_{\texttt{trans}}$ simulate transitions of the Turing machine. Precisely, the effect of action $\alpha = ((s_s, s_t, b_r, b_w, d), i) \in \Sigma_{\texttt{trans}}$ is deterministic and as follows:

- $\Delta(s_s, \alpha) = \{s_t\}$
- $\Delta(p_i, \alpha) = \begin{cases} \{p_{i+1}\} & \text{if } i < P(n) \text{ and } d = \rightarrow \\ \{p_{i-1}\} & \text{if } i > 1 \text{ and } d = \leftarrow \\ \{p_i\} & \text{otherwise;} \end{cases}$
- $\Delta(0_i, \alpha) = \begin{cases} \{0_i\} & \text{if } b_r = 0 \text{ and } b_w = 0 \\ \{1_i\} & \text{if } b_r = 0 \text{ and } b_w = 1 \end{cases}$ $\Delta(1_i, \alpha) = \begin{cases} \{0_i\} & \text{if } b_r = 1 \text{ and } b_w = 0 \\ \{1_i\} & \text{if } b_r = 1 \text{ and } b_w = 1; \end{cases}$
- $\Delta(0_j, \alpha) = 0_j$ and $\Delta(1_j, \alpha) = 1_j$, for $j \neq i$;
- otherwise, $\Delta(q, \alpha) = ☹$.

Last, we described how actions from $\Sigma_{\texttt{check}}$ let the system evolve. Let $\texttt{check}(Q') \in \Sigma_{\texttt{check}}$ be a check action for set $Q' \subseteq Q$. Then

- $\Delta(q, \texttt{check}(Q')) = \begin{cases} ☹ & \text{if } q \in Q' \cup \{q_0, ☹\} \\ \textbf{Acc} & \text{otherwise.} \end{cases}$

We claim that this construction ensures the following equivalence:

Lemma 6.2. *$\mathcal{M}$ halts on input x in q_f if and only if $(\mathcal{A}, \textbf{\textit{Acc}})$ is a positive instance of the sure-synchronization problem.*

Proof.

- case $m \leq P(n) + 1$: not enough tokens for player 2 in the first step to cover all of $\Delta(q_0, \texttt{start})$; player 1 wins in the next step by selecting the adequate check action
- case $m \geq P(n) + 2$: best move for player 2 in the first step is to cover all of $\Delta(q_0, \texttt{start})$; afterwards, if player 1 does not mimic the execution of the Turing machine, some tokens get stuck in ☹; thus the best strategy for player 1 is to mimic the execution of the Turing machine; then, the machine halts if and only if all the tokens in $Q_{\texttt{cont}}$ converge to s_f. Now applying $\texttt{check}(Q_{\texttt{cont}} \setminus \{s_f\})$ moves all tokens to **Acc**. □

We thus performed a PTIME reduction of the halting problem for polynomial space Turing machines to the sure synchronization problem, which is therefore PSPACE-hard.

Now, in order to encode an alternating Turing machine, we assume that the control states of $\mathcal{M}$ alternate between states of Controller and states of Agents. The NFA $\mathcal{A}$ is extended with a state ☺, a state C, which represents that Controller decides what transition to take, and one state q_t per transition t of $\mathcal{M}$, which will represent that Agents chooses to play transition t as the next action. Assume first, that C contains at most an agent; we will later explain how to impose this.

The NFA has an additional transition labelled `init` from q_0 to C, and one transition from C to every state q_t labeled by $a_{t',p}$, for every transition t and action $a_{t',p}$. Intuitively, whatever action $a_{t',p}$ is played by Controller, Agents can choose the next action to be associated with t by placing the agent to state q_t.

From state q_t, only actions of the form $a_{t,p}$ are allowed, leading back to C. That is, actions $a_{t',p}$ with $t' \neq t$ lead from q_t to the sink losing state ☹. This encodes that Controller must follow the transition t chosen by Agents. To punish Agents in case the current tape contents is not the one expected by the transition $t = (s, s', b, b', d)$ he chooses, there are checking actions $\texttt{check}_s$ and $\texttt{check}_{p,b}$ enabled from state q_t. Action $\texttt{check}_s$ leads from q_t to ☺, and also from s to ☹. Similarly, $\texttt{check}_{p,b}$ for any position p and $b \in \{0, 1\}$ leads from q_t to ☺ and from any position state $q \neq p$ to ☹, and from b_p to ☹. In this way, Agents will not move the token from C to an undesired q_t. This ensures that Agents places the agents only in a state q_t which agrees with the configuration.

Last, there are transitions on action `end` from state ☺, C and any of the q_t's to the target state ☺. Action `end` loops around the accepting state **Acc** associated with the Turing machine, and it leads from any other state to ☹. Last, there is an action `win`, which leads from ☺ to ☺, from **Acc** to ☺, and from any other state to ☹. This action `win` may seem unnecessary, but its purpose will appear clear in the following step. This whole construction encodes, assuming that there is a single agent in C after the first transition, that Controller can choose the transition from a Controller state of $\mathcal{M}$, and Agents can choose the transition from an Agents state.

Let us now explain how to deal with the case where Agents places several agents in state C on the initial action `init`, enabling the possibility to later send agents to several q_ts simultaneously. With the current gadget, if there is an agent in q_{t1} and one in q_{t2}, then Controller would be stuck as playing $a_{t1,k}$ would send the agent from q_{t2} to ☹, and vice-versa. To handle this case, consider the gadget from Figure 12. We use an extra state `s`, actions $\texttt{store}_t$ for each transition t, and action `restart`.

Action $\texttt{store}_t$ leads from q_t to `s`, and loops on every other state. From all states except ☺ and ☹ (in particular, `s` and every state associated with the Turing machine, including **Acc**), action `restart` leads to q_0. Last, the effects of `end` and `win` are extended as follow to `s`: `end` loops on `s`, while `win` leads from `s` to ☹.

Assume that input x is not accepted by the alternating Turing machine $\mathcal{M}$, and let m be at least $P(n)+3$. In the m-population game, Agents has a winning strategy placing initially a single agent in state C. If Controller plays $\texttt{store}_t$ (for some t), either no agents are stored, or the unique agent in C is moved to $\texttt{s}$. Playing $\texttt{end}$ does not change the configuration, and Controller cannot play $\texttt{win}$. Thus, there is no way to lead the agents encoding the Turing machine configuration to ☺. Playing $\texttt{restart}$ moves all the agents back to the original configuration q_0. This shows that $\texttt{store}_t$ is useless to Controller and thus Agents wins as in the previous case.

Conversely, assume that Controller has a strategy in $\mathcal{M}$ witnessing the acceptance of x. If Agents never split, then Controller never plays any store actions and wins as in the previous case. Otherwise, assume that Agents places at least two agents in C to eventually split them to $t_1, \dots, t_n$. In this case, Controller can play the corresponding actions $\texttt{store}_{t_2}, \dots, \texttt{store}_{t_n}$ moving all agents (but the ones in t_1) in $\texttt{s}$, after which he plays his winning strategy from t_1 resulting in sending at least one agent to ☺. Then, Controller plays $\texttt{restart}$ and proceeds inductively with strictly less agents from q_0, until there is no agent in C, q_t in which case Controller plays $\texttt{win}$ to win. □

Surprisingly, the cut-off can be as high as doubly exponential in the size of the NFA.

Proposition 6.3. *There exists a family of NFA $(\mathcal{A}_n)_{n\in\mathbb{N}}$ such that $|\mathcal{A}_n| = 2n+7$, and for $M = 2^{2^n+1}+n$, there is no winning strategy in $\mathcal{A}_n^M$ and there is one in $\mathcal{A}_n^{M-1}$.*

Proof. Let $n \in \mathbb{N}$. The NFA $\mathcal{A}_n$ we build is the disjoint union of *two* NFAs with different properties, called $\mathcal{A}_{\mathsf{split}}$ and $\mathcal{A}_{\mathsf{count},n}$. On the one hand, for $\mathcal{A}_{\mathsf{split}}$, it requires $\Theta(\log m)$ steps for Controller to win the m-population game. On the other hand, $\mathcal{A}_{\mathsf{count},n}$ implements a usual counter over n bits, such that Controller can avoid losing for $O(2^n)$ steps. In the combined NFA $\mathcal{A}_n$, we require that Controller win in $\mathcal{A}_{\mathsf{split}}$ *and* avoid losing in $\mathcal{A}_{\mathsf{count},n}$. This ensures that $\mathcal{A}_n$ has a cutoff of $\Theta(2^{2^n})$

Recall Figure 1, which presents the splitting gadget $\mathcal{A}_{\mathsf{split}}$. It has the following properties. In $\mathcal{A}_{\mathsf{split}}^m$ with $m \in \mathbb{N}$ agents:

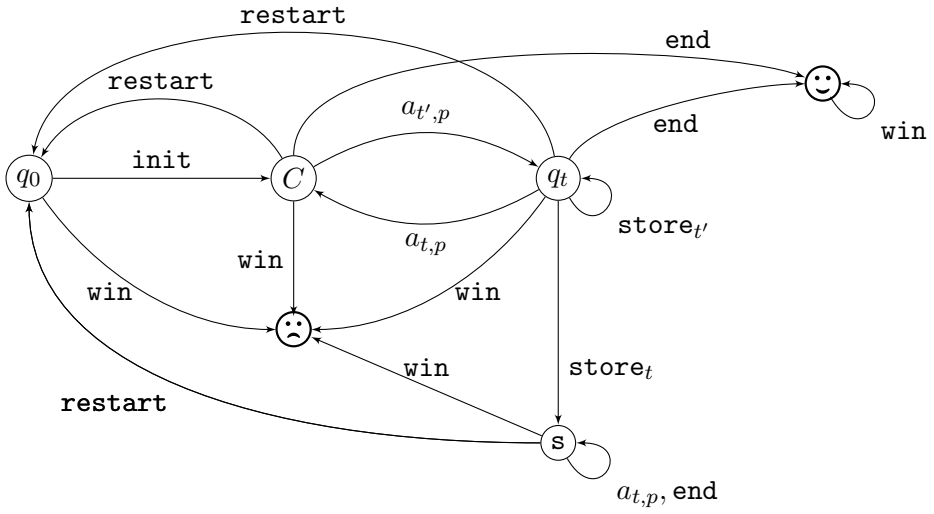


Figure 12: Gadget simulating a single agent in C.

($s1$) **Controller has a strategy to ensure win in $2\lfloor \log_2 m \rfloor + 2$ steps:** Consider the following strategy of Controller.
- Play δ if there is at least one agent in state q_0. Otherwise,
- Play a if the number of agents in q_1 is greater than in q_2.
- Play b if the number of agents in q_2 is greater than or same as in q_1.

For this strategy, let us look at the number of agents in the state q_0 and f respectively. For instance, since the play starts from all agents in q_0, the starting state has count $(m, 0)$ - there are m tokens in q_0 and 0 in f. From a state with counts $(k, m-k)$, after two steps of this strategy (regardless of Agents's play), we will end up in a state with counts $(l, m-l)$ for some $l \leq \lfloor k/2 \rfloor$. Hence within $2 \times \lfloor \log_2 m \rfloor + 2$ steps starting from the initial state, one will reach a state with counts $(0, m)$ and the Controller wins.

($s2$) **No strategy of Controller can ensure a win in less than $2\lfloor \log_2 m \rfloor + 2$ steps:** The transition from q_0 on δ is the only real choice Agents has to make. Assume that Agents decides to send an equal number of agents (up to a difference of 1) to both q_1 and q_2 from q_0. Against this strategy of Agents, let $\alpha_1, \alpha_2, \ldots \alpha_k$ be the shortest sequence of actions by Controller which lead all the agents into f.

We now show that $k \geq 2 \times \lfloor \log_2 m \rfloor + 2$. Initially all agents are in q_0. So by the minimality of k we should have $\alpha_1 = \delta, \alpha_2 \in \{a, b\}, \alpha_3 = \delta, \ldots \alpha_k \in \{a, b\}$, since other actions will not change the state of the agents. For any $i \in \{1, 2, \ldots \frac{k}{2}\}$, after Controller plays α_{2i}, denote the number of agents in q_0 and f by $(l_i, m - l_i)$. Note $l_i \geq \frac{l_{i-1}-1}{2}$, since Agents sends equal number of agents from q_0 to q_1 and q_2. Iterating this equation for any $j \in \{1, 2, \ldots \frac{k}{2}\}$ gives $l_j \geq \frac{l_0+1}{2^j} - 1$, where $l_0 = m$ is the number of agents in state q_0 at the beginning. In particular this shows that $\frac{k}{2} > \log m$ since $l_{\frac{k}{2}} = 0$. Hence $k \geq 2 \times (\lfloor \log_2 m \rfloor + 1)$.

The gadget $\mathcal{A}_{\mathsf{count},n}$, shown in Figure 13, represents a binary counter. For each $i \in \{1 \ldots n\}$, it has states ℓ_i (meaning bit i is 0) and h_i (meaning bit i is 1) and actions α_i (which sets the ith bit and resets the bits for $j < i$). The only real choice that Agents has in this gadget is at the first step, and it is optimal for Agents to place agents in as many states as possible – in particular when $m \geq n$, placing one agent in each l_i for $i \in \{1 \ldots n\}$.

After this, the play is completely determined by Controller's actions. Actually, Controller doesn't have much choice when there is at least one agent is each l_i. Controller must simulate an n-bit counter if it wants to prevent some agent from reaching ☹. More precisely, assume inductively that all the agents are in the states given by $b_n b_{n-1} \ldots b_1$ where $b_k \in \{l_k, h_k\}$ (initially $b_i = l_i$ for each i). If $b_i = h_i$ for each i, then any action α_i will lead the agent in state h_i to ☹. Otherwise, let j be the smallest index such that $b_j = l_j$. Observe that α_j is the only action that doesn't lead some agent to ☹; α_i for $i < j$ would lead the agent in h_i to ☹, while α_i for $i > j$ would lead the agent in l_j to ☹. On playing α_j, the agents now move to the states $b_n b_{n-1} \ldots b_{j-1} h_j l_{j-1} \ldots l_0$ – which can be interpreted as the next number in a binary counter.

This means that the gadget $\mathcal{A}_{\mathsf{count},n}$ has the following properties:

($c1$) For any m, Controller has a strategy in the m-population game on $\mathcal{A}_{\mathsf{count},n}$ to avoid ☹ for 2^n steps by playing α_i whenever the counter suffix from bit i is $01 \cdots 1$;

($c2$) For $m \geq n$, no strategy of Controller in $\mathcal{A}^m_{\mathsf{count},n}$ can avoid ☹ for more than 2^n steps.

To construct $\mathcal{A}_n$, the two gadgets $\mathcal{A}_{\mathsf{split}}$ and $\mathcal{A}_{\mathsf{count},n}$ are combined by adding a new initial state, and an action labeled *init* leading from this new initial state to the initial

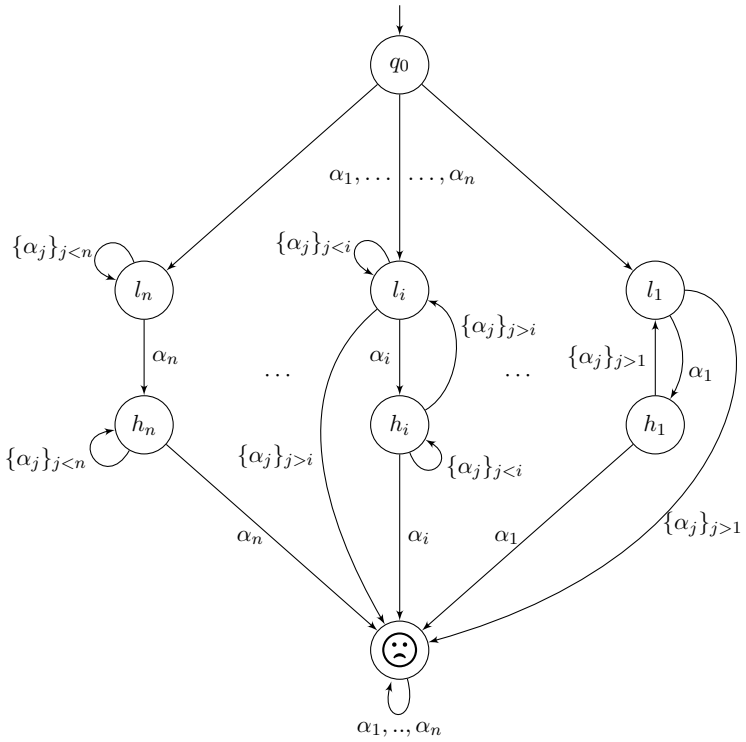


Figure 13: The counting gadget $\mathcal{A}_{\mathsf{count},n}$.

states of both NFAs. Actions for $\mathcal{A}_n$ are made up of pairs of actions, one for each gadget: $\{a, b, \delta\} \times \{\alpha_i \mid 1 \leq i \leq n\}$. We further add an action $*$ which can be played from any state of $\mathcal{A}_{\mathsf{count},n}$ except ☹, and only from f in $\mathcal{A}_{\mathsf{split}}$, leading to the global target state ☺.

Let $M = 2^{2^n+1} + n$. We deduce that the cut-off is $M - 1$ as follows:

- For M agents, a winning strategy for Agents is to first split n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, in order to fill each l_i with 1 token, and 2^{2^n+1} tokens to the q_0 of $\mathcal{A}_{\mathsf{split}}$. Then Agents splits evenly tokens between q_1, q_2 in $\mathcal{A}_{\mathsf{split}}$. In this way, Controller needs at least $2^n + 1$ steps to reach the final state of $\mathcal{A}_{\mathsf{split}}$ $(s2)$, but Controller reaches ☹ after these $2^n + 1$ steps in $\mathcal{A}_{\mathsf{count},n}$ $(c2)$.
- For $M - 1$ agents, Agents needs to use at least n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, else Controller can win easily. But then there are less than 2^{2^n+1} tokens in the q_0 of $\mathcal{A}_{\mathsf{split}}$. And thus by $(s1)$, Controller can reach f within 2^n steps, after which he still avoids ☹ in $\mathcal{A}_{\mathsf{count},n}$ $(c1)$. And then Controller sends all agents to ☺ using $*$.

Thus, the family $(\mathcal{A}_n)$ of NFA exhibits a doubly exponential cut-off. □

7. Discussion

Obtaining an EXPTIME algorithm for the control problem of a population of agents was challenging. We also managed to prove a matching lower-bound. Further, the surprising doubly exponential matching upper and lower bounds on the cut-off imply that the alternative technique, checking that Controller wins all m-population game for m up to the cut-off, is far from being efficient.

The idealised formalism we describe in this paper is not entirely satisfactory: for instance, while each agent can move in a non-deterministic way, unrealistic behaviours can happen, *e.g.* all agents synchronously taking infinitely often the same choice. An almost-sure control problem in a probabilistic formalism should be studied, ruling out such extreme behaviours. As the population is discrete, we may avoid the undecidability that holds for distributions [11] and is inherited from the equivalence with probabilistic automata [17]. Abstracting continuous distributions by a discrete population of arbitrary size could thus be seen as an approximation technique for undecidable formalisms such as probabilistic automata.

Acknowledgement: We are grateful to Gregory Batt for fruitful discussions concerning the biological setting. Thanks to Mahsa Shirmohammadi for interesting discussions. This work was partially supported by ANR project STOCH-MC (ANR-13-BS02-0011-01), and by DST/CEFIPRA/Inria Associated team EQUAVE.

References

[1] Parosh Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *Proceedings of Formats'11*, volume 6919 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2011.

[2] Parosh Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–263, 2003.

[3] S. Akshay, Blaise Genest, Bruno Karelovic, and Nikhil Vyas. On regularity of unary probabilistic automata. In *Proceedings of STACS'16*, volume 47 of *Leibniz International Proceedings in Informatics*, pages 8:1–8:14. Leibniz-Zentrum für Informatik, 2016.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of PODC'04*, pages 290–299. ACM, 2004.

[5] André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 1(303):7–34, 2003.

[6] Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *Proceedings of FSTTCS'13*, volume 24 of *Leibniz International Proceedings in Informatics*, pages 501–513. Leibniz-Zentrum für Informatik, 2013.

[7] Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *Proceedings of FoSSaCS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2014.

[8] Michael Blondin, Javier Esparza, and Antonín Kucera. Automatic analysis of expected termination time for population protocols. In *Proceedings of CONCUR'18*, pages 33:1–33:16, 2018.

[9] Tomás Brázdil, Petr Jančar, and Antonín Kučera. Reachability games on extended vector addition systems with states. In *Proceedings of ICALP'10*, volume 6199 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 2010.

[10] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of STOCS'17*, pages 252–263. ACM, 2017.

[11] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata (erratum). Technical report, CoRR abs/1206.0995, 2012.

[12] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Limit synchronization in Markov decision processes. In *Proceedings of FoSSaCS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2014.

[13] Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of STACS'14*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, 2014.

[14] Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proceedings of LICS'99*, pages 352–359. IEEE Computer Society, 1999.

[15] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017.

[16] Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

[17] Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *Proceedings of ICALP'10*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010.

[18] Marcin Jurdzinski. Small progress measures for solving parity games. In *Proceedings of STACS'00*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[19] Marcin Jurdziński, Ranko Lazić, and Sylvain Schmitz. Fixed-dimensional energy games are in pseudo polynomial time. In *Proceedings of ICALP'15*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015.

[20] Panagiotis Kouvaros and Alessio Lomuscio. Parameterised Model Checking for Alternating-Time Temporal Logic. In *Proceedings of ECAI'16*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1230–1238. IOS Press, 2016.

[21] Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304, 2014.

[22] Mahsa Shirmohammadi. *Qualitative analysis of synchronizing probabilistic systems.* PhD thesis, ULB, 2014.

[23] Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt. In silico control of biomolecular processes. In *Computational Methods in Synthetic Biology*, chapter 13, pages 277–285. Humana Press, Springer, 2015.

[24] Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In *Proceedings of LATA'08*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008.

[25] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. URL: `https://doi.org/10.1016/S0304-3975(98)00009-7`, `doi:10.1016/S0304-3975(98)00009-7`.