The Complexity of Decentralized Control of Markov Decision Processes

Daniel S. Bernstein*
Robert Givan†
Neil Immerman*
Shlomo Zilberstein*

*Department of Computer Science
University of Massachusetts
Amherst, MA 01003

†School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907

Abstract

We consider decentralized control of Markov decision processes and give complexity bounds on the worst-case running time for algorithms that find optimal solutions. Generalizations of both the fully-observable case and the partially-observable case that allow for decentralized control are described. For even two agents, the finite-horizon problems corresponding to both of these models are hard for non-deterministic exponential time. These complexity results illustrate a fundamental difference between centralized and decentralized control of Markov decision processes. In contrast to the problems involving centralized control, the problems we consider *provably* do not admit polynomial-time algorithms. Furthermore, assuming EXP $\neq$ NEXP, the problems require super-exponential time to solve in the worst case.

1 Introduction

Markov decision processes (MDPs) have received considerable attention, and there exist well-known algorithms for finding optimal control strategies in the case where a process is centrally controlled and the controller (or agent) has access to complete state information (Puterman, 1994). Less well understood is the case in which a process is controlled by multiple cooperating distributed agents, each with possibly different information about the state.

We are interested in studying how hard these decentralized control problems are relative to analogous centralized control problems, from the point of view of computational complexity. In particular, we consider two different models of decentralized control of MDPs. One is a generalization of a partially-observable Markov decision process (POMDP), which we call a decentralized partially-observable Markov decision process (DEC-POMDP). In a DEC-POMDP, the process is controlled by multiple distributed agents, each

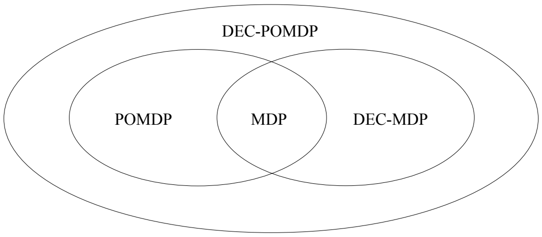


Figure 1: The relationships among the models.

with possibly different information about the state. The other is a generalization of an MDP, called a decentralized Markov decision process (DEC-MDP). A DEC-MDP is a DEC-POMDP with the restriction that at each time step the agents' observations together uniquely determine the state. The MDP, POMDP, and DEC-MDP can all be viewed as special cases of the DEC-POMDP. The relationships among the models are shown in Figure 1.

A number of different problems can be viewed as decentralized control of a Markov process. For example, consider problems involving the control of multiple distributed robots, such as robotic soccer (Coradeschi et al., 2000). In these domains, it is necessary to develop a strategy for each robot under the assumption that the robots will have limited ability to communicate when they execute their strategies. Another problem that fits naturally within this framework is the distributed control of a power grid (Schneider et al., 1999). Finally, several types of networking problems can be viewed within this framework (Altman, 2001).

It would be beneficial to have general-purpose algorithms for solving these decentralized control problems. An algorithm for similar problems was proposed by Ooi et al. (1996). Under the assumption that all agents share state information every K time steps, the authors developed a dynamic programming algorithm to derive optimal policies. A downside of this approach is that the state space for the dynamic programming algorithm grows *doubly exponentially* with K. The only known *tractable* algorithms for these types of problems rely on even more assumptions. One such algorithm was developed by Hsu and Marcus (1982) and works under the assumption that the agents share state information every time step (although it can take one time step for the information to propagate). Approximation algorithms have also been developed for these problems, although they can at best give guarantees of local optimality. For instance, Peshkin et al. (2000) studied algorithms that perform gradient descent in a space of parameterized policies.

Is there something inherent in these problems that forces us to add assumptions and/or use approximation algorithms? Papadimitriou and Tsitsiklis (1982) presented some results aimed at answering this question. The authors proved that a simple decentralized decision-making problem is NP-complete, even with just two decision makers. They later noted that this implies that decentralized control of MDPs must be NP-hard (Papadimitriou & Tsitsiklis, 1986). We strengthen this result by showing that both the DEC-POMDP and DEC-MDP problems are NEXP-hard, even when the horizon is limited to be less than the number of states (and they are NEXP-complete in the latter case). Although it is not known whether the classes P, NP, and PSPACE are distinct, it *is* known that P $\neq$ NEXP, and thus the problems we consider are *provably* intractable. Furthermore, assuming EXP $\neq$ NEXP, the problems take super-exponential time to solve in the worst case. This result is in contrast to the best known bounds for MDPs (P-hard) and POMDPs (PSPACE-hard) (Papadimitriou & Tsitsiklis, 1987; Mundhenk, Goldsmith, Lusena & Allender, 2000). Thus, we have gained insight into the possibility of a fundamental difference between centralized and decentralized control of Markov decision processes.

In Section 2, we give a brief review of the concepts we will need from complexity theory. In Section 3, we define the MDP and POMDP models. Section 4 contains the definitions of the DEC-MDP and DEC-POMDP models, and a proof that the short-horizon versions of these problems fall within the complexity class NEXP. In Section 5, we present our main complexity result — a proof that these decentralized problems are NEXP-hard. Finally, Section 6 contains our conclusions.

2 Computational Complexity

In this section, we give a brief introduction to the theory of computational complexity. More detail can be found in (Papadimitriou, 1994). A complexity class is a set of problems, where a problem is an infinite set of problem instances, each of which has a "yes" or "no" answer. In order to discuss the complexity of optimization problems, we must have a way of converting them to "yes/no" problems. The typical way this is done is to set a threshold and ask whether or not the optimal solution yields a reward that is no less than this threshold. The problem of actually finding the optimal solution can of course be no easier than the threshold problem.

The first complexity class we consider is P, the set of problems that can be solved in polynomial time (in the size of the problem instance) on a sequential computer. NP is the set of problems that can be solved *nondeterministically* in polynomial time. A nondeterministic machine automatically knows the correct path to take any time there is a choice as to how the computation should proceed. An example of a problem that can be solved nondeterministically in polynomial time is deciding whether a sentence of propositional logic is satisfiable. The machine can guess an assignment of truth values to variables and evaluate the resulting expression in polynomial time. Of course, nondeterministic machines do not really exist, and the most efficient known algorithms for simulating them take exponential time in the worst case. In fact, it is strongly believed by most complexity theorists that P $\neq$ NP (but this has *not* been proven formally).

Complexity can also be measured in terms of the amount of *space* a computation requires. One class, PSPACE, includes all problems that can be solved in polynomial space. Any problem that can be solved in polynomial time or nondeterministic polynomial time can be solved in polynomial space (i.e., P $\subseteq$ NP $\subseteq$ PSPACE) — that P $\subseteq$ PSPACE can be seen informally by observing that only polynomially much space can be accessed in polynomially many time steps.

Moving up the complexity hierarchy, we have exponential time (EXP) and nondeterministic exponential time (NEXP). By exponential time, we mean time bounded by 2^{n^k}, where n is the input size and $k > 0$ is a constant. It is known that PSPACE $\subseteq$ EXP $\subseteq$ NEXP, and it is believed that EXP $\neq$ NEXP (but again this has not been proven). It *has* been proven that the classes P and EXP are distinct, however.

The notion of a *reduction* is important in complexity theory. We say that a problem A is *reducible* to a problem B if any instance x of A can be converted into an instance $f(x)$ of B such that the answer to x is "yes" if and only if the answer to $f(x)$ is "yes." A problem A is said to be *hard* for a complexity class C (or C-hard) if any problem in C is *efficiently* reducible to A. If the complexity class in question is P, efficient means that $f(x)$ can be computed using at most logarithmic space, while for the classes above P, efficient means that $f(x)$ can be computed using at most polynomial time. A problem A is said to be *complete* for a complexity class C (or C-complete) if (a) A is contained in C, and (b) A is hard for C. For instance, the satisfiability problem mentioned above is NP-complete and P-hard. However, unless P = NP, satisfiability is not P-complete.

3 Centralized Models

In this paper, we consider discrete-time finite sequential decision processes under the undiscounted finite-horizon optimality criterion. We build into our problem definitions the (unusual) assumption that the horizon is less than the number of states. Note that this assumption actually *strengthens* the hardness results; the general problems must be at least as hard as their short-horizon counterparts. Unfortunately, the assumption is needed for each of the upper bounds given below. Finding tight upper bounds for problems with arbitrary horizons remains an open problem (Blondel & Tsitsiklis, 1999, Section 5).

Below we describe the partially-observable Markov decision process and its associated decision problem. The Markov decision process is viewed as a restricted version of this model.

A *partially-observable Markov decision process (POMDP)* is defined as follows. We are given a tuple $\langle S, A, P, R, \Omega, O, T, K\rangle$, where

- S is a finite set of states, with distinguished initial state s_0.
- A is a finite action set.
- P is a transition probability table. $P(s, a, s')$ is a rational representing the probability of transitioning from s to s' on taking action a. Here $s, s' \in S$ and $a \in A$.
- R is a reward function. $R(s, a, s')$ is a rational representing the reward obtained from taking action a from state s and transitioning to state s'. Again, $s, s' \in S$ and $a \in A$.
- Ω is a finite set of observations.
- O is a table of observation probabilities. $O(s, a, s', o)$ is a rational representing the probability of observing o when taking action a in state s and transitioning to state s' as a result. Here $s, s' \in S$, $a \in A$, and $o \in \Omega$.
- T is a positive integer representing the horizon (and $T < |S|$).
- K is a rational representing the threshold value.

A POMDP is *fully observable* if there exists a mapping $J : \Omega \to S$ such that whenever $O(s, a, s', o)$ is nonzero, $J(o) = s'$. A *Markov decision process (MDP)* is defined to be a POMDP that is fully observable.

A *policy* δ is defined to be a mapping from sequences of observations $\overline{o} = o_1 \cdots o_t$ over Ω to actions in A. We wish to find a policy that maximizes the expected total return over the finite horizon. The definitions below are used to formalize this notion. We use the symbol ϵ to denote the empty observation sequence. For an observation sequence $\overline{o} = o_1 \cdots o_t$, $\overline{o}o$ is taken to represent the sequence $o_1 \cdots, o_t o$.

Definition: The probability of transitioning from a state s to a state s' following policy δ while the agent sees observation sequence $\overline{o}$, written $\overline{P}_\delta(s, \overline{o}, s')$, can be defined recursively as follows:

$$\overline{P}_\delta(s, \epsilon, s) = 1,$$

$$\overline{P}_\delta(s, \overline{o}o, s') = \sum_{q \in S} \overline{P}_\delta(s, \overline{o}, q) P(q, \delta(\overline{o}), s') O(q, \delta(\overline{o}), s', o),$$

where ϵ is the empty sequence.

Definition: The value $V_\delta^T(s)$ of following policy δ from state s for T steps is given by the following equation:

$$V_\delta^T(s) = \sum_{\overline{o}} \sum_{q \in S} \sum_{s' \in S} \overline{P}_\delta(s, \overline{o}, q) P(q, \delta(\overline{o}), s') R(q, \delta(\overline{o}), s'),$$

where the observation sequences have length at most $T - 1$.

The decision problem corresponding to a finite-horizon POMDP is as follows: Given a POMDP $D = \langle S, A, P, R, \Omega, O, T, K \rangle$, is there a policy for which $V_\delta^T(s_0)$ equals or exceeds K? It was shown in (Papadimitriou & Tsitsiklis, 1987) that the decision problem for POMDPs is PSPACE-complete and that the decision problem for MDPs is P-complete.

4 Decentralized Models

We now describe extensions to the aforementioned models that allow for decentralized control. In these models, at each step, each agent receives a *local* observation and subsequently chooses an action. The state transitions and rewards received depend on the vector of actions of all the agents.

A *decentralized partially-observable Markov decision process (DEC-POMDP)* is defined formally as follows (for ease of exposition, we describe the two-agent case). We are given $\langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T, K \rangle$, where

- S is a finite set of states, with distinguished initial state s_0.
- A_1 and A_2 are finite action sets.
- P is a transition probability table. $P(s, a_1, a_2, s')$ is a rational representing the probability of transitioning from s to s' on taking actions a_1, a_2. Here $s, s' \in S$, $a_1 \in A_1$, and $a_2 \in A_2$.
- R is a reward function. $R(s, a_1, a_2, s')$ is a rational representing the reward obtained from taking actions a_1, a_2 from state s and transitioning to state s'. Again $s, s' \in S$, $a_1 \in A_1$, and $a_2 \in A_2$.
- Ω_1 and Ω_2 are finite sets of observations.
- O is a table of observation probabilities. $O(s, a_1, a_2, s', o_1, o_2)$ is a rational representing the probability of observing o_1, o_2 when taking actions a_1, a_2 in state s and transitioning to state s' as a result. Here $s, s' \in S$, $a_1 \in A_1$, $a_2 \in A_2$, $o_1 \in \Omega_1$, and $o_2 \in \Omega_2$.
- T is a positive integer representing the horizon (and $T < |S|$).
- K is a rational representing the threshold value.

A DEC-POMDP generalizes a POMDP by allowing for control by multiple distributed agents that together may not fully observe the system state (so we have only partial observability). We also define a generalization of MDP problems by requiring joint observability. We say that a DEC-POMDP is *jointly observable* if there exists a mapping $J : \Omega_1 \times \Omega_2 \to S$ such that whenever $O(s, a_1, a_2, s', o_1, o_2)$ is nonzero, $J(o_1, o_2) = s'$. We define a *decentralized Markov decision process (DEC-MDP)* to be a DEC-POMDP that is jointly observable.

We define a *local policy for agent i*, δ_i, to be a mapping from *local* histories of observations $\overline{o}_i = o_{i1} \cdots o_{it}$ over Ω_i, to actions in A_i. A *joint policy*, $\delta = \langle \delta_1, \delta_2 \rangle$, is defined to be a pair of local policies, one for each agent. We wish to find a joint policy that maximizes the expected total return over the finite horizon. As in the centralized case, we need some definitions to make this notion more formal.

Definition: The probability of transitioning from a state s to a state s' following joint policy $\delta = \langle \delta_1, \delta_2 \rangle$ while agent 1 sees observation sequence $\overline{o}_1$ and agent 2 sees $\overline{o}_2$ of the same length, written $\overline{P}_\delta(s, \overline{o}_1, \overline{o}_2, s')$, can be defined recursively as follows:

$$\overline{P}_\delta(s, \epsilon, \epsilon, s) = 1,$$

$$\overline{P}_\delta(s, \overline{o}_1 o_1, \overline{o}_2 o_2, s') = \sum_{q \in S} \overline{P}_\delta(s, \overline{o}_1, \overline{o}_2, q) P(q, \delta_1(\overline{o}_1), \delta_2(\overline{o}_2), s') O(q, \delta_1(\overline{o}_1), \delta_2(\overline{o}_2), s', o_1, o_2),$$

where ϵ is the empty sequence.

Definition: The value $V_\delta^T(s)$ of following policy $\delta = \langle \delta_1, \delta_2 \rangle$ from state s for T steps is given by the following equation:

$$V_\delta^T(s) = \sum_{\langle \overline{o}_1, \overline{o}_2 \rangle} \sum_{q \in S} \sum_{s' \in S} \overline{P_\delta}(s, \overline{o}_1, \overline{o}_2, q) P(q, \delta_1(\overline{o}_1), \delta_2(\overline{o}_2), s') R(q, \delta_1(\overline{o}_1), \delta_2(\overline{o}_2), s'),$$

where the observation sequences are of length at most $T - 1$, and both sequences in any pair are of the same length.

The decision problem is stated as follows: Given a DEC-POMDP $D = \langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T, K \rangle$, is there a joint policy for which $V_\delta^T(s_0)$ equals or exceeds K? We let DEC-POMDP$_m$ and DEC-MDP$_m$ denote the decision problems for the m-agent DEC-POMDP and the m-agent DEC-MDP, respectively.

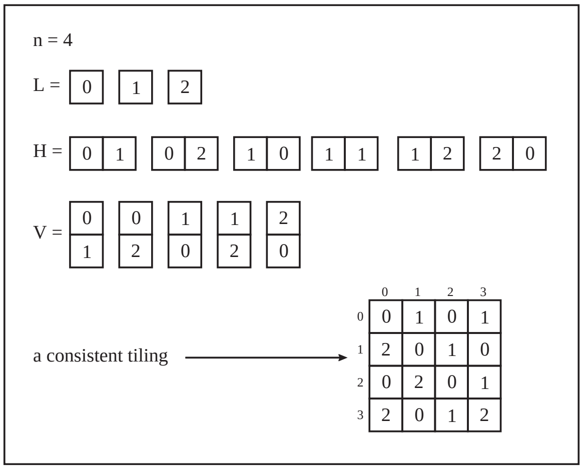
We conclude this section by showing a straightforward upper bound on the worst-case time complexity of DEC-POMDP$_m$ for any $m \geq 2$. Because any DEC-MDP is trivially a DEC-POMDP, this upper bound also applies to DEC-MDP$_m$.

Theorem 1 *For all $m \geq 2$, DEC-POMDP$_m$ $\in$ NEXP.*

Proof: We must show that a nondeterministic machine can solve any instance of DEC-POMDP$_m$ using at most exponential time. First, a joint policy δ can be "guessed" and written down in exponential time. This is because a joint policy consists of m mappings from local histories to actions, and since $T < |S|$, all histories have length less than $|S|$. A DEC-POMDP together with a joint policy can be viewed as a POMDP together with a policy, where the observations in the POMDP correspond to the observation m-tuples in the DEC-POMDP (one from each agent), and the POMDP actions correspond to m-tuples of DEC-POMDP actions (again, one from each agent). In exponential time, each of the exponentially many possible sequences of observations can be converted into a belief state (i.e., a probability distribution over the state set giving the probability of being in each state after seeing the given observation sequence). We note that every POMDP (Kaelbling, Littman & Cassandra, 1998) is equivalent to a "belief-state MDP" whose state set is the set of reachable belief states of the POMDP. The transition probabilities and expected rewards for the corresponding exponential-sized belief-state MDP can be computed in exponential time. Using standard MDP solution techniques (Puterman, 1994), it is possible to determine whether the guessed policy yields expected reward at least K in this belief-state MDP in time that is at most polynomial in the size of the belief-state MDP, which is exponential in the size of the original DEC-POMDP problem. Therefore, there exists an accepting computation path if and only if there is a policy that can achieve reward K. □

5 Decentralized Control of MDPs is NEXP-Hard

We now turn our attention to proving that the upper bound just shown in Theorem 1 is tight — specifically, we show that NEXP is also a lower bound for the worst-case time complexity of decentralized problems by


n = 4
L = 0 1 2
H = 0 1 | 0 2 | 1 0 | 1 1 | 1 2 | 2 0
V = 0/1 | 0/2 | 1/0 | 1/2 | 2/0
a consistent tiling

	0	1	2	3
0	0	1	0	1
1	2	0	1	0
2	0	2	0	1
3	2	0	1	2

Figure 2: An example of a tiling instance.

showing that any problem in the class NEXP can be reduced in polynomial time to a DEC-MDP$_2$ problem. It then follows that both DEC-MDP$_m$ and DEC-POMDP$_m$ are NEXP-complete for any $m \geq 2$.

The proof of this lower bound is quite involved and will occupy most of the remainder of this paper. Each subsection of this section contains a piece of the development, and at the end of the section the main theorem is asserted. We begin by introducing the known NEXP-complete problem TILING used in the proof. We then present an overview of the proof and its major constituents. Next we present the reduction from TILING formally, and finally we prove that the reduction is correct.

5.1 The TILING Problem

We can show this lower bound by reducing any NEXP-complete problem to DEC-MDP$_2$ using a polynomial-time algorithm. For our reduction, we use an NEXP-complete problem called TILING (Lewis, 1978; Papadimitriou, 1994, p. 501), which is described as follows. We are given a board size n (represented compactly in binary), a set of tile types $L = \{\mathsf{tile\text{-}0}, \ldots, \mathsf{tile\text{-}k}\}$, and a set of binary horizontal and vertical compatibility relations $H, V \subseteq L \times L$. A *tiling* is a mapping $f : \{0, \ldots, n-1\} \times \{0, \ldots, n-1\} \rightarrow L$. A tiling f is *consistent* if and only if (a) $f(0,0) = \mathsf{tile\text{-}0}$, and (b) for all x, y $\langle f(x,y), f(x+1,y)\rangle \in H$, and $\langle f(x,y), f(x,y+1)\rangle \in V$. The decision problem is to determine, given L, H, V, and n, whether a consistent tiling exists. An example of a tiling instance and a corresponding consistent tiling is shown in Figure 2.

In the remainder of this section, we assume that we have fixed an arbitrarily chosen instance of the tiling problem, so that L, H, V, and n are fixed. We then construct an instance of DEC-MDP that is solvable if and only if the selected tiling instance is solvable. We note that the DEC-MDP instance must be constructible in time polynomial in the size of the tiling instance (which in particular is logarithmic in the value of n), which will require the DEC-MDP instance to be at most polynomially larger than the tiling instance.

5.2 Overview of the Reduction

The basic idea of our reduction is to create a two-agent DEC-MDP that randomly selects two tiling locations bit by bit, informing one agent of the first location and the other agent of the second location. The agents' local policies are observation-history based, so the agents can base their future actions on the tiling locations given to them. After generating the locations, the agents are simultaneously "queried" (i.e., a state is reached in which their actions are interpreted as answers to a query) for a tile type to place at the location given. We design the DEC-MDP problem so that the only way the agents can achieve nonnegative expected reward is to base their answers to the query on a single jointly-understood tiling that meets the constraints of the tiling problem.

This design is complicated because the DEC-MDP state set itself cannot remember which tiling locations were selected (this would cause exponential blowup in the size of the state set, but our reduction must expand the problem size at most polynomially — we note that the tiling grid itself is not part of the tiling problem size, only the compactly represented grid size n is in the problem specification); the state will only contain certain limited information about the relative locations of the two tile positions. The difficulty of the design is also increased by the fact that any information remembered about the specified tiling locations must be shared with at least one of the agents to satisfy the joint observability requirement. To deal with these two issues, we have designed the DEC-MDP to pass through the following phases (a formal description follows later):

Select Phase **Select two bit indices and values, each identifying a bit position and the value at that position in the location given to one of the agents**. These are the only bits that are remembered in the state set from the locations given to the agents in the next phase — the other location bits are generated and forgotten by the process.

The bit values remembered are called value-1 and value-2, and the indices to which these values correspond are called index-1 and index-2. Bit index-1 of the address given to agent 1 will have the value value-1, and likewise for index-2, value-2, and agent 2.

Generate Phase **Generate two tile locations at random, revealing one to each agent.** The bits selected in the above select phase are used, and the other location bits are generated at random and immediately "forgotten" by the DEC-MDP state set.

Query Phase **Query each agent for a tile type to place in the location that was specified to that agent.** These tile types are remembered in the state.

Echo Phase **Require the agents to echo the tile locations they received in the generate phase bit by bit.** In order to enforce the accuracy of these location echoes, the DEC-MDP is designed to yield a negative reward if the bit remembered from the original location generation is not correctly echoed (the DEC-MDP is designed to ensure that each agent cannot know which bit is being checked in its echoes). As the agents echo the bits, the process computes state information representing whether the locations are equal or adjacent horizontally or vertically, and whether the agents' locations are both $(0, 0)$ (again, we cannot just remember the location bits because it would force an exponential state set). The echo phase allows us to compute state information about adjacency/equality of the locations *after* the tile types have been chosen, so that the agents' tile choices cannot depend on this information. This is critical in making the reduction correct.

Test Phase **Check whether the tile types provided in the query phase come from a single consistent tiling.** In other words, check that if the agents were asked for the same location they gave the same tile types during query, if they were asked for adjacent locations they gave types that satisfy the relevant adjacency constraints, and if the agents were both queried for location $(0, 0)$ they both gave tile type tile-0. The process gives a zero reward only if the tile types selected during the query phase meet any applicable constraints as determined by the echoed location bits. Otherwise, a negative reward is obtained.

Note that because we are designing a DEC-MDP, we are required to maintain joint observability: the observations given to the agents at each time step must be sufficient to reconstruct all aspects of the DEC-MDP state at that time step. In particular, the bit indices and values selected in the select phase must be known to the agents (jointly), as well as the information computed in the echo phase regarding the relative position of the two locations.

We achieve this joint observability by making all aspects of the DEC-MDP state observable to both agents, except for the indices and values selected in the select phase and the tile types that are given by the agents (and stored by the process) during the query phase. Each agent can observe which bit index and value are being remembered from the *other* agent's location, and each agent can observe the stored tile type it gave (but not the tile type given by the other agent). Because each agent can see what bit is saved from the other agent's location, we say that one location bit of each agent's location is *visible* to the other agent.

We call the five phases just described "select," "generate," "query," "echo," and "test" in the development below. A formal presentation of the DEC-MDP just sketched follows below, but first we outline the proof that this approach represents a correct reduction.

5.3 Overview of the Correctness Proof

Here we give an overview of our argument that the reduction sketched above is correct in the sense that there exists a policy that achieves expected total reward zero at the start state if and only if there is a solution to the tiling problem we started with.

It is straightforward to show that if there exists a consistent tiling there must exist a policy achieving zero reward. The agents need only "agree on" a consistent tiling ahead of time, and base their actions on the agreed upon tiling (waiting during selection and generation, giving the tile type present at the generated location during query, faithfully echoing the generated location during echo, and then waiting during test — at each point being guaranteed a zero reward by the structure of the problem). Note that it does not matter how expensive it might be to find and represent a consistent tiling or to carry out the policy just described because we are merely arguing for the existence of such a policy.

We now outline the proof of the harder direction, that if there is no consistent tiling then there is no policy achieving expected reward zero. Note that since all rewards are nonpositive, any chance of receiving any negative reward forces the expected total reward to be negative.

Consider an arbitrary policy that yields expected reward zero. Our argument rests on the following claims, which will be proved as lemmas in Section 5.5:

Claim 1. The policy must repeat the two locations correctly during the echo phase.

Claim 2. When executing the policy, the agents' selected actions during the query phase determine a single tiling, as follows. We define a query situation to be *dangerous* to an agent if and only if the observable bit value of the other agent's location (in the observation history) agrees with the bit value at the same index in the agent's own location (so that as far as the agent in danger knows, the other agent is being queried about the same location). During dangerous queries, the tile type selected by the agent in danger must depend only on the location queried (and not on the index or value of the bit observed from the other agent, on any other observable information, or on which agent is selecting the tile type). The agents' selected actions for dangerous queries thus determine a single tiling.

Claim 3. The single tiling from Claim 2 is a consistent tiling.

Claim 3 directly implies that if there is no consistent tiling, then all policies have negative expected reward, as desired.

5.4 Formal Presentation of the Reduction

Now we give the two-agent DEC-MDP $D = \langle S, A_1, A_2, P, R, \Omega_1, \Omega_2, O, T, K \rangle$ that is constructed from the selected tiling instance $\langle L, H, V, n \rangle$. We assume throughout that n is a power of two. It is straightforward to modify the proof to deal with the more general case — one way to do so is summarized briefly in Appendix A.

5.4.1 The State Set

We describe the state set S of D below by giving a sequence of finite-domain "state variables," and then taking the state set to be the set of all possible assignments of values to the state variables.

The Finite-State Automaton One of the state variables will be maintained by a finite-state automaton (FSA) described in Appendix A. This variable, called rel-pos because it maintains a record of the relative position of the two location addresses echoed by the agents in the echo phase, can take on values from the state set of the FSA. Appendix A describes that state set, Q, and also defines two functions (FSANEXT and FSA) based on the underlying FSA. These functions allow us to refer to the critical FSA behaviors here in our reduction while deferring most other details of the FSA to Appendix A. The function FSANEXT updates the state variable rel-pos based on one more bit of echoed location from each agent. The function FSA updates the state variable rel-pos based on a sequence of echoed location bits from each agent — so FSA is defined as a repeated application of the function FSANEXT.

Appendix A also describes distinguished subsets of the FSA state set Q called apart, equal, hor, and ver representing possible relative positions for pairs of locations (not adjacent or equal, equal, horizontally adjacent, and vertically adjacent, respectively). These subsets are used below in defining the transition and reward behavior of the DEC-MDP D. Appendix A also defines the initial state q_0 of the FSA.

The State Variables We now list the state variables defining the state set for D. We list the variables in three groups: the first group is observable to both agents, the second group only to agent 1, and the third group only to agent 2. These restrictions on observability are described below in section 5.4.4

Observable to both agents:

$\mathsf{phase} \in \{\mathsf{select}, \mathsf{gen}, \mathsf{query}, \mathsf{echo}, \mathsf{test}\}$	current phase of the process
$\mathsf{index} \in \{0, \ldots, 2 \log n\}$	index of next location bit to be generated/echoed
$\mathsf{origin} \in \{\mathsf{yes}, \mathsf{no}\}$	eventually true if both tile locations are $(0, 0)$
$\mathsf{rel\text{-}pos} \in Q$	relative tile positions during echo — controlled by the FSA

Observable only to agent 1:

$\mathsf{index\text{-}2} \in \{0, \ldots, 2 \log n - 1\}$	index of bit remembered for agent 2
$\mathsf{value\text{-}2} \in \{0, 1\}$	value of bit remembered for agent 2
$\mathsf{pos\text{-}bit\text{-}1} \in \{0, 1\}$	bit for transmitting tile position to agent 1
$\mathsf{tile\text{-}sel\text{-}1} \in L$	tile type selected by agent 1 in query

Observable only to agent 2:

$\mathsf{index\text{-}1} \in \{0, \ldots, 2 \log n - 1\}$	index of bit remembered for agent 1
$\mathsf{value\text{-}1} \in \{0, 1\}$	value of bit remembered for agent 1
$\mathsf{pos\text{-}bit\text{-}2} \in \{0, 1\}$	bit for transmitting tile position to agent 2
$\mathsf{tile\text{-}sel\text{-}2} \in L$	tile type selected by agent 2 in query

We write a state by enumerating its variable values, e.g., as follows: $\langle \mathsf{gen}, 3, \mathsf{yes}, q_0; 4, 0, 1, \mathsf{tile\text{-}1}; 5, 1, 0, \mathsf{tile\text{-}3} \rangle \in S$. Semicolons are used to group together variables that have the same observability properties. We can represent sets of states by writing sets of values in some of the components of the tuple rather than just values. The "*" symbol is used to represent the set of all possible values for a component. We sometimes use a state variable as a function from states to domain values for that variable. For instance, if q matches $\langle \mathsf{gen}, *, *, *; *, *, *, *; *, *, *, * \rangle$, then we will say $\mathsf{phase}(q) = \mathsf{gen}$.

The initial state s_0 is as follows: $\langle \mathsf{select}, 0, \mathsf{yes}, q_0; 0, 0, 0, \mathsf{tile\text{-}0}; 0, 0, 0, \mathsf{tile\text{-}0} \rangle$.

5.4.2 The Action Sets and Table of Transition Probabilities

We must allow "wait" actions, "zero" and "one" actions for echoing location address bits, and tile type actions from the set of tile types L for answering during the query phase. We therefore take the action sets $A_1 = A_2$ to be $\{\mathsf{wait}, 0, 1\} \cup L$.

We give the transition distribution $P(s, a_1, a_2, s')$ for certain action pairs a_1, a_2 for certain source states s. For any source-state/action-pair combination not covered by the description below, the action pair is taken to cause a probability 1.0 self-transition back to the source state. The combinations not covered are not reachable from the initial state under any joint policy. Also, we note that the FSA-controlled state component $\mathsf{rel\text{-}pos}$ does not change from its initial state q_0 until the echo phase.

Select Phase. This is the first step of the process. In this step, the process chooses, for each agent, which of that agent's bits it will be checking in the echo phase. The value of that bit is also determined in this step. Transition probabilities when $\mathsf{phase} = \mathsf{select}$ are given as follows.

$P(s, a_1, a_2, s') = \frac{1}{(4 \log n)^2}$ in the following situations:

$s = s_0 = \langle \mathsf{select}, 0, \mathsf{yes}, q_0; 0, 0, 0, \mathsf{tile\text{-}0}; 0, 0, 0, \mathsf{tile\text{-}0} \rangle$,
$s' = \langle \mathsf{gen}, 0, \mathsf{yes}, q_0; i_2, v_2, 0, \mathsf{tile\text{-}0}; i_1, v_1, 0, \mathsf{tile\text{-}0} \rangle$,
$i_1, i_2 \in \{0, \ldots, 2 \log n - 1\}$, and
$v_1, v_2 \in \{0, 1\}$.

Generate Phase. During these steps, the two tile positions are chosen bit by bit. Note that we have to check for whether we are at one of the bits selected during the select phase, so that the value of the bit is the same as the value chosen during selection. Transition probabilities when $\mathsf{phase} = \mathsf{generate}$ are given as follows. The second case describes the deterministic transition from the generate phase to the query phase.

$P(s, a_1, a_2, s') = \frac{1}{h}$ in the following situations:
$s = \langle \mathsf{gen}, k, \mathsf{yes}, q_0; i_2, v_2, *, \mathsf{tile\text{-}0}; i_1, v_1, *, \mathsf{tile\text{-}0}\rangle$ where $0 \leq k \leq 2\log n - 1$,
$s' = \langle \mathsf{gen}, k+1, \mathsf{yes}, q_0; i_2, v_2, b_1, \mathsf{tile\text{-}0}; i_1, v_1, b_2, \mathsf{tile\text{-}0}\rangle$, where
$b_1 = v_1$ if $k = i_1$, else b_1 is either 0 or 1,
$b_2 = v_2$ if $k = i_2$, else b_2 is either 0 or 1, and
h is the number of allowed settings of b_1, b_2 from the previous two lines.

$P(s, a_1, a_2, s') = 1$ in the following situations:
$s = \langle \mathsf{gen}, 2\log n, \mathsf{yes}, q_0; i_2, v_2, *, \mathsf{tile\text{-}0}; i_1, v_1, *, \mathsf{tile\text{-}0}\rangle$, and
$s' = \langle \mathsf{query}, 0, \mathsf{yes}, q_0; i_2, v_2, 0, \mathsf{tile\text{-}0}; i_1, v_1, 0, \mathsf{tile\text{-}0}\rangle$.

Query Phase. The query phase consists of just one step, during which each agent chooses a tile type. Transition probabilities when $\mathsf{phase} = \mathsf{query}$ are given as follows.

$P(s, a_1, a_2, s') = 1$ in the following situations:
$s = \langle \mathsf{query}, 0, \mathsf{yes}, q_0; i_2, v_2, 0, \mathsf{tile\text{-}0}; i_1, v_1, 0, \mathsf{tile\text{-}0}\rangle$,
$t_1 = \begin{cases} a_1 & \text{if } a_1 \in L \\ \mathsf{tile\text{-}0} & \text{otherwise} \end{cases}, \quad t_2 = \begin{cases} a_2 & \text{if } a_2 \in L \\ \mathsf{tile\text{-}0} & \text{otherwise} \end{cases}$, and
$s' = \langle \mathsf{echo}, 0, \mathsf{yes}, q_0; i_2, v_2, 0, t_1; i_1, v_1, 0, t_2\rangle$.

Echo Phase. During the echo phase the agents are asked to repeat back the addresses seen in the generate phase, and information about the relative position of the addresses is calculated by the FSA described in Appendix A and recorded in the state. The FSA is accessed here using the function FSANEXT described in Appendix A. Transition probabilities when $\mathsf{phase} = \mathsf{echo}$ are given as follows.

$P(s, a_1, a_2, s') = 1$ in the following situations:
$s = \langle \mathsf{echo}, k, o, q; i_2, v_2, 0, t_1; i_1, v_1, 0, t_2\rangle$,
$b_1 = \begin{cases} a_1 & \text{if } a_1 \in \{0,1\} \\ 0 & \text{otherwise} \end{cases}, \quad b_2 = \begin{cases} a_2 & \text{if } a_2 \in \{0,1\} \\ 0 & \text{otherwise} \end{cases}$,
$s' = \langle p, k', o', \text{FSANEXT}(q, b_1, b_2); i_2, v_2, 0, t_1; i_1, v_1, 0, t_2\rangle$,
where $p, k' = \begin{cases} \mathsf{echo}, k+1 & \text{for } 0 \leq k < 2\log n - 1 \\ \mathsf{test}, 0 & \text{for } k = 2\log n - 1 \end{cases}$, and
$o' = \mathsf{yes}$ if and only if ($o = \mathsf{yes}$ and $a_1 = a_2 = 0$).

Test Phase. The test phase consists of just one step terminating the process in a zero-reward absorbing state.

$P(s, a_1, a_2, s') = 1$ in the following situations:
$s = \langle \mathsf{test}, 0, *, *; *, *, 0, *; *, *, 0, *\rangle$, and
$s' = \langle \mathsf{test}, 0, \mathsf{yes}, q_0; 0, 0, 0, \mathsf{tile\text{-}0}; 0, 0, 0, \mathsf{tile\text{-}0}\rangle$.

5.4.3 The Reward Function

We now describe the reward function for D. The reward $R(s, a_1, a_2, s')$ given when transitioning from state s to state s' taking action pair a_1, a_2 is -1 in any situation except those situations matching one of the following patterns. Roughly, we give zero reward for waiting during select and generate, for answering with a tile type during query, for echoing a bit consistent with any remembered information during echo, and for having given tile types satisfying the relevant constraints during test. The relevant constraints during test are determined by the rel-pos state component computed by the FSA during the echo phase.

$R(s, a_1, a_2, s') = 0$ if and only if one of the following holds:

Select phase: $s = \langle \mathsf{select}, *, *, *; *, *, *, *; *, *, *, * \rangle$ and $a_1 = a_2 = \mathsf{wait}$.

Generate phase: $s = \langle \mathsf{gen}, *, *, *; *, *, *, *; *, *, *, * \rangle$ and $a_1 = a_2 = \mathsf{wait}$.

Query phase: $s = \langle \mathsf{query}, *, *, *; *, *, *, *; *, *, *, * \rangle$ and both $a_1 \in L$ and $a_2 \in L$.

Echo phase: $s = \langle \mathsf{echo}, k, *, *; i_2, v_2, *, *; i_1, v_1, *, * \rangle$ and $a_1, a_2 \in \{0, 1\}$,
where $(a_1 = v_1$ or $k \neq i_1)$ and $(a_2 = v_2$ or $k \neq i_2)$.

Test Phase (i): $s = \langle \mathsf{test}, *, o, \mathsf{equal}; *, *, *, t_1; *, *, *, t_1 \rangle$ and $a_1 = a_2 = \mathsf{wait}$,
where $(o = \mathsf{no}$ or $t_1 = \mathsf{tile\text{-}0})$.

Test Phase (ii): $s = \langle \mathsf{test}, *, *, \mathsf{hor}; *, *, *, t_1; *, *, *, t_2 \rangle$ and $a_1 = a_2 = \mathsf{wait}$,
where $\langle t_1, t_2 \rangle \in H$.

Test Phase (iii): $s = \langle \mathsf{test}, *, *, \mathsf{ver}; *, *, *, t_1; *, *, *, t_2 \rangle$ and $a_1 = a_2 = \mathsf{wait}$,
where $\langle t_1, t_2 \rangle \in V$.

Test Phase (iv): $s = \langle \mathsf{test}, *, *, \mathsf{apart}; *, *, *, *; *, *, *, * \rangle$ and $a_1 = a_2 = \mathsf{wait}$.

5.4.4 Observations, Threshold, and Horizon

The first four component fields of each state description are fully visible to both agents. The last eight state component fields are split into two groups of four, each group visible only to one agent. We therefore take the agent 1 observations Ω_1 to be partial assignments to the following state variables: phase, index, origin, rel-pos, index-2, value-2, pos-bit-1, and tile-sel-1. Similarly, the observations Ω_2 are partial assignments to the following state variables: phase, index, origin, rel-pos, index-1, value-1, pos-bit-2, and tile-sel-2. The observation distribution $O(s, a_1, a_2, s', o_1, o_2)$ simply reveals the indicated portion of the just-reached state s' to each agent deterministically.

We say that an observation sequence is p-phase if the sequence matches the pattern $*\langle p, *, *, *; *, *, *, * \rangle$, where the first "*" stands for any observation sequence. Here, p can be any of gen, query, echo, or test.

We take the horizon T to be $4 \log n + 4$, because the process spends one step in each of the select, query, and test phases, $2 \log n + 1$ steps in the generate phase, and $2 \log n$ steps in the echo phase. We take the threshold value K to be 0. This completes the construction of the DEC-MDP by polynomial-time reduction from the selected tiling instance. An example of a zero-reward trajectory of the process is shown in Figure 3. We now turn to correctness.

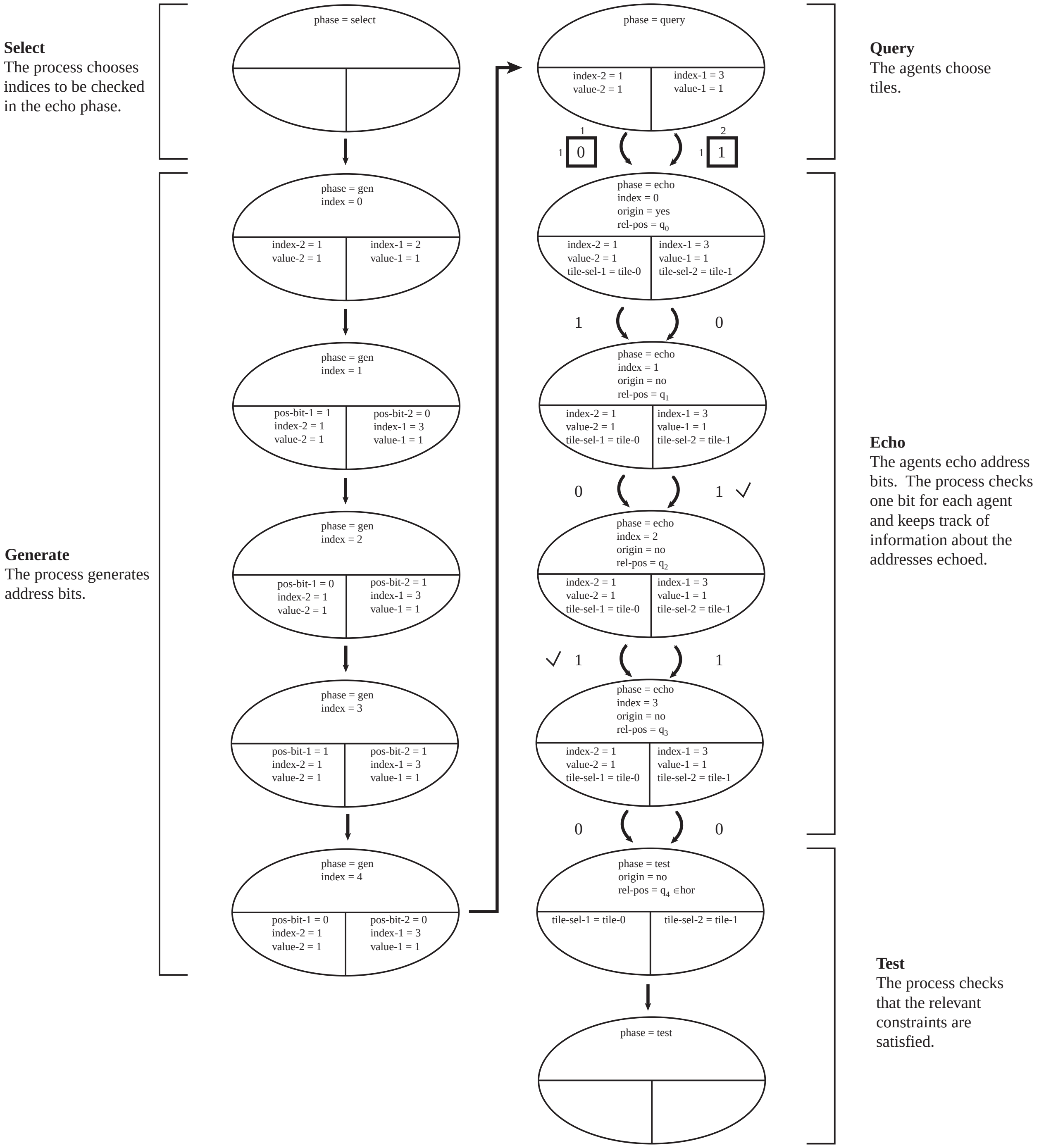


Figure 3: An example of a zero-reward trajectory of the process constructed from the tiling example given in Figure 2. The total reward is zero because the agents echo the "checked" bits correctly and choose tiles that do not violate any constraints, given the two addresses that are echoed. (For clarity, some state components are not shown.)

5.5 Formal Correctness Argument

Next we show that the reduction presented above is indeed correct. Our main claim is that there exists a policy that achieves expected total reward zero at the start state if and only if there is a solution to the tiling problem we started with.

To make our notation easier to read, we define the following abbreviations.

Definition: Given an observation sequence $\overline{o}_1$ over Ω_1, we write $\mathrm{loc}_1(\overline{o}_1)$ for the location value represented by the bits transmitted to agent 1 in the generate phase of the process. We note that during the select and generate phases this value may be only partially specified (because not all of the bits have been generated). More precisely, $\mathrm{loc}_1(\overline{o}_1) = b_k \cdots b_0$, where the b_i values are chosen by the first match of the following sequence in $\overline{o}_1$ (with k as large as possible while allowing a match):

$$\langle \mathsf{gen}, 1, *, *; *, *, b_0, * \rangle \cdots \langle \mathsf{gen}, k+1, *, *; *, *, b_k, * \rangle.$$

We define $\mathrm{loc}_2(\overline{o}_2)$ similarly. In addition, we define $\mathrm{bit}(i, l)$ to be b_i, where $b_k, \ldots, b_0$ is the binary coding of the (possibly only partially specified) location l — we take $\mathrm{bit}(i, l)$ to be undefined if the bit i is not specified in location l.

By abuse of notation we treat $\mathrm{loc}_1(\overline{o}_1)$ (or $\mathrm{loc}_2(\overline{o}_2)$) as a tiling location (x, y) sometimes (but only when it is fully specified) and as a bit string at other times.

The easier direction of correctness is stated in the following lemma, which is formally proven in Appendix B.

Lemma 1 *If there exists a consistent tiling, then there must exist a policy achieving zero expected total reward.*

We now discuss the more difficult reverse direction of the correctness proof. In the following subsections, we prove Claims 1 to 3 from Section 5.3 to show that if there is a policy which achieves nonnegative expected reward for horizon T, then there is also a consistent tiling. *Throughout the remainder of the proof, we focus on a fixed, arbitrary policy δ that achieves zero expected reward.* Given this policy, we must show that there is a consistent tiling.

5.5.1 Proof of Claim 1

Before proving the first claim, we need to formalize the notion of "faithfulness during echo."

Definition: A pair of observation sequences $\langle \overline{o}_1, \overline{o}_2 \rangle$ over Ω_1 and Ω_2, respectively, is said to be *reachable* if $\overline{P}_\delta(s_0, \overline{o}_1, \overline{o}_2, s)$ is nonzero for some state s. An observation sequence $\overline{o}_1$ over Ω_1 is said to be *reachable* if there exists an observation sequence $\overline{o}_2$ over Ω_2 such that the pair of observation sequences $\langle \overline{o}_1, \overline{o}_2 \rangle$ is reachable. Likewise $\overline{o}_2$ over Ω_2 is reachable if there is some $\overline{o}_1$ over Ω_1 such that $\langle \overline{o}_1, \overline{o}_2 \rangle$ is reachable.

Definition: The policy $\delta = \langle \delta_1, \delta_2 \rangle$ is *faithful during echo* if it satisfies both of the following conditions for all indices k in $[0, 2\log n - 1]$, and all reachable observation sequence pairs $\langle \overline{o}_1, \overline{o}_2 \rangle$:

1. $\delta_1(\overline{o}_1) = \mathrm{bit}(k, \mathrm{loc}_1(\overline{o}_1))$ when $\overline{o}_1 = \langle *, *, *, *; *, *, *, * \rangle \cdots \langle \mathsf{echo}, k, *, *; *, *, *, * \rangle$.
2. $\delta_2(\overline{o}_2) = \mathrm{bit}(k, \mathrm{loc}_2(\overline{o}_2))$ when $\overline{o}_2 = \langle *, *, *, *; *, *, *, * \rangle \cdots \langle \mathsf{echo}, k, *, *; *, *, *, * \rangle$.

We say the policy δ *lies during echo* otherwise. If the two conditions listed above are satisfied for all indices k in $[0, d-1]$, where $0 \leq d \leq 2\log n$, we say that the policy *faithfully echoes the first d bits.*

Much of our proof revolves around showing that the reachability of a pair of observation sequences is not affected by making certain changes to the sequences. We focus without loss of generality on changes to the observations of agent 2, but similar results hold for agent 1. The changes of particular interest are changes to the (randomly selected) value of the index-1 state component — this is the component that remembers which bit of agent 1's queried location will be checked during echo. It is important to show that agent 1 cannot determine which bit is being checked before that bit has to be echoed. To show this, we define a way to vary the observation sequences seen by agent 2 (preserving reachability) such that without changing the observations seen by agent 1 we have changed which agent 1 address bit is being checked. We now present this approach formally.

Definition: We say that an observation sequence $\overline{o}_1$ over Ω_1 is *superficially consistent* if the values of the index-2 component and the value-2 component do not change throughout the sequence, and the value of the tile-sel-1 component is tile-0 for generate-phase and query-phase observations and some fixed tile type in L for echo-phase and test-phase observations. Given a superficially consistent observation sequence $\overline{o}_1$, we can write index-2$(\overline{o}_1)$ and value-2$(\overline{o}_1)$ to denote the value of the indicated component throughout the sequence. In addition, we can write tile-sel-1$(\overline{o}_1)$ to denote the fixed tile type for echo-phase and test-phase observations (we take tile-sel-1$(\overline{o}_1)$ to be tile-0 if the sequence contains no echo- or test-phase observations). Corresponding definitions hold for observation sequences over Ω_2, replacing "1" by "2" and "2" by "1" throughout.

Note that any reachable observation sequence must be superficially consistent, but the converse is not necessarily true. The following technical definition is necessary so that we can discuss the relationships between observation sequences without assuming reachability.

Definition: We say that two superficially consistent observation sequences $\overline{o}_1$ over Ω_1 and $\overline{o}_2$ over Ω_2 are *compatible* if

$$\begin{gathered}\mathrm{bit}(\mathsf{index\text{-}1}(\overline{o}_2), \mathrm{loc}_1(\overline{o}_1)) = \mathsf{value\text{-}1}(\overline{o}_2) \text{ or this bit of } \mathrm{loc}_1(\overline{o}_1) \text{ is not defined,}\\ \text{and}\\ \mathrm{bit}(\mathsf{index\text{-}2}(\overline{o}_1), \mathrm{loc}_2(\overline{o}_2)) = \mathsf{value\text{-}2}(\overline{o}_1) \text{ or this bit of } \mathrm{loc}_2(\overline{o}_2) \text{ is not defined.}\end{gathered}$$

Definition: Given an index i in $[0, 2\log n - 1]$, a reachable pair of observation sequences $\langle \overline{o}_1, \overline{o}_2 \rangle$, and an observation sequence $\overline{o}_2'$ over Ω_2, we say that $\overline{o}_2'$ is an *i-index variant of $\overline{o}_2$ relative to $\overline{o}_1$* when $\overline{o}_2'$ is any sequence compatible with $\overline{o}_1$ that varies from $\overline{o}_2$ only as follows:

1. index-1 has been set to i throughout the sequence,
2. value-1 has been set to the same value v throughout the sequence,
3. pos-bit-2 can vary arbitrarily from $\overline{o}_2$, and
4. for any echo- or test-phase observations, tile-sel-2 has been set to the tile type selected by δ on the query-phase prefix of $\overline{o}_2'$, or to tile-0 if δ selects a non-tile-type action on that query.

If the pos-bit-2 components of $\overline{o}_2$ and $\overline{o}_2'$ are identical, we say that $\overline{o}_2'$ is a *same-address* index variant of $\overline{o}_2$.

We note that, given a reachable pair of observation sequences $\langle \overline{o}_1, \overline{o}_2 \rangle$, there exists an i-index variant of $\overline{o}_2$ relative to $\overline{o}_1$, for any i in $[0, 2\log n - 1]$. This remains true even if we allow only same-address index variants. The following technical lemma asserts that index variation as just defined preserves reachability under very general conditions. Its proof is deferred to Appendix C.

Lemma 2 *Suppose δ is faithful during echo for the first k bits of the echo phase, for some k. Let $\langle \overline{o}_1, \overline{o}_2 \rangle$ be a reachable pair of observation sequences that end no later than the kth bit of echo (i.e., the last observation in the sequence has* index *no greater than k if it is an echo-phase observation), and let $\overline{o}_2'$ be an i-index variant of $\overline{o}_2$ relative to $\overline{o}_1$, for some i. If the observation sequences are echo-phase or test-phase, then we require that the index variation be a same-address variation. We can then conclude that $\langle \overline{o}_1, \overline{o}_2' \rangle$ is reachable.*

We are now ready to assert and prove Claim 1 from Section 5.3.

Lemma 3 *(Claim 1) δ is faithful during echo.*

Proof: We argue by induction that δ faithfully echoes all $2 \log n$ address bits. As an inductive hypothesis, we assume that δ faithfully echoes the first k bits, where $0 \leq k < 2 \log n$. Note that if k equals zero, this is a null assumption, providing an implicit base case to our induction. Now suppose for contradiction that δ lies during the $k+1$st step of the echo phase. Then one of the agents' policies must incorrectly echo bit $k+1$; we assume without loss of generality that this is so for agent 1, i.e., under some reachable observation sequence pair $\langle \overline{o}_1, \overline{o}_2 \rangle$ of length $2 \log n + k + 2$, the policy δ_1 dictates that the agent choose action $1 - \mathrm{bit}(k, \mathrm{loc}_1(\overline{o}_1))$. Lemma 2 implies that the observation sequence pair $\langle \overline{o}_1, \overline{o}_2' \rangle$ is also reachable, where $\overline{o}_2'$ is any same-address $k+1$-index variant of $\overline{o}_2$ relative to $\overline{o}_1$.

Since all of the agent 1 observations are the same for both $\langle \overline{o}_1, \overline{o}_2 \rangle$ and $\langle \overline{o}_1, \overline{o}_2' \rangle$, when the latter sequence occurs, agent 1 chooses the same action $1 - \mathrm{bit}(k, \mathrm{loc}_1(\overline{o}_1))$ as given above for the former sequence, and a reward of -1 is obtained (because in this case it is bit $k+1$ that is checked). Therefore, the expected total reward is not zero, yielding a contradiction. □

5.5.2 Proof of Claim 2

Now we move on to prove Claim 2 from Section 5.3. We show that δ can be used to define a particular mapping from tile locations to tile types based on "dangerous queries." In Section 5.3, we defined an agent 1 observation sequence to be "dangerous" if it reveals a bit of agent 2's queried location that agrees with the corresponding bit of agent 1's queried location (and vice versa for agent 2 observation sequences). We now present this definition more formally.

Definition: A query-phase observation sequence $\overline{o}_1$ over Ω_1 is *dangerous* if it is reachable and

$$\mathrm{bit}(\textsf{index-2}(\overline{o}_1), \mathrm{loc}_1(\overline{o}_1)) \quad = \quad \textsf{value-2}(\overline{o}_1).$$

Likewise, a query-phase sequence $\overline{o}_2$ over Ω_2 is dangerous if it is reachable and $\mathrm{bit}(\textsf{index-1}(\overline{o}_2), \mathrm{loc}_2(\overline{o}_2)) = \textsf{value-1}(\overline{o}_2)$.

Dangerous query-phase sequences are those for which the agent's observations are consistent with the possibility that the other agent has been queried on the same location. We note that for any desired query location l, and for either agent, there exist dangerous observation sequences $\overline{o}$ such that $\mathrm{loc}_k(\overline{o}) = l$. Moreover, such sequences still exist when we also require that the value of $\textsf{index-}k(\overline{o})$ be any particular desired value (where k is the number of the non-observing agent).

Lemma 4 *Two same-length query-phase observation sequences, $\overline{o}_1$ over Ω_1 and $\overline{o}_2$ over Ω_2, are reachable together as a pair if and only if they are compatible and each is individually reachable.*

Proof: The "only if" direction of the theorem follows easily — the reachability part follows from the definition of reachability, and the compatibility of jointly reachable sequences follows by a simple induction on sequence length given the design of D.

The "if" direction can be shown based on the following assertions. First, a generate-phase observation sequence (for either agent) is reachable if and only if it matches the following pattern:

$$\langle \mathsf{gen}, 0, \mathsf{yes}, q_0; i, v, *, \mathsf{tile\text{-}0}\rangle \cdots \langle \mathsf{gen}, k, \mathsf{yes}, q_0; i, v, *, \mathsf{tile\text{-}0}\rangle,$$

for some k, i, and v — this can be established by a simple induction on sequence length based on the design of D. A similar pattern applies to the query phase.

Given two compatible reachable sequences of the same length, $\overline{o}_1$ and $\overline{o}_2$, we know by the definition of reachability that there must be some sequence $\overline{o}_2'$ such that $\langle \overline{o}_1, \overline{o}_2' \rangle$ is reachable. But given the patterns just shown for reachable sequences, $\overline{o}_2$ and $\overline{o}_2'$ can differ only in their choice of i, v, and in the address given to agent 2 via the pos-bit-2 component. It follows that $\overline{o}_2$ is an i-index variant of $\overline{o}_2'$ relative to $\overline{o}_1$, for some i. Lemma 2 then implies that the pair $\langle \overline{o}_1, \overline{o}_2 \rangle$ is reachable as desired. □

Lemma 5 *(Claim 2) There exists a mapping f from tiling locations to tile types such that $f(loc_i(\overline{o})) = \delta_i(\overline{o})$ on all dangerous queries $\overline{o}$ over Ω_i for both agents ($i \in \{1, 2\}$).*

Proof: To prove the lemma, we prove that for any two dangerous query sequences $\overline{o}_i$ and $\overline{o}_j$ over Ω_i and Ω_j respectively for arbitrary $i, j \in \{1, 2\}$, if $\mathrm{loc}_i(\overline{o}_i) = \mathrm{loc}_j(\overline{o}_j) = l$ then $\delta_i(\overline{o}_i) = \delta_j(\overline{o}_j)$. This implies that for any such $\overline{o}_i$ we can take $f(l) = \delta_i(\mathrm{loc}_i(\overline{o}_i))$ to construct f satisfying the lemma. Suppose not. Then there must be a counterexample for which $i \neq j$ — because given a counterexample for which $i = j$, either $\overline{o}_i$ or $\overline{o}_j$ must form a counterexample with any dangerous query $\overline{o}_k$ over Ω_{1-i} such that $\mathrm{loc}_{1-i}(\overline{o}_k) = l$.

We can now consider a counterexample where $i \neq j$. Let $\overline{o}_1$ and $\overline{o}_2$ be dangerous (and thus reachable) sequences over Ω_1 and Ω_2, respectively, such that $\mathrm{loc}_1(\overline{o}_1) = \mathrm{loc}_2(\overline{o}_2)$ but $\delta_1(\overline{o}_1) \neq \delta_2(\overline{o}_2)$. Note that $\mathrm{loc}_1(\overline{o}_1) = \mathrm{loc}_2(\overline{o}_2)$ together with the fact that $\overline{o}_1$ and $\overline{o}_2$ are dangerous implies that $\overline{o}_1$ and $\overline{o}_2$ are compatible and thus reachable together (using Lemma 4).

The faithfulness of echo under δ (proven in Claim 1, Lemma 3) then ensures that the extension (there is a single extension because D is deterministic in the echo and test phases) of these observation sequences by following δ to the test phase involves a faithful echo. The correctness of the FSA construction in Appendix A then ensures that the rel-pos state component after this extension will have the value equal. The reward structure of D during the test phase then ensures that to avoid a negative reward the tile types given during query, $\delta_1(\overline{o}_1)$ and $\delta_2(\overline{o}_2)$, must be the same, contradicting our choice of $\overline{o}_1$ and $\overline{o}_2$ above and thus entailing the lemma. □

5.5.3 Proof of Claim 3 and our Main Hardness Theorem

We now finish the proof of our main theorem by proving Claim 3 from Section 5.3. We start by showing the existence of a useful class of pairs of dangerous observation sequences that are reachable together.

Lemma 6 *Given any two locations l_1 and l_2 sharing a single bit in their binary representations, there are* *dangerous observation sequences* $\overline{o}_1$ *over* Ω_1 *and* $\overline{o}_2$ *over* Ω_2 *such that:*

$$\begin{array}{rcl} loc_1(\overline{o}_1) & = & l_1, \\ loc_2(\overline{o}_2) & = & l_2, \text{ and} \\ \langle \overline{o}_1, \overline{o}_2 \rangle & & \text{is reachable.} \end{array}$$

Proof: It is straightforward to show that there exist dangerous observation sequences $\overline{o}_1$ over Ω_1 and $\overline{o}_2$ over Ω_2 such that $\mathrm{loc}_1(\overline{o}_1) = l_1$ and $\mathrm{loc}_2(\overline{o}_2) = l_2$ as desired. In these sequences, both index-1 and index-2 are set throughout to the index of a single bit shared by l_1 and l_2. Since this bit is in common, these sequences are compatible, so by Lemma 4 they are reachable together. □

Lemma 7 *(Claim 3) The mapping f defined in Lemma 5 is a consistent tiling.*

Proof: We prove the contrapositive. If the mapping f is not a consistent tiling, then there must be some particular constraint violated by f. It is easy to show that any such constraint is tested during the test phase if $\mathrm{loc}_1(\overline{o}_1)$ and $\mathrm{loc}_2(\overline{o}_2)$ have the appropriate values. (The faithfulness during echo claim proven in Lemma 3 implies that the origin and rel-pos components on entry to the test phase will have the correct values for comparing the two locations). For example, if a horizontal constraint fails for f, then there must be locations (i, j) and $(i+1, j)$ such that the tile types $\langle f(i, j), f(i+1, j) \rangle$ are not in H — since these two locations share a bit (in fact, all the bits in j, at least) Lemma 6 implies that there are dangerous $\overline{o}_1$ and $\overline{o}_2$ with $\mathrm{loc}_1(\overline{o}_1) = (i, j)$ and $\mathrm{loc}_2(\overline{o}_2) = (i+1, j)$ that are reachable together. During the test phase, the tile-sel-1 and tile-sel-2 state components are easily shown to be $f(i, j)$ and $f(i+1, j)$, and then the definition of the reward function for D ensures a reachable negative reward. The arguments for the other constraints are similar. □

Claim 3 immediately implies that there exists a consistent tiling whenever there exists a policy achieving zero expected total reward. This completes the proof of the other direction of our main complexity result. We have thus shown that there exists a policy that achieves expected reward zero if and only if there exists a consistent tiling, demonstrating that DEC-MDP$_2$ is NEXP-hard.

Theorem 2 *DEC-MDP$_2$ is NEXP-hard.*

Corollary 1 *For all $m \geq 2$, both DEC-POMDP$_m$ and DEC-MDP$_m$ are NEXP-complete.*

6 Discussion

Using the tools of worst-case complexity analysis, we analyzed two variants of decentralized control of Markov decision processes. Specifically, we proved that the finite-horizon m-agent DEC-POMDP problem is NEXP-hard for $m \geq 2$ and the finite-horizon m-agent DEC-MDP problem is also NEXP-hard for $m \geq 2$. When the horizon is limited to be less than the number of states, the problems are NEXP-complete.

The results have some theoretical implications. First, unlike the MDP and POMDP problems, the problems we studied *provably* do not admit polynomial-time algorithms, since P $\neq$ NEXP. Second, we have drawn a connection between work on Markov decision processes and the body of work in complexity theory that deals with the exponential jump in complexity due to decentralization (Peterson & Reif, 1979; Babai, Fortnow & Lund, 1991).

There are also more direct implications for researchers trying to solve problems of this nature. Consider the growing body of work on algorithms for obtaining exact or approximate solutions for POMDPs (e.g., Jaakkola, Singh & Jordan, 1995; Cassandra, Littman & Zhang, 1997; Hansen, 1998; Meuleau, Kim, Kaelbling & Cassandra, 1999; Lusena, Goldsmith, Li, Sittinger & Wells, 1999; Zhang, 2001). For the finite-horizon case, we now have stronger evidence that there is no way to efficiently convert a DEC-MDP or DEC-POMDP into an equivalent POMDP and solve it using established techniques. This knowledge can provide direction for research on the development of algorithms for these problems.

Finally, consider the infinite-horizon versions of the aforementioned problems. It has recently been shown that the infinite-horizon POMDP problem is undecidable (Madani, Hanks & Condon, 1999) under several different optimality criteria. Since a POMDP is a special case of a DEC-POMDP, the corresponding infinite-horizon DEC-POMDP problems are also undecidable. In addition, because it is possible to reduce a POMDP to a two-agent DEC-MDP (simply add a second "dummy" agent that observes the state but cannot affect the state transitions and rewards obtained), the infinite-horizon DEC-MDP problems are also undecidable.

A Description of the Finite-State Automaton Used

Here we describe the domain of the rel-pos state component and how the component's value evolves during the echo phase based on the bit pairs chosen by the agents (recall that it remains fixed during the other phases). The component is controlled by a deterministic finite state automaton (FSA) with a state set Q of size polylogarithmic in n. The state set is assumed to include four distinguished subsets of states: apart, equal, hor, and ver. Each subset corresponds to a possible relation between the two agents' echoed tile positions. The automaton takes as input the string of bit pairs (the alphabet for the automaton consists of the four symbols $[00], [01], [10], [11]$, with the first component of each symbol representing the bit produced by agent 1 and the second component representing the bit produced by agent 2). This automaton is the cross product of three individual automata, each of which keeps track of a different piece of information about the two tile positions represented by the sequence of bit pairs. These automata are described as follows:

1. Equal Tile Positions
   This automaton computes whether the two tile positions produced are equal or not. Consider the following regular expression:
   $$([00] + [11])^*.$$
   There is a constant-sized FSA corresponding to the above expression that, on inputs of length $2 \log n$, ends in an accept state if and only if $(x_1, y_1) = (x_2, y_2)$, where (x_1, y_1) is the tile position represented by the sequence of bits given by agent 1, and (x_2, y_2) is the tile position represented by the sequence of bits given by agent 2.

2. Horizontally Adjacent Tile Positions
   This automaton computes whether the second tile position is horizontally adjacent to the first tile position by a single increment in the x coordinate. Its regular expression is as follows:
   $$[10]^*[01]([00] + [11])^* \underbrace{([00] + [11]) \cdots ([00] + [11])}_{\log n}.$$
   There is an $O(\log n)$-sized FSA corresponding to the above expression that, on inputs of length $2 \log n$, ends in an accept state if and only if $(x_1 + 1, y_1) = (x_2, y_2)$, where x_1, y_1, x_2, and y_2 are as in the description of the first automaton. (We note that it is not always the case that a regular expression has a corresponding FSA that is only polynomially bigger. However, for all the regular expressions we consider this property does hold.)

3. Vertically Adjacent Tile Positions
   This automaton computes whether the second tile position is vertically adjacent to the first tile position by a single increment in the y coordinate. Its regular expression is as follows:

$$\underbrace{([00]+[11])\cdots([00]+[11])}_{\log n}[10]^*[01]([00]+[11])^*.$$

   There is an $O(\log n)$-sized FSA corresponding to the above expression that, on inputs of length $2\log n$, ends in an accept state if and only if $(x_1, y_1 + 1) = (x_2, y_2)$ where x_1, y_1, x_2, and y_2 are as in the descriptions of the previous two automata.

We can take the cross product of these three automata to get a new automaton with size $O((\log n)^2)$. Let accept_1, accept_2, and accept_3 be the sets of accept states from the three component automata, respectively, and let reject_1, reject_2, and reject_3 be the corresponding sets of reject states. From these sets we construct distinguished sets apart, equal, hor, and ver of cross-product automaton states as follows.

$$\begin{aligned}
\mathsf{apart} &= \text{reject}_1 \times \text{reject}_2 \times \text{reject}_3.\\
\mathsf{equal} &= \text{accept}_1 \times \text{reject}_2 \times \text{reject}_3.\\
\mathsf{hor} &= \text{reject}_1 \times \text{accept}_2 \times \text{reject}_3.\\
\mathsf{ver} &= \text{reject}_1 \times \text{reject}_2 \times \text{accept}_3.
\end{aligned}$$

The rest of the automaton's states comprise the set Q'. Let $q_{1,0}$, $q_{2,0}$, and $q_{3,0}$ denote the start states of the three component automata. We define the start state of the cross-product automaton to be the state $q_0 = \langle q_{1,0}, q_{2,0}, q_{3,0}\rangle$.

We now define two functions based on this automaton that are needed in the main body of the proof. One function takes as input the state of the automaton and a bit pair, and returns the next state of the automaton. The second function takes as input a pair of bit strings of the same length and returns the state that the automaton will be in starting from its initial state and reading symbols formed by the corresponding bits in the two strings in sequence.

Definition: For $q \in Q$ and $a_1, a_2 \in \{0,1\}$, $\text{FSANEXT}(q, a_1, a_2) = q'$, where $q' \in Q$ is the resulting state if the automaton starts in state q and reads the input symbol $[a_1 a_2]$.

Definition: The function FSA is defined inductively as follows:

$$\text{FSA}(\epsilon, \epsilon) = q_0.$$

$$\text{FSA}(b_0 \cdots b_{k+1}, c_0 \cdots c_{k+1}) = \text{FSANEXT}(\text{FSA}(b_0 \cdots b_k, c_0 \cdots c_k)), b_{k+1}, c_{k+1}).$$

Note that the range of FSA for inputs of length $2\log n$ is $\mathsf{apart} \cup \mathsf{equal} \cup \mathsf{hor} \cup \mathsf{ver}$.

In the proof given in Section 5 we assumed that the TILING grid size n was an exact power of two. We note that the proof can be adapted by adding two components to the cross-product FSA described here, where the two new components are both FSAs over the same alphabet. The first new component accepts a string only when both x_1 and y_1 (as described above) are less than n (so that the tiling location represented by (x_1, y_1) is in the tiling grid). The second new component behaves similarly for (x_2, y_2). The DEC-MDP can then be constructed using the smallest power of two larger than n, but modified so that whenever either new component of the FSA rejects the (faithfully) echoed bit sequences, then the process gives a zero reward regardless of the tile types returned during query.

Each new component can be viewed as an FSA over a {0,1} alphabet, because each focuses either on just the agent 1 echoes or on just the agent 2 echoes. We describe the FSA for checking that x_1 is less than n — constructing the two components is then straightforward. Suppose that $k = \lceil \log n \rceil$ is the number of bits in the binary representation of n, and that the bits themselves are given from least to most significant as $b_1 \cdots b_k$. Suppose also that there are j different bits equal to 1 among $b_1 \cdots b_k$, and that these bits are at indices $i_1, \ldots, i_j$. We can then write a regular expression for detecting that its input of k bits from least to most significant represents a number in binary that is strictly less than n:

$$\left[(0+1)^{i_1-1}0\,b_{i_1+1}\cdots b_k\right] + \left[(0+1)^{i_2-1}0\,b_{i_2+1}\cdots b_k\right] + \cdots + \left[(0+1)^{i_j-1}0\,b_{i_j+1}\cdots b_k\right].$$

It can be shown that this regular expression has an equivalent FSA of size $O((\log n)^2)$.

B Proof of Lemma 1

We assume there exists at least one consistent tiling, and we select a particular such mapping f. We describe a policy $\delta = \langle \delta_1, \delta_2 \rangle$ that achieves zero expected reward at the initial state. δ_1 is a mapping from sequences of observations in Ω_1 to actions in A_1, and δ_2 from sequences over Ω_2 to actions in A_2. Only the reachable part of the mapping δ_1 is specified below — any unspecified observation sequence maps to the action wait. We note that δ_1 and δ_2 are symmetric.

$\delta_1(\overline{o}_1) = a_1$ when one of the following holds:

Select phase: $\overline{o}_1 = \langle \mathsf{select}, *, *, *; *, *, *, * \rangle$ and $a_1 = \mathsf{wait}$.

Generate phase: $\overline{o}_1 = *\langle \mathsf{gen}, *, *, *; *, *, *, * \rangle$ and $a_1 = \mathsf{wait}$.

Query phase: $\overline{o}_1 = *\langle \mathsf{query}, *, *, *; *, *, *, * \rangle$ and $a_1 = f(\mathrm{loc}_1(\overline{o}_1))$.

Echo phase: $\overline{o}_1 = *\langle \mathsf{echo}, k, *, *; *, *, *, * \rangle$ and $a_1 = \mathrm{bit}(k, \mathrm{loc}_1(\overline{o}_1))$.

Test phase: $\overline{o}_1 = *\langle \mathsf{test}, *, *, *; *, *, *, * \rangle$ and $a_1 = \mathsf{wait}$.

The local policy δ_2 is defined identically to δ_1 except that loc_1 is replaced by loc_2.

We first characterize the set of all reachable states from s_0 under the policy δ. We then note that taking the action prescribed by δ from any of these states yields a reward of zero. Thus, $V_\delta^T(s_0) = 0$.

It is straightforward to show by induction that $\overline{P}_\delta(s_0, \overline{o}_1, \overline{o}_2, s)$ is zero except where one of the following patterns applies:

- $s = s_0 = \langle \mathsf{select}, 0, \mathsf{yes}, q_0; 0, 0, 0, \mathsf{tile\text{-}0}; 0, 0, 0, \mathsf{tile\text{-}0} \rangle$.
- $s = \langle \mathsf{gen}, k, \mathsf{yes}, q_0; i_2, v_2, *, \mathsf{tile\text{-}0}; i_1, v_1, *, \mathsf{tile\text{-}0} \rangle$, where
  $0 \le k \le 2 \log n$,
  ($k \le i_1$ or $v_1 = \mathrm{bit}(i_1, \mathrm{loc}_1(\overline{o}_1))$), and
  ($k \le i_2$ or $v_2 = \mathrm{bit}(i_2, \mathrm{loc}_2(\overline{o}_2))$).
- $s = \langle \mathsf{query}, 0, \mathsf{yes}, q_0; i_2, v_2, *, \mathsf{tile\text{-}0}; i_1, v_1, *, \mathsf{tile\text{-}0} \rangle$, where

$v_1 = \mathrm{bit}(i_1, \mathrm{loc}_1(\overline{o}_1))$, and

$v_2 = \mathrm{bit}(i_2, \mathrm{loc}_2(\overline{o}_2))$.

- $s = \langle \mathsf{echo}, k, o, q; i_2, v_2, *, t_1; i_1, v_1, *, t_2 \rangle$, where
  $0 \le k \le 2\log n - 1$,
  $v_1 = \mathrm{bit}(i_1, \mathrm{loc}_1(\overline{o}_1))$,
  $v_2 = \mathrm{bit}(i_2, \mathrm{loc}_2(\overline{o}_2))$,
  $t_1 = f(\mathrm{loc}_1(\overline{o}_1))$,
  $t_2 = f(\mathrm{loc}_2(\overline{o}_2))$,
  $o = \mathsf{yes}$ if and only if $b_j = c_j = 0$ for $0 \le j \le k-1$, with b_j and c_j as in the next item, and
  $q = \mathrm{FSA}(b_0 \cdots b_{k-1}, c_0 \cdots c_{k-1})$,
  with $b_0 \cdots b_{k-1}$ and $c_0 \cdots c_{k-1}$ the least significant bits of $\mathrm{loc}_1(\overline{o}_1)$ and $\mathrm{loc}_2(\overline{o}_2)$, respectively.
- $s = \langle \mathsf{test}, *, o, r; *, *, *, t_1; *, *, *, t_2 \rangle$, where
  $o = \mathsf{yes}$ if and only if $\mathrm{loc}_1(\overline{o}_1) = (0,0)$ and $\mathrm{loc}_2(\overline{o}_2) = (0,0)$,
  $r = \mathrm{FSA}(\mathrm{loc}_1(\overline{o}_1), \mathrm{loc}_2(\overline{o}_2))$,
  $t_1 = f(\mathrm{loc}_1(\overline{o}_1))$, and
  $t_2 = f(\mathrm{loc}_2(\overline{o}_2))$.

It can then be shown that the reward for any action prescribed by the policy δ given any of these reachable state/observation sequence combinations is zero given that f is a consistent tiling. □

C Proof of Lemma 2

We need some new notation to carry out this proof. Given a state s, a state component c and corresponding value v from the domain of c, we define the state *"s with c set to v"* (written $s[c := v]$) to be the state s' that agrees with s at all state components except possibly c and has value v for state component c. We also write $\overline{o}_{1,j}$ for the first j observations in the sequence $\overline{o}_1$, and likewise $\overline{o}_{2,j}$ and $\overline{o}'_{2,j}$.

For any state s_j reachable while observing $\langle \overline{o}_{1,j}, \overline{o}_{2,j} \rangle$, we define a state s'_j that we will show is reachable while observing $\langle \overline{o}_{1,j}, \overline{o}'_{2,j} \rangle$, as follows:

$$s'_j = s_j[\mathsf{index\text{-}1} := \mathsf{index\text{-}1}(\overline{o}'_{2,j})][\mathsf{value\text{-}1} := \mathsf{value\text{-}1}(\overline{o}'_{2,j})][\mathsf{tile\text{-}sel\text{-}2} := \mathsf{tile\text{-}sel\text{-}2}(\overline{o}'_{2,j})].$$

We can now show by an induction on sequence length j that for any state s_j such that $\overline{P}_\delta(s_0, \overline{o}_{1,j}, \overline{o}_{2,j}, s_j)$ is nonzero, then $\overline{P}_\delta(s_0, \overline{o}_{1,j}, \overline{o}'_{2,j}, s'_j)$ is also nonzero. From this we can conclude that $\langle \overline{o}_1, \overline{o}'_2 \rangle$ is reachable, as desired.

For the base case of this induction, we take j to be 1, so that the observation sequences involved all have length 1, ending in the generate phase with index equal to zero. Inspection of the definition of the transition probabilities P shows that changing $\mathsf{index\text{-}1}$ and $\mathsf{value\text{-}1}$ arbitrarily has no effect on reachability.

For the inductive case, we suppose some state s_j is reachable by $\langle \overline{o}_{1,j}, \overline{o}_{2,j} \rangle$, and that state s'_j is reachable by $\langle \overline{o}_{1,j}, \overline{o}'_{2,j} \rangle$. Let a_1 be $\delta(\overline{o}_{1,j})$, a_2 be $\delta(\overline{o}_{2,j})$, and a'_2 be $\delta(\overline{o}'_{2,j})$. We must show that for any state s_{j+1} such that $P(s_j, a_1, a_2, s_{j+1})$ is nonzero, $P(s'_j, a_1, a'_2, s'_{j+1})$ is also nonzero, for s'_{j+1}. This follows from the following observations:

- When $\mathsf{phase}(s_j)$ is select or $\mathsf{generate}$, neither agent 2's action a_2 nor the values of $\mathsf{index\text{-}1}(s_j)$ or $\mathsf{value\text{-}1}(s_j)$ have any affect on $P(s_j, a_1, a_2, s_{j+1})$ being nonzero, as long as either $\mathsf{index\text{-}1}$ is not equal to j or $\mathsf{pos\text{-}bit\text{-}1}(s_{j+1})$ equals $\mathsf{value\text{-}1}(s_j)$. However, this last condition is ensured to hold of the $\mathsf{index\text{-}1}$ and $\mathsf{value\text{-}1}$ components of s_j and s'_j by the compatibility of $\overline{o}'_2$ with $\overline{o}_1$.

- When $\mathsf{phase}(s_j)$ is query, the action a'_2 must equal the $\mathsf{tile\text{-}sel\text{-}2}$ state component of s'_{j+1} by the definitions of s'_{j+1} and "index variant," and changes to the $\mathsf{index\text{-}1}$ and $\mathsf{value\text{-}1}$ components have no effect on $P(s_j, a_1, a_2, s_{j+1})$ being nonzero during the query phase.

- When $\mathsf{phase}(s_j)$ is echo, the actions a_2 and a'_2 must be a faithful echo of the location address bit indicated by the index state component (since we have assumed as part of out inductive hypothesis that δ is faithful during echo for at least j bits), and this bit's value does not vary between $\overline{o}_2$ and $\overline{o}'_2$ because if the observation sequences reach the echo phase we have the assumption that these are same-address variants. Thus $a_2 = a'_2$ during echo. Again, changes to the $\mathsf{index\text{-}1}$ and $\mathsf{value\text{-}1}$ components have no effect on $P(s_j, a_1, a_2, s_{j+1})$ being nonzero during the echo phase.

□

Acknowledgment

The authors thank Micah Adler, Andy Barto, Judy Goldsmith, Dexter Kozen, Victor Lesser, Frank McSherry, Ted Perkins, Bob Sloan, and Ping Xuan for helpful discussions. This work was supported in part by the National Science Foundation under grants IRI-9624992 and IRI-9634938 to Shlomo Zilberstein, grant CCR-9877078 to Neil Immerman, grants 9977981-IISI and 0093100-IIS to Robert Givan, and an NSF Graduate Fellowship to Daniel Bernstein.

References

Babai, L., Fortnow, L. & Lund, C. (1991). Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, *1*, 3–40.

Blondel, V. D. & Tsitsiklis, J. N. (2000). A survey of computational complexity results in systems and control. *Automatica*, *36(9)*, 1249–1274.

Cassandra, A., Littman, M. L. & Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence* (pp. 54–61).

Coradeschi, S., Karlsson, L., Stone, P., Balch, T., Kraetzschmar, G. & Asada, M. (2000). Overview of RoboCup-99. *AI Magazine*, *21(3)*, 11–18.

Hansen, E. (1998). Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence* (pp. 211–219).

Hsu, K. & Marcus, S. I. (1982). Decentralized control of finite state Markov processes. *IEEE Transactions on Automatic Control*, *AC-27(2)*, 426–431.

Jaakkola, T., Singh, S. P. & Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Proceedings of Advances in Neural Information Processing Systems 7* (pp. 345–352).

Kaelbling, L. P., Littman, M. L. & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101(1-2)*, 99–134.

Lewis, H. (1978). Complexity of solvable cases of the decision problem for predicate calculus. In *Proceedings of the Nineteenth Symposium on the Foundations of Computer Science* (pp. 35–47).

Lusena, C., Goldsmith, J., Li, T., Sittinger, S. & Wells, C. (1999). My brain is full: When more memory helps. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 374–381).

Madani, O., Hanks, S. & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (pp. 541–548).

Meuleau, N., Kim, K.-E., Kaelbling, L. & Cassandra, A. R. (1999). Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 417–426).

Mundhenk, M., Goldsmith, J., Lusena, C. & Allender, E. (2000). Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, *47(4)*, 681–720.

Ooi, J. M. & Wornell, G. W. (1996). Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proceedings of the 35th Conference on Decision and Control* (pp. 293–298).

Papadimitriou, C. H. (1994). *Computational Complexity.* Reading, MA: Addison-Wesley.

Papadimitriou, C. H. & Tsitsiklis, J. (1982). On the complexity of designing distributed protocols. *Information and Control*, *53*, 211–218.

Papadimitriou, C. H. & Tsitsiklis, J. (1986). Intractable problems in control theory. *SIAM Journal of Control and Optimization*, *24(4)*, 639–654.

Papadimitriou, C. H. & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12(3)*, 441–450.

Peshkin, L., Kim, K.-E., Meuleau, N. & Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence* (pp. 489–496).

Peterson, G. L. & Reif, J. R. (1979). Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science* (pp. 348–363).

Puterman, M. L. (1994). *Markov Decision Processes.* New York: J Wiley & Sons.

Schneider, J., Wong, W.-K., Moore, A. & Riedmiller, M. (1999). Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 371–378).

Zhang, W. (2001). *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Hong Kong University of Science and Technology, Kowloon, Hong Kong.