Complexity of Decentralized Control: Special Cases (*Supplemental Materials*)

Martin Allen
Department of Computer Science
Connecticut College
New London, CT 06320
martin.allen@conncoll.edu

Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
shlomo@cs.umass.edu

Abstract

The worst-case complexity of general decentralized POMDPs, which are equivalent to partially observable stochastic games (POSGs) is very high, both for the cooperative and competitive cases. Some reductions in complexity have been achieved by exploiting independence relations in some models. We show that these results are somewhat limited: when these independence assumptions are relaxed in very small ways, complexity returns to that of the general case. (*Here, we present background material and detailed proofs omitted from our main paper.*)

1 Bernstein's proof of NEXP-completeness

Before establishing our new claims, we review the proof of NEXP-completeness for finite-horizon Dec-MDPs, as given by Bernstein et al. [1, 2] (we draw here on perhaps the most complete version of the proof, from Bernstein's thesis [3]).

First, we note that the upper bound—namely that finite-horizon Dec-POMDPs are in NEXP, meaning that they can actually be solved in nondeterministic exponential time—will immediately and without modification establish the same upper bound for all the problems we will consider, since each is a special sub-case of the general finite-horizon framework. As we show in Proposition 3, any Dec-POMDP has an equivalent factored form. Thus, Bernstein's upper bound immediately establishes the identical bound on the complexity of any factored finite-horizon Dec-POMDP; since all the problems we will look at are but special restricted cases (like the transition- and observation-independent Dec-MDPs with shared rewards), the upper bound applies to all of them as well.[1]

Theorem 1 (Upper Bound)**.** The decision version of the finite-horizon, n-agent problem Dec-POMDP $\in$ NEXP.

Proof (after Theorem 4, Bernstein [3]). Consider arbitrary finite-horizon, n-agent Dec-POMDP $\mathcal{D}$, constant k, and relevant joint policy π. We show that we can evaluate whether the value of π is at least k in exponential time. Policy π will, by definition, consist of n mappings from agents' observation-histories to actions. Any Dec-POMDP, taken together with such a joint policy, can be treated as a single-agent POMDP with individual policy, by treating each n-tuple of observations as a single, complex observation, and likewise for joint action-tuples. In exponential time, each of the exponentially-many possible sequences of such complex observations can be converted into a belief-state over possible states of the underlying system. Similarly, state-transitions and expected

[1] Note that, as is standard in such complexity proofs, results are stated in terms of the "*decision version*" of the problem. For the class of Dec-POMDPs, for instance, the decision version is the problem of determining, for a given Dec-POMDP $\mathcal{D}$, and constant value k, whether there exists any policy for $\mathcal{D}$ with value at least k. This suffices to establish the complexity of the optimal-policy problem, since any method that could produce an optimal policy could answer the decision question in the affirmative for suitably chosen values of k.

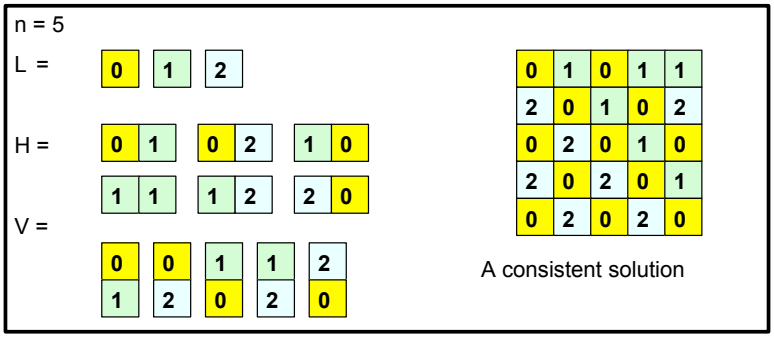

Figure 1: An example of the TILING problem, and a consistent solution.

rewards over belief-states can be calculated in exponential time, turning the problem into a belief-state MDP, with state-space exponential in the size of the original Dec-POMDP. Finally, dynamic programming can be used to evaluate policy π over the MDP, in time that is polynomial in the size of the new state-space (or exponential overall), to determine whether its value is at least k. □

Since this result gives us an upper bound on the complexity of each problem-type we examine, we will simply refer back to it as needed. More challenging (and more interesting) is the process of establishing lower bounds on these problems. That is, in each case, we shall want to show that the problems are in fact *NEXP-hard*, and actually require nondeterministic exponential time to solve. The theoretical tool that allows us to establish this result is the *reduction*, whereby any instance of a problem already known to have that complexity is transformed (without expanding the size of the problem representation beyond a polynomial factor) into an instance of the new problem of interest. If it can then be shown that a solution to the latter form of the problem is in fact a solution to the former, we have established that the latter form is no easier than the original. Furthermore, if the problem that is reduced is known to be *complete* for its complexity class—meaning that any problem in that class can be reduced to it—then the new problem class is also complete for the class, since reduction is a transitive operation.

1.1 The TILING problem

The problem used in our reductions is the NEXP-complete TILING problem (see Lewis [4], Papadimitriou [5]). A TILING problem instance consists of a board size n, given concisely in $\log n$ binary bits, a set of tile-types $L = \{t_0, \ldots, t_k\}$, and a collection of binary and vertical compatibility relations between tiles H, $V \subseteq L \times L$. A *tiling* is a mapping of board locations to tile-types, $t : \{0, \ldots, n-1\} \times \{0, \ldots, n-1\} \rightarrow L$; such a tiling is *consistent* just in case (i) the origin location of the board receives tile-type 0 ($t(0,0) = tile_0$); and (ii) all adjoint tile assignments are compatible:

$$(\forall x, y)\ \langle t(x,y), t(x+1,y)\rangle \in H\ \ \&\ \ \langle t(x,y), t(x,y+1)\rangle \in V.$$

The TILING problem is thus to decide, for a given instance, whether such a consistent tiling exists. Figure 1 shows an example instance and consistent solution.

In what follows, we sketch the details of the reduction of TILING to a 2-agent Dec-MDP (this will establish the hardness result for the more general Dec-POMDP case). A key detail to keep in mind going forward is that such a reduction must not increase the size of the original TILING problem beyond a polynomial factor. Since the TILING instance is given with the board size n represented logarithmically, it will be necessary that any reductions likewise produce problems with size that is $\mathcal{O}(\log n)$. (For the purposes of proof, we can presume that n is suitably large, so that the description of the set of tile-types and compatibility relations is also possible in size logarithmic in n.)

1.2 The basic reduction

The essential idea behind the reduction is to begin with some arbitrary instance of TILING, and create a finite-horizon, 2-agent Dec-MDP such that the latter has a policy with non-negative value

if and only if the former has a consistent tiling. Thus, we will have shown that the problem of determining whether a Dec-MDP has a policy with value at least $k = 0$ is as hard as TILING, which, together with the upper bound result, will establish NEXP-completeness.

To do this, the Dec-MDP is designed to choose two random tile positions from the board, which are then revealed separately to each agent, bit-by-bit. (The state-set of the Dec-MDP can not directly include the chosen positions, else it would have a state-set of impermissible size $\mathcal{O}(n)$, i.e., exponential in $\log n$.) Following the revelation of the tile locations, each agent chooses a tile to place there. The agents are then required to repeat the bits they have seen back to the system, which calculates the relative locations of the tiles, and checks them for compatibility as necessary. In order to ensure that the agents are honest in repeating the bits they have seen, the system takes certain precautions to remember one such bit for each agent, without letting the agent know the precise one that is being remembered. Deviations from the remembered bit, or tile-types that are incompatible, lead to negative reward; therefore, an optimal (0-value) policy is only possible if the agents already possess a consistent solution for the original TILING problem, i.e., such a tiling must exist.

To accomplish this, the reduction thus creates a Dec-MDP with dynamics that work in the following 5 phases.

Stage 1: Select. The system randomly chooses two possible indices, and one bit-value at each of those indices, to encode one bit of each of the grid-locations given to the two agents. Each of these is remembered for later.

Stage 2: Generate. Under the constraints given by the bits chosen in the prior phase, two otherwise-random locations in the TILING grid, revealing one to each agent, bit-by-bit.

Stage 3: Query. Each agent supplies a tile-type, which is remembered for later.

Stage 4: Echo. The agents repeat back the bits they have seen, allowing the system to calculate the relative positions of the two tiles. The location-bits remembered from the **Select** phase are used to keep the agents honest.

Stage 5: Test Having calculated the location of the two tiles, they are evaluated to see whether they are consistent, according to the given TILING problem.

We give more detail of this reduction below.

1.3 The proof of correctness

In later sections, we present modifications of Bernstein's reduction, extending the proof technique to new sub-classes of the Dec-MDP framework. Thus, in the next section, we go over the nature of his reduction in detail, in order to make the extent of our own alterations clear. Before doing so, however, we outline Bernstein's *correctness results*, which demonstrate that his reduced problem has a policy with non-negative value if and only if the original TILING instance has a consistent solution. Like his upper-bound result (Theorem 1), we will rely upon these claims in our own proofs; we therefore describe their salient features here, so that we may refer back to them.

Proposition 1. If a consistent solution to the arbitrary TILING problem given exists, then there exists a policy for the two agents with non-negative expected value.

Sketch of proof (after Lemma 1, Bernstein [3]). In the reduced problem, it is easy to show that the sole policy with non-negative expected value involves two main elements: (1) during the **Query** phase, providing a tile-type for the location provided that is identical with that in an agreed-upon consistent tiling; and (2) during the **Echo** phase, faithfully repeating the observed location-bits, so that the system can calculate the compatibility relationships accurately. □

In our own reductions, we will rely upon this result, arguing that the sole policy with non-negative expected value must have the same features. Thus, we can use Bernstein's proof directly.

Proposition 2. The existence of a policy with non-negative value for the reduced problem implies the existence of a consistent solution to the TILING instance given.

Sketch of proof (after Claims 1–3, Bernstein [3]). In the reduced problem, if either agent deviates from the policy that faithfully echoes the location-bits seen during the earlier **Generate** phase of the

problem, then there will exist observation sequences for which they have some positive probability of providing an incompatible pair of tile-types. That is, since the agents do not know which bit of the location provided them is being memorized by the system, they have no way of "gaming the system" reliably. In particular, they have no way of knowing whether the system has revealed the same tile location (or adjacent locations) to each agent, and cannot avoid the possibility that they are providing an incompatible pair of tile-types, leading to negative expected reward in the final **Test** phase. Only by following the strictly faithful policy can they avoid this possibility, and thus they must provide a consistent tiling. □

As for the previous proposition, we will rely upon Bernstein's proof directly. Our own reductions will also require that agents provide faithful echoes of the locations given, and choose compatible tile-types at any such location. The main differences will be in the specific structure of the reduced problem, not in the nature of the non-negative policies for same.

Together with Theorem 1, and the fact that the finite-horizon, 2-agent Dec-MDP is a special case of the general finite-horizon Dec-POMDP, these propositions establish Bernstein's main complexity result:

Theorem 2 (NEXP-Completeness)**.** The finite-horizon Dec-POMDP problem is NEXP-complete.

In this connection, we note that the infinite-horizon version of the problem is undecidable, a fact that follows directly from the undecidable nature of infinite-horizon, single-agent POMDPs (Madani, et al. [6]).

1.4 Details of the reduction: state space

We now outline the specifics of the Dec-MDP problem used in the reduction result. Given a TILING instance $\langle n, L, H, V \rangle$, we create a finite-horizon, 2-agent Dec-MDP as follows. A key element of the model derives from the fact that in a Dec-MDP, unlike a Dec-POMDP, we must be able to determine the global state, once we know the observations of each agent. This, along with the need to ensure that neither agent can observe the remembered bit that the system uses to keep them honest during the **Query** phase, leads us to divide the set of variables defining a given state into 3 parts, according to which of the agents observes them.

Both Agents. Four (4) variables are observed at each step. First, there is variable $phase \in \{sel, gen, query, echo, test\}$, indicating the current phase of the process. Second, there is variable $index \in \{0, \ldots, 2 \log n\}$, indicating the next (x, y)-location bit to be generated or echoed back. Third, there is $origin \in T, F$, indicating whether or not we are at the origin point $(0, 0)$ of the tiling grid. Lastly, there is $q \in Q$, tracking the state of a finite-state automaton that calculates the relative position of the locations echoed back by each agent (as will be described briefly below, we ensure that $|Q| = \mathcal{O}(\log n)$, as required).

Agent 1. This agent observes four (4) variables of its own, as well. First, it observes the index and value of the location bit that the system remembers for the *other agent*, $index_2 \in \{0, \ldots, 2 \log n - 1\}$ and $value_2 \in \{0, 1\}$. Second, it observes the current position-bit used in the **Generate** phase to provide a tiling-location, $pos_1 \in \{0, 1\}$, and its own choice of a tile-type $tile_1 \in L$.

Agent 2. This agent likewise observes (4) variables of its own, each of which is directly analogous to those observed by the other agent: $index_1$, $value_1$, pos_2, and $tile_2$.

The set of all states, S, is thus any combination of these variables, and thus $|S| = \mathcal{O}(\log n)$; we write any such state as a tuple of possible variable values, using a semicolon to separate each of the three classes of observed variables, such as:

$$s = \langle query, 5, F, q_0; 3, 1, 0, t_3; 4, 0, 1, t_5 \rangle$$

As a convention, we can abstract over a set of variables by only partially specifying their values, using "$*$" where we are indifferent. For instance, we can identify the set of all variables during the **Echo** phase, writing $s \in \langle echo, *, *, *; *, *, *, *; *, *, *, * \rangle$ to indicate such a state.

The Dec-MDP begins in the **Select** phase, with neither agent having chosen a tile-type, and the FSA at its initial state: $s_0 = \langle sel, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle$

1.5 Details of the reduction: action transitions

Each agent possesses two possible actions for repeating bits back to the system, and an action for selecting each possible tile-type; therefore, $A_1 = A_2 = \{0, 1\} \cup L$.[2] The state-action transitions are specified as follows, for each phase of the problem dynamics (any combinations not covered by the description are not reachable in the problem, and can be treated arbitrarily).

Phase 1: Select. The process selects one bit of the location to be revealed to each agent uniformly at random, and reveals it solely to the other agent.

$$\begin{aligned} P(s, a_1, a_2, s') = \frac{1}{(4 \log n)^2} \text{ whenever:} \\ s = s_0 = \langle sel, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle, \\ s' = \langle gen, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle, \\ i_1,\, i_2 \in \{0, \ldots, (2 \log n) - 1\}, \text{ and } v_1,\, v_2 \in \{0, 1\}. \end{aligned}$$

Phase 2: Generate. Two tile positions are chosen and revealed to each agent, bit-by-bit; this is done randomly, under the constraint that the bit chosen in the prior phase (**Select**) must be honored. After the entire bit-sequence has been revealed to each agent, the problem transitions deterministically to the next phase (**Query**).

$$\begin{aligned} P(s, a_1, a_2, s') = \frac{1}{h} \text{ whenever:} \\ s = \langle gen, k, T, q_0; i_2, v_2, *, t_0; i_1, v_1, *, t_0 \rangle \quad (0 \leq k < 2 \log n), \\ s' = \langle gen, k+1, T, q_0; i_2, v_2, b_1, t_0; i_1, v_1, b_2, t_0 \rangle, \text{ where} \\ b_1 = v_1 \text{ if } k = i_1, \text{ else } b_1 \in \{0, 1\}, \\ b_2 = v_2 \text{ if } k = i_2, \text{ else } b_2 \in \{0, 1\}, \\ h \text{ is the number of allowed combinations of } b_1 \text{ and } b_2. \\ P(s, a_1, a_2, s') = 1 \text{ whenever:} \\ s = \langle gen, 2 \log n, T, q_0; i_2, v_2, *, t_0; i_1, v_1, *, t_0 \rangle, \\ s' = \langle query, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle. \end{aligned}$$

Phase 3: Query. This is a one-step phase, in which each agent chooses a tile-type, and we transition to the **Echo** phase.

$$\begin{aligned} P(s, a_1, a_2, s') = 1 \text{ whenever:} \\ s = \langle query, 0, T, q_0; i_2, v_2, 0, t_0; i_1, v_1, 0, t_0 \rangle, \\ s' = \langle echo, 0, T, q_0; i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle, \text{ where} \\ \mathbf{t_i} = \begin{cases} a_i & \text{if } a_i \in L \\ t_0 & \text{else.} \end{cases} \end{aligned}$$

Phase 4: Echo. The system now has the agents repeat back the location-bits that they previously observed. Using these bits, a finite-state automaton calculates the relative positions of the tile-locations. For any state $q \in Q$ of the FSA, and two echoed bits b_1, b_2, we write $FSA(q, b_1, b_2)$ for the output of the automaton at that stage. Full details of the FSA construction can be found in Bernstein [3, §3.4.4]; here, we simply note again the important fact that it can be constructed within the proper space bounds, $|Q| = \mathcal{O}(\log n)$. Once all the bits have been echoed back, the system

[2] Here, we simplify Bernstein's proof somewhat. In the original, agents also possessed a "wait" action, allowing them to do nothing during some phases of the problem. However, as we will see, all state-transitions during such "idle" phases are independent of any actions taken, so no separate waiting action is required (the agents can choose any of their other possible actions whatsoever with no effect on possible outcomes).

transitions to the final phase (**Test**).

$$
\begin{aligned}
P(s, a_1, a_2, s') = 1 \text{ whenever:} \\
s &= \langle echo, k, o, q; i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle, \\
s' &= \langle p, k', o', FSA(q, b_1, b_2); i_2, v_2, 0, \mathbf{t_1}; i_1, v_1, 0, \mathbf{t_2} \rangle, \text{ where} \\
b_i &= \begin{cases} 1 & \text{if } a_i = 1 \\ 0 & \text{else} \end{cases}, \quad p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases}, \\
o' &= T \Leftrightarrow (o = T \;\&\; a_1 = a_2 = 0).
\end{aligned}
$$

Phase 5: Test. The final phase is a single-step process in which the system terminates.

$$
\begin{aligned}
P(s, a_1, a_2, s') = 1 \text{ whenever:} \\
s &= \langle test, 0, *, *; *, *, 0, *; *, *, 0, * \rangle, \\
s' &= \langle test, 0, T, q_0; 0, 0, 0, t_0; 0, 0, 0, t_0 \rangle.
\end{aligned}
$$

1.6 Details of the reduction: rewards

The Reward function for the Dec-MDP is simple: agents receive reward of -1 for any state-action pair, except for a special designated set, for which they receive reward of 0. This set corresponds to the key features of the desired policy: faithfully repeating the bit that the system has remembered from their location during the **Echo** phase, and ending up with a compatible pair of tiles in the **Test** phase.[3] This last reward depends upon the final state of the FSA that calculates relative tile-locations; it will be drawn from the set $\{\mathbf{equal}, \mathbf{hor}, \mathbf{ver}, \mathbf{apart}\}$, as the locations are respectively identical, horizontally adjacent, vertically adjacent, or otherwise.

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

Select: $s = \langle sel, *, *, *; *, *, *, *; *, *, *, * \rangle$.

Generate: $s = \langle gen, *, *, *; *, *, *, *; *, *, *, * \rangle$.

Query: $s = \langle query, *, *, *; *, *, *, *; *, *, *, * \rangle$.

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

Test (i): $s = \langle test, *, o, \mathbf{equal}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_1} \rangle$,
$o = F$ or $\mathbf{t_1} = t_0$.

Test (ii): $s = \langle test, *, *, \mathbf{hor}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_2} \rangle$,
$\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$.

Test (iii): $s = \langle test, *, *, \mathbf{ver}; *, *, *, \mathbf{t_1}; *, *, *, \mathbf{t_2} \rangle$,
$\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$.

Test (iv): $s = \langle test, *, *, \mathbf{apart}; *, *, *, *; *, *, *, * \rangle$.

1.7 Details of the reduction: observations and time horizon

Finally, we simply set the observation function for each agent to be deterministic: given a state, agent α_i observes the four variables that make up its portion of the state space, along with the global variables that make up the first portion. The observation-set for α_1 is thus $\Omega_1 = \{phase, index, origin, Q, index_2, value_2, pos_1, tile_1\}$, and similarly for α_2. The time horizon is $T = (4 \log n) + 4$, consisting of one step each for the **Select**, **Query**, and **Test** phases, $(2 \log n) + 1$ steps to **Generate** the bits of the pair of (x, y) locations, and $2 \log n$ steps for the agents to **Echo** these bits back.

[3] Again, we simplify Bernstein somewhat: since we have no need of a "wait" action, and since the state-transitions have the effect of choosing a default tile-type (t_0) or bit (0) should the agents use an inappropriate action at the relevant phases, we can allow agents to act without penalty in any fashion they choose during the "idle" phases, where transitions do not depend upon those actions.

1.8 Discussion of the reduction

It is intuitively easy to see that the Dec-MDP dynamics force agents to faithfully repeat the bits they have seen, and to choose tile-types from a known consistent tiling, if they are to expect non-negative reward. In the **Select** and **Generate** phase, the agents are idle, as no actions affect the system dynamics, and the only variables that change are those related to the initial selection of bits, and the revelation of their own tile-location. This has three important effects. First, since the transitions are otherwise random, the agents can not know, given what they have observed, what bit of their own location is being memorized by the state (recall, they know only the bit for the *other agent*). Second, since they can not affect the state during these phases, no action can serve as a signal, allowing them to communicate any revealed bits to one another, explicitly or implicitly.

Finally, since the FSA state-variable Q—observed by both—does not change at all in these phases, and is only affected by the bits echoed back later, agents have no information about the relative positions of the locations revealed, and no choice but to use the known consistent tiling when picking a tile-type during **Query**, lest they risk a negative reward later on. Further, during **Echo**, since agents still can not know which of their own location-bits has been memorized, the only policy guaranteed not to produce a negative, incompatible-tiling situation is the one that repeats the bits back exactly as given. Any attempt to lie about these bits, even based upon interim FSA states, risks negative reward. Thus, Bernstein's reduction ingeniously forces any policy with non-negative value to correspond directly to a consistent tiling, as required. We will make use of similar reductions in our next results.

2 Factored Dec-MDPs and independence

We begin with the basic formal definitions of this special sub-class of Dec-MDP, adapted from existing literature, and review the NP-completeness claim.

Definition 1 (Factored Dec-POMDP)**.** A factored, n-agent Dec-POMDP is a Dec-POMDP such that the system state can be factored into $n+1$ distinct components, so that $S = S_0 \times S_1 \times \cdots \times S_n$, and no state-variable appears in any S_i, S_j, $i \neq j$.

As with the *local* (agent-specific) actions, a_i, and observations, o_i, in the general Dec-POMDP definition, we now refer to the *local state*, $\hat{s} \in S_i \times S_0$, namely that portion of the overall state-space that is either specific to agent α_i ($s_i \in S_i$), or shared among all agents ($s_o \in S_0$). Note that we insist that the agent-specific portions of local states be non-overlapping and distinct portions of the remaining state-space; however, this restriction is not very limiting, as we can easily show.

Proposition 3. Any n-agent Dec-POMDP has an equivalent factored version.

Proof. Consider an arbitrary n-agent Dec-POMDP:

$$\mathcal{D} = \langle \{\alpha_i\},\, S,\, \{A_i\},\, P,\, \{\Omega_i\},\, O,\, R,\, T \rangle.$$

We define a factored version of the problem

$$\mathcal{D}^f = \langle \{\alpha_i\},\, S^f,\, \{A_i\},\, P^f,\, \{\Omega_i\},\, O^f,\, R^f,\, T \rangle,$$

with components:

- $\{\alpha_i\}$, $\{A_i\}$, $\{\Omega_i\}$, T are identical between $\mathcal{D}$ and $\mathcal{D}^f$.
- We add n state variables, $v_1, \ldots, v_n$, each of which has a single constant value, $v_i = 1$, $\forall i$. For each agent α_i, the purely local portion of the state space is $S_i = \{v_i\}$, and we let $S_0 = S \in \mathcal{D}$, the original state-space. Every state $s^f \in (S^f = S_0 \times S_1 \times \cdots \times S_n)$ is then simply some state $s \in S \in \mathcal{D}$, with n 1's appended; for any state s in the original state-space, we thus write $s1^n$ for the corresponding state in the new state-space. Note that $|S| = \left|S^f\right|$.
- $\forall s1^n,\, s'1n,\, \forall a_1, \ldots, a_n,\, \forall o_1, \ldots, o_n,$
  1. $P^f(s1^n, a_1, \ldots, a_n, s'1^n) = P(s, a_1, \ldots, a_n, s')$.
  2. $O^f(\forall a_1, \ldots, a_n, s'1^n, o_1, \ldots, o_n) = O(\forall a_1, \ldots, a_n, s', o_1, \ldots, o_n)$.
  3. $R^f(s1^n, a_1, \ldots, a_n) = R(s, a_1, \ldots, a_n)$.

It is easy to see that the new Dec-POMDP, $\mathcal{D}^f$ is equivalent to the original, since all state-transitions, observations, and rewards are based on precisely the same state-variables in both problems. It is also worth noting that the proof establishes the same result for Dec-MDPs. If the original problem $\mathcal{D}$ is a Dec-MDP, then any observation sequence determines the global state s; thus, in the derived version $\mathcal{D}^f$, the observation sequence for $s1^n$ will also determine s, and the rest of the state is fixed. $\square$

It is more interesting, however, to consider problems with system dynamics that actually break down separately and meaningfully over the various local states. In doing so, for convenience of presentation, we will use the notation $\overline{s}_{-i}$ for the sequence of all state-components *except that* for agent α_i:

$$\overline{s}_{-i} = (s_0,\, s_1,\, \ldots,\, s_{i-1},\, s_{i+1},\, \ldots,\, s_n)$$

(and similarly for action- or observation-sequences, $\overline{a}_{-i}$ and $\overline{o}_{-i}$).

Definition 2 (Transition Independence)**.** A factored, n-agent DEC-POMDP is *transition independent* if and only if the state-transition function can be separated into $n+1$ distinct transition functions $P_0,\, \ldots,\, P_n$, where, for any next state $s'_i \in S_i$,

$$P(s'_i \mid (s_0, \ldots, s_n), (a_1, \ldots, a_n), \overline{s}_{-i}) = \begin{cases} P_0(s'_0 \mid s_0) & \text{if } i = 0; \\ P_i(s'_i \mid \hat{s}_i, a_i, s'_0) & \text{else.} \end{cases}$$

In other words, the next local state of each agent is independent of the local states of all others, given its previous local state and local action, and the external system features (S_0). Further, the external features evolve independently of all local agent actions or states. Given these independence relations, the joint probability distribution over global states is a product of the relevant individual distributions.

Definition 3 (Observation Independence)**.** A factored, n-agent Dec-POMDP is *observation independent* if and only if the joint observation function can be separated into n separate probability functions $O_1,\, \ldots,\, O_n$, where, for any local observation $o_i \in \Omega_i$,

$$O(o_i \mid (a_1, \ldots, a_n), (s'_0, \ldots, s'_n), \overline{o}_{-i}) = O_i(o_i \mid a_i, \hat{s}'_i)$$

In such cases, the probability of each agent's individual observations is a function of their own local states and actions alone, independent of the states of other agents, and of what those others do or observe. As with transition probabilities, the conditional independence given by this definition implies that the joint observation function O can be represented as a product of the individual-agent observation functions. Finally, we can extend the notion of independence to the Dec-POMDP reward function, separating it into distinct local functions.

Definition 4 (Reward Independence)**.** A factored, n-agent Dec-POMDP is *reward independent* if and only if the joint reward function can be represented using a set of local reward functions $R_1, \ldots, R_n$, such that:

$$R((s_0, \ldots s_n), (a_0, \ldots, a_n)) = f(R_1(\hat{s}_1, a_1), \ldots, R_n(\hat{s}_n, a_n))$$

and

$$R_i(\hat{s}_i, a_i) \geq R_i(\hat{s}_i,\, a'_i) \;\Leftrightarrow\; f(R_1, \ldots, R_i(\hat{s}_i, a_i), \ldots, R_n) \geq f(R_1, \ldots, R_i(\hat{s}_i, a'_i), \ldots, R_n)$$

That is, the joint reward function can be represented as a function over local reward functions alone, under the constraint that we maximize the global reward if and only if we maximize each of the local rewards. A typical example of such an independent factorization is the additive sum of local rewards,

$$R((s_0, \ldots s_n), (a_0, \ldots, a_n)) = R_1(\hat{s}_1, a_1) + \cdots + R_n(\hat{s}_n, a_n),$$

where the system simply receives the total reward gathered by individual agents.

It is important to note that each of the three definitions of independence applies equally to Dec-MDPs, just as well as Dec-POMDPs. (The reader will recall that a Dec-MDP is a Dec-POMDP that is *jointly fully observable*, so that the global state is determined by the current tuple of agent-observations.) When we are dealing with the latter type of problem, we are often interested in instances for which the joint full observability of the overall state is accompanied by full observability at the local level.

Definition 5 (Local Full Observability)**.** A factored, n-agent Dec-MDP is *locally fully observable* if and only if any agent's local observation uniquely determines its local state: $\forall o_i \in \Omega_i, \exists \hat{s}_i : P(\hat{s}_i \mid o_i) = 1$.

Local full observability is not simply equivalent to independence of observations. In particular, a problem may be locally fully observable without being observation independent (since agents may simply observe the results of non-independent joint actions). On the other hand, it is easy to show that an observation-independent Dec-MDP must also be locally fully observable.[4]

Proposition 4. If any factored, n-agent Dec-MDP is observation independent, then it is locally fully observable as well.

Proof. Suppose an arbitrary factored, n-agent Dec-MDP $\mathcal{D}$ is observation independent. Assume, for contradiction, that it is not locally fully observable. By Definition 5, this means that there exists some agent α_i, and some observation $o_i \in \Omega_i$, for which there is no local state $\hat{s}_i$ such that $P(\hat{s}_i \mid o_i) = 1$. Let $o_i^\star$ be that observation. Now, since the observation function for agent α_i, O_i, must define a proper probability distribution (and so it cannot be that all states $\hat{s}_i$ have 0 probability given o_i), there must exist at least two local states with positive probability given the observation in question; that is, there exist $\hat{\mathbf{s}}_i^1, \hat{\mathbf{s}}_i^2$ such that:

1. $\hat{\mathbf{s}}_i^1 \neq \hat{\mathbf{s}}_i^2$;
2. $P(\hat{\mathbf{s}}_i^1 \mid o_i^\star) = k$ and $P(\hat{\mathbf{s}}_i^2 \mid o_i^\star) = m$, with $0 \lneq k, m \lneq 1$.

By Bayes' Rule, and conditionalization:

$$P(\hat{\mathbf{s}}_i^1 \mid o_i^\star) = \frac{P(o_i^\star \mid \hat{\mathbf{s}}_i^1)\, P(\hat{\mathbf{s}}_i^1)}{P(o_i^\star)}$$

$$= \frac{\sum_{a_i} P(o_i^\star \mid a_i, \hat{\mathbf{s}}_i^1)\, P(a_i \mid \hat{\mathbf{s}}_i^1)\, P(\hat{\mathbf{s}}_i^1)}{P(o_i^\star)}$$

and therefore, since $P(\hat{\mathbf{s}}_i^1 \mid o_i^\star) \neq 0$, we know that there exists some action, call it $\mathbf{a}_i^1$, such that $P(o_i^\star \mid \mathbf{a}_i^1, \hat{\mathbf{s}}_i^1) \neq 0$. By similar reasoning, there exists some action, call it $\mathbf{a}_i^2$, such that $P(o_i^\star \mid \mathbf{a}_i^2, \hat{\mathbf{s}}_i^2) \neq 0$. Now, by Definition 3, since $\mathcal{D}$ is observation independent, we can factor the observation function:

$$O((a_1, \ldots, a_n), (s_0, \ldots, s_n), (o_1, \ldots, o_n)) = \prod_{1 \leq i \leq n} O_i(a_i, \hat{s}_i, o_i)$$

Now consider any combination of observation-function values with non-zero value, featuring designated components for agent α_i, and otherwise identical:

$$O_1(a_1, \hat{s}_1, o_1) \times \cdots \times O_i(\mathbf{a}_i^1, \hat{\mathbf{s}}_i^1, o_i^\star) \times \cdots \times O_n(a_n, \hat{s}_n, o_n) \neq 0 \quad (1)$$

$$O_1(a_1, \hat{s}_1, o_1) \times \cdots \times O_i(\mathbf{a}_i^2, \hat{\mathbf{s}}_i^2, o_i^\star) \times \cdots \times O_n(a_n, \hat{s}_n, o_n) \neq 0 \quad (2)$$

(Such a combination must exist, trivially.) Since $\mathcal{D}$ satisfies the definition of a Dec-MDP, then by Equation (1) there exists a function J from observation-tuples to states such that: $J(o_1, \ldots, o_i^\star, \ldots, o_n) = (\hat{s}_1, \ldots, \hat{\mathbf{s}}_i^1, \ldots, \hat{s}_n)$. Similarly, by Equation (2), $J(o_1, \ldots, o_i^\star, \ldots, o_n) = (\hat{s}_1, \ldots, \hat{\mathbf{s}}_i^2, \ldots, \hat{s}_n)$. However, since $\hat{\mathbf{s}}_i^1 \neq \hat{\mathbf{s}}_i^2$, and J is a function, this is impossible. Therefore, by *reductio*, $\mathcal{D}$ must be locally fully observable. □

3 Shared reward structures

As just mentioned, if a Dec-MDP (or Dec-POMDP) has all three forms of independence given by Definitions 2–4, it can be decomposed straightforwardly into n separate problems, where each agent

[4] A version of this result was originally proven by Goldman and Zilberstein [7], for a somewhat more complicated case, and required the additional assumption that the Dec-MDP was transition-independent as well. Our result is new, and the proof distinct.

α_i works solely within the sub-environment given by local states in $S_i \times S_0$. Such single-agent problems are known to be P-complete, and can be generally solved to high degrees of optimality in efficient fashion.[5] However, it is possible to create much more complex problem instances when nearly all of the independence relationships hold. In particular, it has been shown that Dec-MDPs that are both transition- and observation-independent, but do not separate their reward-structure, are NP-complete.

We review this result, found in Goldman and Zilberstein [7] and Becker et al. [10]. To do so, we first describe the *event-based* reward structure used in that work. While somewhat complicated, these ideas are of broader use to us. Later sections will also make use of these definitions to show how other combinations of independent dynamics relate to complexity, and how the nature of the reward structure relates precisely to the essential dimensionality of problem instances. Again, we give all definitions in terms of Dec-POMDPs; they apply immediately to Dec-MDPs in particular.

Definition 6 (History)**.** A *history* for an agent α_i in a factored, n-agent Dec-POMDP $\mathcal{D}$ is a sequence of possible local states and actions, beginning in the agent's initial state: $\Phi_i = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, \ldots]$. When a problem has a finite time-horizon T, all possible complete histories will be of the form $\Phi_i^T = [\hat{s}_i^0, a_i^0, \hat{s}_i^1, a_i^1, \ldots, \hat{s}_i^{T-1}, a_i^T, \hat{s}_i^T]$.

Definition 7 (Events in a History)**.** A *primitive event* $e = (\hat{s}_i, a_i, \hat{s}_i')$ for an agent α_i is a triple representing a transition between two local states, given some action $a_i \in A_i$. An *event* $E = \{e_1, e_2, \ldots, e_h\}$ is a set of primitive events. A primitive event e *occurs in the history* Φ_i, written $\Phi_i \vDash e$, if and only if the triple e is a sub-sequence of the sequence Φ_i. An event E *occurs in the history* Φ_i, written $\Phi_i \vDash E$, if and only if some component occurs in that history: $\exists e \in E : \Phi_i \vDash e$.

Events can therefore be thought of disjunctively. That is, they specify a set of possible state-action transitions from a Dec-POMDP, local to one of its agents. If the historical sequence of state-action transitions that the agent encounters contains any one of those particular transitions, then the history satisfies the overall event. Events can thus be used, for example, to represent such things as taking a particular action in any one of a number of states over time, or taking one of several actions at some particular state. For technical reasons, namely the use of a specialized solution algorithm, these events are usually restricted in structure, as follows.[6]

Definition 8 (Proper Events)**.** A primitive event e is *proper* if it occurs at most once in any given history. That is, for any history Φ_i if $\Phi_i = \Phi_i^1 \, e \, \Phi_i^2$ then neither sub-history contains e: $\neg(\Phi_i^1 \vDash e) \wedge \neg(\Phi_i^2 \vDash e)$. An event E is *proper* if it consists of proper primitive events that are mutually exclusive, in that no two of them both occur in any history:

$$\forall \Phi_i \, \neg \exists x, y : (x \neq y) \wedge (e_x \in E) \wedge (e_y \in E) \wedge (\Phi_i \vDash e_x) \wedge (\Phi_i \vDash e_y).$$

Proper primitive events can be used, for instance, to represent actions that take place at particular times (building the time into the local state $\hat{s}_i \in e$). Since any given point in time can only occur once in any history, the events involving such time-steps will be proper by default. A proper event E can then be formed by collecting all the primitive events involving some single time-step, or by taking all possible primitive events involving an unrepeatable action. These ideas can then be used to define a non-independent reward structure for a Dec-POMDP $\mathcal{D}$, in terms of combinations of events for various subsets of the agents involved. Intuitively, the overall reward will be given in terms of what is to be gained or lost when every event in one of these combinations actually occurs during some joint history of the process.

Definition 9 (Joint Reward Structure)**.** A *joint reward structure* maps various sets of events for distinct agents to reward values:

$$\rho = \{\langle E_i^k, \ldots, E_i^m, c_i \rangle \mid 0 \leq i \leq j, \, j \in \mathbb{N}, \, c_i \in \Re\}$$

[5] See Papadimitriou and Tsitsiklis [8]. Strictly speaking, MDPs are only solvable in *pseudopolynomial* time by most common dynamic programming methods, since the iterations of the algorithm depend directly upon the time horizon, which is typically given in concise (binary) form; thus T iterations of an algorithm for a problem whose size is $(O)(\log T)$ is actually exponential in the problem size. If T is given in prolix (unary) fashion, then strictly polynomial solving time results. (These points are discussed in detail by Littman, Dean, and Kaelbling [9].) This is not typically considered a serious objection to treating MDPs as feasible, polytime problems; often the stipulation is made that $T \ll |S|$, for convenience.

[6] Becker et al. [10] describe how these limitations on the structure or events can be overcome, allowing a wider class of possible event-types.

where each event E_i^k ($1 \leq k \leq n$) is an event for agent α_k, and no agent appears twice in the sequence, so that there is no E_i^k and E_i^m such that $k = m$. We write ρ_i for the ith sequence in ρ, featuring events E_i^k and value c_i. We refer to $|\rho| = j$, the number of such tuples, as the size of the reward structure.

For any tuple $\langle E_i^k, \ldots, E_i^m, c_i \rangle \in \rho$, the system receives the reward c_i if each event in the tuple occurs; that is, reward is received if and only if, for each event E_i^k in the tuple, $\Phi_k \vDash E_i^k$. That is, the reward is received if every event it contains occurs somewhere in the joint history of the process. Note that, as defined, the sequence of events may involve a proper subset of agents, so that a given reward value c_i can depend only upon events featuring some, but not all, of the agents.

In principle, any reward-function R in a Dec-POMDP $\mathcal{D}$ can be defined in terms of such an event-based reward structure, and so this does not in any way specialize the general definition. However, such structures allow us to isolate some interesting properties of otherwise-independent problems. Furthermore, restricting dependence to the reward function alone reduces the overall complexity of finding optimal solution.

Any agent α_i in a factored Dec-MDP that is both transition- and observation-independent can act optimally based upon a *local policy*, $\pi_i : (S_i \times S_0) \rightarrow A_i$, mapping local states to local actions. The probability that some primitive event occurs, given that the agent follows some local policy is then straightforward. Letting $e = (\hat{s}_i, a_i, \hat{s}'_i)$, then $P(e \,|\, \pi_i) = 0$ if $\pi_i(\hat{s}_i) \neq a_i$, since the policy will guarantee that the agent never takes that particular action in that particular state; otherwise,

$$P(e \,|\, \pi_i) = \sum_{t=0}^{T} P_t(\hat{s}_i \,|\, \pi_i)\, P_i(\hat{s}'_i \,|\, \hat{s}_i, a_i), \tag{3}$$

where $P_t(\hat{s}_i \,|\, \pi_i)$ gives the probability of being in state $\hat{s}_i$ at time t, given policy π_i. This can be calculated straightforwardly from the Dec-MDP specification of state-transitions (and the other probability, $P_i(\hat{s}'_i \,|\, \hat{s}_i, a_i)$, is simply α_i's transition function). The expected value of a joint policy for all n agents follows immediately.

Definition 10 (Policy Value with Shared Rewards)**.** For a factored, n-agent Dec-MDP $\mathcal{D}$ with joint reward structure ρ, the *value of a joint policy* $(\pi_1, \ldots, \pi_n)$ is:

$$V(\pi_1, \ldots, \pi_n) = \sum_{i=1}^{|\rho|} c_i \prod_{E_i^k \in \rho_i} P(E_i^k \,|\, \pi_k).$$

The *optimal joint policy*, $(\pi_1, \ldots, \pi_n)^\star$, is one that maximizes the above equation.

3.1 Complexity of the joint-reward case

As it turns out, finding a policy that maximizes value in such a problem is provably easier than the worst-case NEXP-completeness of general Dec-POMDPs. While we do not go through the proof in detail, we state this important result here.

Theorem 3. Solving Dec-MDPs with independent transitions and observations, and a joint reward structure is NP-complete.

Proof Sketch (after Lemma 4, Goldman and Zilberstein [7]). For the upper bound, we show that the problem of deciding whether a given policy has expected value greater than some constant r is in NP. This follows straightforwardly from the policy value as given in Definition 10. Given any joint policy $(\pi_1, \ldots, \pi_n)$ for consideration, each $\rho_i \in \rho$ can be evaluated in time that is polynomial in the state space and number of agents, since each proper event involves a single distinct agent. Furthermore, each of these events can be evaluated in time polynomial in the problem size, simply summing up the probability of constituent primitive events, as determined in Equation (3). Since the reward-structure, ρ, is part of the problem description, the number of reward-components in the main sum, $|\rho|$, results in overall effort polynomial in the problem size.[7]

[7] This argument, while distinct from the one given by Goldman and Zilberstein, establishes the equivalent point. We include this version for variety's sake.

For the lower bound, we can reduce the NP-complete problem DTEAM (Papadimitriou & Tsitsiklis [11, 12]) to a Dec-MDP of the given type. We omit this reduction, which is quite elementary; the interested reader is directed to the full proof of Theorem 1, Becker et al. [10]. □

4 Other subclasses of interactions

Unfortunately, there is a limit to the power of the NP-completeness result. As we shall show, other relatively obvious combinations of independence relationships in Dec-MDPs do not yield the same general complexity reduction. Before we prove these new results, we consider the original proof of NEXP-completeness in Dec-POMDPs, which we draw on later.

We now prove our new results, establishing the complexity of certain other sub-classes of the Dec-POMDP framework, including two apparently restricted problem instances that have been of some interest in the existing literature. From one point of view, our results are wholly negative; that is, we show that the NP-completeness result of the prior section is specific to the fully transition- and observation-independent problems considered there. When these independence properties are not present to the full extent, the worst-case complexity is once again in NEXP.

4.1 Reward-independent-only models

We begin with a result that is rather simple, although it has not, to the best of our knowledge, been established before, and is worth noting. Namely, we consider the *inverse* of the NP-complete problem of Theorem 3; that is, rather than independent transitions and observations with shared rewards, we look at the complexity of problems for which rewards are independent, but transitions and observations are not. As we show, this returns complexity to the general case.

Theorem 4. Factored, reward-independent Dec-MDPs with n agents are NEXP-complete.

For our upper bound, as already described, we simply cite Theorem 1, which immediately establishes that such problems are in NEXP. For the lower bound, we modify the Dec-MDP from Bernstein's reduction proof so as to ensure that the reward-function factors appropriately into strictly local, single-agent rewards.

Proof of Lower Bound. For our reduction, we create a Dec-MDP that is identical to that of the Bernstein reduction, in terms of its state-set S, transition function P, observation sets and functions O_i and Ω_i, and time horizon T. We are then required to show how to factor the state-space appropriately, as required by Definition 1, and re-design the reward function to be independent, in accords with Definition 4.

State Space Factorization. We divide the state space into three distinct (3) parts:

1. S_0 consists of the jointly observable variables, $\{phase, index, origin, Q\}$.
2. S_1 consists of all those variables indexed to agent α_1, with the addition of the tile-type variable for the other agent, α_2, $\{index_1, value_2, pos_1, tile_1, tile_2\}$.
3. S_2 consists of all those variables indexed to agent α_2, with the exception of its tile-type variable, $\{index_2, value_2, pos_2\}$.

Clearly, then $S = S_0 \times S_1 \times S_2$, as required. To avoid confusion, we stress that we have not changed which portions of the state-space are observable to each agent. In particular, each α_i still *does not observe* the location-bit values $index_i$ and $value_i$, instead observing those for the other agent. Thus, the system still requires them to echo back location bits that they have seen faithfully. Since we do not require that the problem be transition- or observation-independent, it is permissible for an agent to fail to fully observe their own local state; so long as *some* agent's observations determine that portion of the state space, all is well. Furthermore, it does not matter that the actions of α_2 can affect the local state-space of α_1 (via the tile-type selection process), since the problem is not assumed to be transition-independent.

As before, we write an agent's local state as a tuple, with semicolons separating variables according to how they are observed. Thus, the example global state:

$$s = \langle query, 5, F, q_0; 3, 1, 0, t_3; 4, 0, 1, t_5 \rangle$$

is the combination of state $\hat{s}_1 \in S_0 \times S_1$ for α_1: $\langle query, 5, F, q_0; 0, t_3; 4, 0, t_5 \rangle$, and state $\hat{s}_2 \in S_0 \times S_2$ for α_2: $\langle query, 5, F, q_0; 3, 1; 1 \rangle$. Once again, we use the symbol "$*$" as a wildcard, allowing us to write, e.g., $\hat{s}_1 = \langle sel, *, *, *; *, *; *, *, * \rangle$ to designate the any state for α_1 during the **Select** phase.

Reward Factorization. For each agent, we define a local reward function as follows. Each agent receives reward of -1 for all local state-transitions, except for a special designated set of transitions, for which they receive reward 0. Each agent remains idle in the same phases as before, and so receives reward 0 in those phases, independent of their action-choices. In the **Echo** phase, each reward function gives 0 reward if and only if the agents faithfully echo their own stored bit (the exact nature of which they are still unaware). In the **Test** phase, the second agent α_2 receives 0 reward, no matter what; however, the first agent α_1 receives 0 just in case the tiling pair selected by the two of them is compatible according to the defined TILING instance.

$R(\hat{s}_1, a_1) = 0$ iff any of:	$R(\hat{s}_2, a_2) = 0$ iff any of:
Select: $\hat{s}_1 = \langle sel, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle sel, *, *, *; *, *; * \rangle$
Generate: $\hat{s}_1 = \langle gen, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle gen, *, *, *; *, *; * \rangle$
Query: $\hat{s}_1 = \langle query, *, *, *; *, *; *, *, * \rangle$	$\hat{s}_2 = \langle query, *, *, *; *, *; * \rangle$
Echo: $\hat{s}_1 = \langle echo, k, *, *; *, *; i_1, v_1, * \rangle$, $(a_1 = v_1$ or $k \neq i_1)$	$\hat{s}_2 = \langle echo, k, *, *; i_2, v_2; * \rangle$, $(a_2 = v_2$ or $k \neq i_2)$
Test (i): $\hat{s}_1 = \langle test, *, o, \mathbf{equal}; *, \mathbf{t_1}; *, *, \mathbf{t_1} \rangle$, $o = F$ or $\mathbf{t_1} = t_0$	$\hat{s}_2 = \langle test, *, *, *; *, *; * \rangle$.
Test (ii): $\hat{s}_1 = \langle test, *, *, \mathbf{hor}; *, \mathbf{t_1}; *, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$	
Test (iii): $s = \langle test, *, *, \mathbf{ver}; *, \mathbf{t_1}; *, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$	
Test (iv): $s = \langle test, *, *, \mathbf{apart}; *, *, ; *, *, * \rangle$.	

Finally, let our joint reward function simply be the sum of the local reward functions:

$$R(\langle s_0, s_1, s_2 \rangle, a_1, a_2) = R_1(\langle s_0, s_1 \rangle, a_1) + R_2(\langle s_0, s_2 \rangle, a_2) = R_1(\hat{s}_1, a_1) + R(\hat{s}_2, a_2).$$

Correctness of the Reduction. It is relatively easy to see that any policy in this new reduced problem has an expected non-negative (0) value precisely when the same policy has expected non-negative value in Bernstein's original reduced problem (since all actions are the same, and rewards are negative or not in exactly the same circumstances). Thus, the correctness proof for the Bernstein reduction applies directly here: there exists such a non-negative policy if and only if the TILING instance has some consistent solution. □

Thus we see that in some respects, transition and observation independence are fundamental to the reduction of worst-case complexity from NEXP to NP. When only the rewards depend upon the actions of both agents, the problems become easier; however, when the situation is reversed, the general problem remains NEXP-hard. As we remarked before, this is not entirely surprising: much of the complexity of planning in decentralized domains stems from the necessity to take account of how one's action-outcomes are affected by the actions of others, and from the complications that ensue when observed information about the system is tied to those actions as well. The structure of rewards, while obviously key to the nature of the optimal (or otherwise) solution, is not as vital—even if agents can separate their individual reward-functions, making them entirely independent, other dependencies can still make the problem extremely complex.

We therefore turn to two other interesting special-case Dec-MDP frameworks, in which independent reward functions are accompanied by restricted degrees of transition- and observation-based interaction. These models attempt to create simpler problem instances by imposing further structure on

the degree to which agents can affect one another, short of full independence. While some empirical evidence has suggested that these problems may be easier on average to solve, nothing has previously been shown about their worst-case complexity. We fill in these gaps, showing that even under such restricted dynamics, the problems remain NEXP-hard.

4.2 Event-driven interactions

The first model we consider is one of Becker et al. [13], which generalizes the notion of a fully transition-independent Dec-MDP, in a manner analogous to that used in the event-based reward structures discussed in Section 3. In this model, as before, a set of *primitive events*, consisting of state-action transitions, is defined for each agent. Such events can be thought of as occasions upon which that agent takes the given action to generate the associated state transition. *Dependencies* are then introduced in the form of relationships between one agent's possible actions in given states and another agent's primitive events; essentially, that is, an agent's actions are enabled (or not) due to the pre-existing occurrence (or not) of an event upon which it depends, modelled as a boolean variable. State-transitions are thus dependent upon the actions of other agents only insofar as some such dependence exists.

This model thus allows a simple counting approach to agent interactions, and the authors show that the Coverage Set algorithm introduced for fully transition-independent problem instances [13] in fact also applies to their less-restricted model in the presence of separate additive rewards. While no precise worst-case complexity results have been previously proven, the authors do point out that the class of problems has an upper-bound deterministic complexity that is exponential in the size of the state space, $|S|$, and doubly exponential in the number of defined interactions (which suggests nondeterministic exponential time complexity as an upper bound). This potentially bad news is mitigated by noting that if the number of interactions is small, then reasonably-sized problems can still be solved. Here, we examine this issue in detail, showing that, in fact these problems are NEXP-hard (indeed, NEXP-complete); however, when the number of dependencies is a log-factor of the size of the problem state-space, NP-hardness is achieved.

We begin with the formal framework of the model. *Histories* (Definition 6), *events* (Definition 7), and *proper events* (Definition 8), are all as before. The model is a Dec-MDP with:

1. Two (2) agents.[8]
2. A factored state-space: $S = S_0 \times S_1 \times S_n$.
3. Local full observability: each agent α_i can determine its own portion of the state-space, $\hat{s}_i \in S_0 \times S_i$, exactly.
4. Independent (additive) rewards: $R(\langle s_0, s_1, s_2 \rangle, a_1, a_2) = R_1(\hat{s}_1, a_1) + R_2(\hat{s}_2, a_2)$.

Interactions between agents are given in terms of a set of *dependencies* between certain state-action transitions for one agent, and events featuring transitions involving the other agent. Thus, if a history contains one of the primitive events from the latter set, this can have some direct effect upon the transition-model for the first agent, introducing probabilistic transition-dependencies.

Definition 11 (Dependency)**.** A *dependency* is a pair $d_{ij}^k = \langle E_i^k, D_j^k \rangle$, where E_i^k is a proper event defined over primitive events for agent α_i, and D_j^k is a set of state-action pairs $\langle \hat{s}_j, a_j \rangle$ for agent α_j, such that each pair occurs in at most one dependency:

$$\neg(\exists\, k, k', s_j, a_j)\ (k \neq k')\ \ \&\ \ \langle s_j, a_j \rangle \in D_j^k \in d_{ij}^k\ \ \&\ \ \langle s_j, a_j \rangle \in D_j^{k'} \in d_{ij}^{k'}.$$

Such a dependency is thus a collection of possible actions that agent α_j can take in one of its local state, each of which depends upon whether the other agent α_i has made one of the state-transitions in its own set of primitive events. Such structures can be used to model, for instance, cases where one agent cannot successfully complete some task until the other agent has completed an enabling sub-task, or where the precise outcome depends upon the groundwork laid by the other agent.

Definition 12 (Satisfying Dependencies)**.** A dependency $d_{ij}^k = \langle E_i^k, D_j^k \rangle$ is satisfied when the current history for enabling agent α_i contains the relevant event: $\Phi_i \vDash E_i^k$. For any state-action pair

[8] The model can be extended to n agents with little real difficulty. Since we will show that the 2-agent case is NEXP-hard, however, this will suffice for the general claim.

$\langle \hat{s}_j, a_j \rangle$, we define a Boolean indicator variable $b_{\hat{s}_j a_j}$, which is true if and only if some dependency that contains the pair is satisfied:

$$b_{\hat{s}_j a_j} = \begin{cases} 1 & \text{if } (\exists\, d_{ij}^k = \langle E_i^k, D_j^k \rangle)\ \langle \hat{s}_j, a_j \rangle \in D_j^k\ \ \&\ \ \Phi_i \vDash E_i^k, \\ 0 & \text{otherwise.} \end{cases}$$

The existence of dependencies allows us to factor the overall state-transition function into two parts, each of which depends only on an agent's local state, action, and the status of the relevant indicator variable.

Definition 13 (Local Transition Function)**.** The transition function for our Dec-MDP is factored into two functions, P_1 and P_2, each defining the distribution over next possible local states: $P_i(\hat{s}'_i \mid \hat{s}_i, a_i, b_{\hat{s}_i a_i})$. We can thus write $P_i(\hat{s}_i, a_i, b_{\hat{s}_i a_i}, \hat{s}'_i)$ for this transition probability.

When agents take some action in a state for which dependencies exist, they observe whether or not the related events have occurred; that is, after taking any action a_j in state s_j, they can observe the state of indicator variable $b_{\hat{s}_j a_j}$. (As Becker et al. [13] note, this can be changed so that agents observe the status of the indicator *before* they choose an action, if desired.)

With these definitions in place, we can now show that the worst-case complexity of the event-based problems is the same as the general Dec-POMDP class.

Theorem 5. Factored, finite-horizon, n-agent Dec-MDPs with local full observability, independent rewards, and event-driven interactions are NEXP-complete.

Again, the upper bound is immediate from the general case (Theorem 1). The event-based structure is just a specific case of general reward-dependence in a Dec-MDP, and such models can always be converted into Dec-MDPs where we define a joint transition function P, without using events in particular. Thus, if the more general Dec-POMDP class can be solved in nondeterministic exponential time, so can this special class. For the lower bound, we provide a reduction, again based on the Bernstein model, that builds in the special properties required.

Proof of Lower Bound. Unlike the previous reduction, in the proof of Theorem 4, we will need to add some variables to our state-space in this case, and alter the dynamics of the problem accordingly. In the original proof, each step of **Echo** consisted of both agents providing one of their location-bits, after which the problem advanced a step, and the FSA calculated its next state as it worked to determine the relative locations of the tiles given by the agents. The reward function provided a non-negative (0) reward just in case both agents were faithful in echoing back the memorized bits (observed by the other agent), and if the final result was a compatible pair of tiles. Such a reduction will not work directly for this problem for two reasons. First, the rewards given in the **Echo** phase depend upon how the actions of one agent relate to features (memorized location-bits) observed only by the *other* agent. Thus, our problem cannot be locally fully observed *and* locally reward independent, if we retain this feature. Second, the final reward, in the **Test** phase, the reward gained was based on the final state of the FSA, something observed by *both* agents, again defying local reward independence.

To solve the first of these problems, the intuitive idea will be to break each step in the **Echo** phase into sub-steps. Individual sub-steps will allow agents to act in sequence, after which the event-structure will ensure that there are local state features that lead to appropriate reward for the *other* agent. The second problem can be solved similarly, by adding extra sub-steps to the **Test** phase, and making the FSA state a locally observable part of the individual state-space of only one agent.

States of the Problem. Our reduction Dec-MDP will feature the following state variables, again broken down according to how they are observed.

S_0: Both Agents. Three (3) variables are observed at each step. First are variables $phase \in \{sel, gen, query, echo, test\}$, and $index \in \{0, \ldots, 2 \log n\}$, indicating the state of the system and the next (x, y)-location bit to be generated or echoed, just as before. Third, there is $sub \in A, B, C, D$, indicating possible sub-steps of each part of the **Echo** or **Test** phases (as explained below).

$\mathbf{S_1}$: Agent 1. This agent observes seven (8) variables of its own, as well. First, it observes $origin \in T, F$, and $q \in Q$, which are just the origin-point and FSA variables as before, although now only observed by the single agent. Second, it observes the index and value of the other agent's remembered location bit, $index_2 \in \{0, \ldots, 2\log n - 1\}$ and $value_2 \in \{0, 1\}$, and its own position-bit and chosen tile-type, $pos_1 \in \{0, 1\}$, and $tile_1 \in L$, again as before. Last, it observes a variable $act_2 \in \{0, 1, \bullet\}$, that corresponds to possible bit-echo actions the other agent may take during the relevant phase and variable $choice_2 \in L \cup \{\circ\}$, corresponding to possible tile-choice actions by the other agent, to be used in the **Test** phase (with $\bullet$ and $\circ$ being "null" values, in each case).

$\mathbf{S_2}$: Agent 2. This agent observes five (5) variables of its own, each of which is directly analogous to one of those observed by the first agent: $index_1$, $value_1$, pos_2, $tile_2$, and act_1. (Note that it does not observe the origin or FSA variables, nor does it have a recorder variable for the tile-choice of agent 1.)

Again, since the new variables each have a constant number of values, the set of all states $S = S_0 \times S_1 \times S_2$, is composed of factors, and has required size $|S| = \mathcal{O}(\log n)$, for suitably large n. Further, the problem is a locally fully observable Dec-MDP, since each agent observes its own local portion in entirety. Again, we write any such state as a tuple of possible variable values, using a semicolon to separate each of the three classes of observed variables, and use the "$*$" convention to abstract over features to which we are indifferent. As before, the Dec-MDP begins in the **Select** phase, with neither agent having chosen a tile-type, and the FSA at its initial state; the sub, act_i, and $choice_2$ variables are set to default values: $s_0 = \langle sel, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ; 0, 0, 0, t_0, \bullet\rangle$

State Transitions. As in our prior reduction, agent actions can either echo location-bits, or choose tile-types, so that $A_1 = A_2 = \{0, 1\} \cup L$. For the first three phases, **Select**, **Generate**, and **Query**, the problem dynamics are essentially identical to the original version (Section 1.5). That is, **Select** chooses memorized location-bits at random, and **Generate** produces two otherwise-random sequences of location-bits for each agent to memorize, and all other variables remain the same. For these two phases, the transition-models are thus the same as before, with the addition of the sub, act_i, and $choice_2$ variables, which do not change throughout from the initial state. Since those transitions are indifferent to the actions that agents choose, the joint transition function given in the original can be trivially factored into the separate local transition functions, P_1 and P_2 as required. Similarly, in the **Query** phase, the action-choices affect only local state-variables, and so factorization of the transition function is straightforward. (In both cases, no events or dependencies are required, and so the indicator variables are false by default, meaning that all transitions are entirely independent of them, and they need not be specified at all.) Therefore, we do not present all the details here, as they are unnecessary; we simply note that the **Query** phase ends with a transition to the first state in the **Echo** phase, namely:

$$s' = \langle echo, 0, A; T, q_0, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ; i_1, v_1, 0, \mathbf{t_2}, \bullet\rangle.$$

Each step in the previous **Echo** phase is now broken up into four sub-steps, A through D, as follows:

A: In this sub-step, the agents can choose whichever actions they like, and the state transitions to the next sub-step, regardless. (Note, however, that the action of agent 2 will be "recorded" in the next sub-step, based on a set of events that will be specified in the next section.)

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle echo, k, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ\rangle$,
$\hat{s}'_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ\rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle echo, k, A; i_1, v_1, 0, \mathbf{t_2}, \bullet\rangle$,
$\hat{s}'_2 = \langle echo, k, B; i_1, v_1, 0, \mathbf{t_2}, \bullet\rangle$.

Again, there are no event-dependencies for these state-action pairs, so the boolean indicator variables need not be specified.

B: We define a set of dependencies, one per step $k \in \{0, \ldots, (2\log n) - 1\}$ of the **Echo** phase. Letting $a_1^1, \ldots, a_1^m \in A_1$ be the possible actions for agent 1, each such dependency will be of the form $d_{21}^k = \langle E_2^k, D_1^k\rangle$ (that is, an event for agent 2, and a set of state-action pairs for

agent 1), with:

$$E_2^k = \{(\hat{s}_2, 1, \hat{s}_2')\}, \text{ and}$$
$$D_1^k = \{(\hat{s}_1, a_1^1), \ldots, (\hat{s}_1, a_1^m)\},$$

where $\hat{s}_2$, $\hat{s}_2'$ are the two local states of agent 2 as given in sub-step **A**, above, and $\hat{s}_1$ is the state for agent 1 as in the following:

$P(\hat{s}_1, a_1, b_{\hat{s}_1 a_1}, \hat{s}_1') = 1$ iff:
$\hat{s}_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$\hat{s}_1' = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, b_{\hat{s}_1 a_1}, \circ \rangle$,
where: $\mathbf{a_1} = 1$ if $a_1 = 1$, and 0 otherwise.

$P(\hat{s}_2, a_2, \hat{s}_2') = 1$ iff:
$\hat{s}_2 = \langle echo, k, B; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}_2' = \langle echo, k, C; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$.

That is, agent 1 transitions to a state that depends upon the state of the indicator variable: if agent 2 chose action "1" at the prior sub-step **A**, then this will be recorded in agent 1's local variable act_2, otherwise a zero (0) will be recorded. In this step, we now employ the variable pos_1, previously unused in this phase, to record the action taken at sub-step **B** by agent 1 (in the next step we will also use dependencies to record this choice in the other agent's state-space, just as we have recorded agent 2's here). We note that the dependency-structure defined fits the requirements of Definitions 7 and 11. Since there is only a single atomic event for agent 2, indexed by a k-value that ensures that it does not repeat, in each dependency-event E_2^k, it is obviously proper. In addition, since each state-action pair for agent 1 in D_1^k also features an indexed k-value, each these appear in no more than one dependency.

C: In this sub-step, we use the same trick to record the prior sub-step's action for agent 1, using the local state-variable for agent 2, act_1. Again, we define a set of dependencies, one per step $k \in \{0, \ldots, (2 \log n) - 1\}$. Letting $a_2^1, \ldots, a_2^m \in A_2$ be the possible actions for agent 2, each such dependency will be of the form $d_{12}^k = \langle E_1^k, D_2^k \rangle$, with:

$$E_1^k = \{(\hat{s}_1, 1, \hat{s}_1'), (\hat{s}_1, 1, \hat{s}_1'')\}, \text{ and}$$
$$D_1^k = \{(\hat{s}_2, a_2^1), \ldots, (\hat{s}_2, a_2^m)\},$$

where $\hat{s}_1$, $\hat{s}_1'$, $\hat{s}_1''$ are the local states of agent 1 as given in sub-step **B**, above ($\hat{s}_1'$ and $\hat{s}_1''$ correspond to the two possible outcomes for boolean variable $b_{\hat{s}_1 a_1}$), and $\hat{s}_2$ is the state for agent 2 as in the following:

$P(\hat{s}_1, a_1, \hat{s}_1') = 1$ iff:
$\hat{s}_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,
$\hat{s}_1' = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$.

$P(\hat{s}_2, a_2, b_{\hat{s}_2 a_2}, \hat{s}_2') = 1$ iff:
$\hat{s}_2 = \langle echo, k, C; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}_2' = \langle echo, k, D; i_1, v_1, 0, \mathbf{t_2}, b_{\hat{s}_2 a_2} \rangle$.

D: In this last sub-step, the system transitions to the next step of **Echo**, or to **Test**, as required. The variables for the origin (o) and FSA state (q) are updated in the local state of agent 1, based on the recorded actions of both agents.

$P(\hat{s}_1, a_1, \hat{s}_1') = 1$ iff:
$\hat{s}_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,
$\hat{s}_1' = \langle p, k', A; o', FSA(q, \mathbf{a_1}, \mathbf{act_2}), i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$$p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases}$$

$P(\hat{s}_2, a_2, \hat{s}_2') = 1$ iff:
$\hat{s}_2 = \langle echo, k, D; i_1, v_1, 0, \mathbf{t_2}, \mathbf{act_1} \rangle$,
$\hat{s}_2' = \langle p, k', A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$o' = T \Leftrightarrow (o = T \;\&\; \mathbf{a_1} = \mathbf{act_2} = 0)$.

Thus, we have used our dependencies to allow each agent to record the other agent's actions in certain sub-steps of the **Echo** phase (which will allow us, later on, to separate the reward function in a proper local manner). We note that there are a total of $4 \log n$ dependencies in the formulation, each of which is of constant size (depending only upon the number of actions each agent possibly has). Again, then, the size of this additional information is $\mathcal{O}(\log n)$, for suitable values of n, the size of the TILING grid.

Finally, we extend the **Test** phase. Where before the process the system would take a single step and terminate, we here take three (3) separate sub-steps.

A. This sub-step is deterministic, independent of choices of agent actions. However, we will set up a dependency in the next step in order to record the action of agent 2; later, we will set up the reward functions to force that agent to repeat the same choice of tile-types it made in the **Query** phase in order to gain non-negative joint reward.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle test, 0, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$\hat{s}'_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle test, 0, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}'_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle$.

B. For this sub-step, we create a set of dependencies, one for each possible tile-choice action for agent 2. Letting $m = |L|$, we let $a_2^1, \ldots, a_2^m$ be this set of actions. Then, for each such action a_2^k, we create a dependency $d_{21}^k = \langle E_2^k, D_1^k \rangle$, with:

$$E_2^k = \{(\hat{s}_2, a_2^k, \hat{s}'_2)^1, \ldots, (\hat{s}_2, a_2^k, \hat{s}'_2)^r\}, \text{ and}$$
$$D_1^k = \{(\hat{s}_1, a_1^1), \ldots, (\hat{s}_1, a_1^u)\},$$

where each primitive event for agent 2, $(\hat{s}_2, a_2^k, \hat{s}'_2)^j$, is one of the possibilities given in the previous step (there will be different ones, given possible combinations of variables i_1, v_1, and $\mathbf{t_2}$, but again no two can ever occur in a single history, so the event will be proper). Also, $a_1, \ldots, a_1^u$ are the set of all possible actions for agent 1, and the single state $\hat{s}_1$ is as in the following:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \circ \rangle$.
$\hat{s}'_1 = \langle test, 0, C; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle$,
where: $\mathbf{t_2} = a_2^k \Leftrightarrow b_{\hat{s}_1 a_1} = 1$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle$,
$\hat{s}'_2 = \langle test, 0, C; 0, 0, 0, t_0, \bullet \rangle$.

Thus, this step will record the value of agent 2's prior action, allowing the reward function to give the right reward in the final sub-step of **Test**, based only on agent 1's state. Again, the number of necessary dependencies is bounded by the number of tile-types in L, and so will not require undue amounts of space to specify during the reduction.

C. Finally, the **Test** phase terminates, moving to an absorbing final state.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle test, 0, C; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle$,
$\hat{s}'_1 = \langle test, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle test, 0, C; 0, 0, 0, t_0, \bullet \rangle$,
$\hat{s}'_2 = \langle test, 0, A; 0, 0, 0, t_0, \bullet \rangle$.

Local Reward Functions. We now present the factored, local reward function, similar to that one in the prior reduction (Section 4.1). In this problem, each agent again receives reward of -1 for all local state-transitions, except for a special designated set of transitions, for which they receive reward 0. Each agent remains idle in the same phases as before, and so receives reward 0 in those phases, independent of their action-choices. In the **Echo** phase, each reward function again gives 0 reward if and only if the agents faithfully echo their own stored bit (the exact nature of which they are still unaware). However, there is one catch, namely that the use of the sub-steps and recorder variables act_1 and act_2 allow us to give the reward for these choices to the *other agent*. Thus, agents can receive 0 reward even if they lie, but the do so knowing that they may cause the other agent to receive negative reward in future. The only joint policy with non-negative expected value is therefore one that forces both agents to tell the truth during **Echo**, just as before. As in the previous proof for the Reward-Independent-Only case, the **Test** phase only really applies to the first agent, who receives 0 just in case the tiling pair selected by both agents is compatible according to the defined TILING instance (again forcing the second agent to choose from a consistent tiling solution,

even if it personally accrues no local reward for doing so).

	$R(\hat{s}_1, a_1) = 0$ iff any of:	$R(\hat{s}_2, a_2) = 0$ iff any of:
Select:	$\hat{s}_1 = \langle sel, *, *; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle sel, *, *; *, *, *, *, * \rangle$
Generate:	$\hat{s}_1 = \langle gen, *, *; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle gen, *, *; *, *, *, *, * \rangle$
Query:	$\hat{s}_1 = \langle query, *, *; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle query, *, *; *, *, *, *, * \rangle$
Echo (A):	$\hat{s}_1 = \langle echo, *, A; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, *, A; *, *, *, *, * \rangle$
Echo (B):	$\hat{s}_1 = \langle echo, *, B; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, *, B; *, *, *, *, * \rangle$
Echo (C):	$\hat{s}_1 = \langle echo, k, C; *, *, i_2, v_2, *, *, \mathbf{act_2}, * \rangle$, $(\mathbf{act_2} = v_2$ or $k \neq i_2)$	$\hat{s}_2 = \langle echo, *, C; *, *, *, *, * \rangle$
Echo (D):	$\hat{s}_1 = \langle echo, *, D; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle echo, k, D; i_1, v_1, *, *, \mathbf{act_1} \rangle$, $(\mathbf{act_1} = v_1$ or $k \neq i_1)$
Test (A):	$\hat{s}_1 = \langle test, *, A; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle test, *, A; *, *, *, \mathbf{t_2}, * \rangle$, $a_2 = \mathbf{t_2}$
Test (B):	$\hat{s}_1 = \langle test, *, B; *, *, *, *, *, *, *, * \rangle$	$\hat{s}_2 = \langle test, *, B; *, *, *, *, * \rangle$
Test (C.i):	$\hat{s}_1 = \langle test, *, C; o, \mathbf{equal}, *, *, *, \mathbf{t_1}, *, \mathbf{t_1} \rangle$, $o = F$ or $\mathbf{t_1} = t_0$	$\hat{s}_2 = \langle test, *, *; *, *, *, *, * \rangle$.
Test (ii):	$\hat{s}_1 = \langle test, *, C; *, \mathbf{hor}, *, *, *, \mathbf{t_1}, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in H$	
Test (iii):	$s = \langle test, *, C; *, \mathbf{ver}, *, *, *, \mathbf{t_1}, *, \mathbf{t_2} \rangle$, $\langle \mathbf{t_1}, \mathbf{t_2} \rangle \in V$	
Test (iv):	$s = \langle test, *, C; *, \mathbf{apart}, *, *, *, *, *, * \rangle$.	

Finally, let our joint reward function simply be the sum of the local reward functions:

$$R(\langle s_0, s_1, s_2 \rangle, a_1, a_2) = R_1(\langle s_0, s_1 \rangle, a_1) + R_2(\langle s_0, s_2 \rangle, a_2) = R_1(\hat{s}_1, a_1) + R(\hat{s}_2, a_2).$$

Observations and Horizon. Again, we simply set the observation function for each agent to be deterministic: given a state, agent α_i simply observes the four variables that make up its portion of the state space, along with the global variables that make up the first portion. This also takes care of the requirement that agents be able to observe whether or not dependencies are satisfied after taking relevant actions, since each agent's state-space contains variables that determine the value of the indicator variables in the right spots. The time horizon is $T = (10 \log n) + 6$, consisting of one step each for the **Select**, and **Query** stages, three steps for **Test**, $(2 \log n) + 1$ steps to **Generate** the bits of the pair of (x, y) locations, and $4 \times (2 \log n)$ steps for the multi-step **Echo** phase.

Correctness of the Reduction. Again, it is straightforward that any policy in this new reduced problem has an expected non-negative (0) value precisely when the same policy has expected non-negative value in Bernstein's original reduced problem. The first three phases are exactly the same, since agents "idle" and accumulate 0 reward no matter what. Then, in the **Echo** phase, agents are again required to faithfully repeat observed location-bits: if they do not, then there is some positive probability that the other agent will receive a negative reward on a later sub-step. Finally, during the **Test** phase, sub-step **A** requires that agent 2 repeats the same tile-type it chose during **Query**. Finally, the last step simply replicates the final step of Bernstein's reduction, giving non-negative reward only for compatible tile choices. Thus, the correctness proof for the original reduction again applies directly: there exists such a non-negative policy if and only if the TILING instance has some consistent solution. □

4.2.1 A special, NP-hard case

The reduction just presented relies on the fact that we allow the number of dependencies in the problem to grow as a factor of $\log n$, where n is the size of the grid in the original TILING instance. (We do not allow the set to grow any larger, lest the reduction no longer suffice for the complexity proof.) Since the overall size of the state-space S in the reduced Dec-MDP state-space is also $\mathcal{O}(\log n)$, this means that the number of dependencies is $\mathcal{O}(|S|)$. Thus, the NEXP-completeness

result will hold for any Dec-MDP with event-based transitions where the number of dependencies is polynomial in the size of the state-space.

When we are able to restrict the number of dependencies further, however, we can do better in terms of the upper bound on worst-case complexity.

Theorem 6. A factored, finite-horizon, n-agent Dec-MDP with local full observability, independent rewards, and event-driven interactions are solvable in nondeterministic polynomial time (NP) if the number of dependencies is $\mathcal{O}(\log |S|)$, where S is the state-set of the problem.

Proof. Let $\mathcal{D}$ be a Dec-MDP as described, and let S be its state-set. Let $\mathbf{d}$ be the set of dependencies; by assumption, $|\mathbf{d}| = \mathcal{O}(\log |S|)$. We let $d_1, \ldots, d_{|\mathbf{d}|}$ be a set of new variables, one per dependency. For any agent α_i, we have that the state-set $S_i = S_0 \times S_i$ is some portion of the global state-set S. Further, for any α_i, we let $\mathbf{d}_i \subseteq \mathbf{d}$ be those dependencies $d^k_{ji} = \langle E^k_j, D^k_i \rangle$ that involve possible state-action pairs of that agent. Since agents can observe whether or not dependencies are fulfilled, we can augment the local state-space of each α_i with the dependencies for that agent, so that $S'_i = S_0 \times S_i \times \mathbf{d}_i$.

The result is a Dec-MDP with state-space $S' = S_0 \times S_1 \times \mathbf{d}_1 \times \cdots \times S_n \times \mathbf{d}_n$, where any local policy for an agent α_i is a mapping from local states $\hat{s}'_i \in S'_i$ to actions $a_i \in A_i$. (Since the local states now contain all information to which each agent has access during the course of any policy run, these now suffice. Given any joint policy for the augmented problem, we can use dynamic programming to evaluate it in time that is polynomial in the size of the new state-space. Since the sum-total of all dependencies is $\mathcal{O}(\log n)$, we have that the size of the augmented state-space is polynomial in the size of the original: $|S'| = \mathcal{O}(|S|)$, and therefore that we can evaluate the policy in time that is polynomial in the size of the original, as well. Thus, the problem class can be solved in nondeterministic polynomial time. □

4.3 State-dependent actions

Guo [14, 15, 16] considers another specialized subclass Dec-MDPs based on apparently even more restricted types of interaction. In this model, we again deal with separate agent state-spaces, and all agent action-transitions and rewards are independent of the actions of other agents. Such problems are not wholly decoupled, however, as the actions available to each agent at any point depend upon the global state of the system. Thus, agents interact by making choices that restrict or broaden the range of actions available to others.

Definition 14 (Dec-MDP with State-Dependent Actions)**.** An n-agent *Dec-MDP with state-dependent actions* is a tuple $\mathcal{D} = \langle S_0, \{S_i\}, \{A_i\}, \{B_i\}, \{P_i\}, \{R_i\}, T \rangle$, where:

- S_0 is a set of shared states, and each S_i is the state-space of agent s_i, with the global state space $S = S_0 \times S_1 \times \cdots \times S_n$. We let $s^0 \in S$ be the initial state.
- Each A_i is the action-set for α_i.
- Each $B_i : S \to 2^{A_i}$ is a mapping from *global states* of the system to some set of available actions for each agent α_i. For all $s \in S$, $B_i(s) \neq \emptyset$.[9]
- $P_i : (S_0 \times S_i) \times A_i(S_0 \times S_i)$ is the state-transition function over local states for α_i. The global transition function is simply the product of individual P_i.
- $R_i : (S_0 \times S_i) \to \Re$ is a local reward function for agent α_i. We let the global reward function be the sum of local rewards.[10]
- $T \in \mathbb{N}$ is the finite time-horizon of the problem.

[9] This requirement—that agents have some action-choice in every state—is a basic presumption of most Markov decision models. We note that it can always be relaxed by simply providing a "no-op" action that causes self-transitions in some state, having the same practical effect as an absence of actions. Guo also requires that $B_i(s) \subset S$, for all s, so that at every state there is some action that is not available. Since this requirement can be trivially satisfied in any problem by simply adding some new action that is never available anywhere, we choose to ignore it here.

[10] In the original formulation, this additive feature is not present, and the global reward function can be any arbitrary combination of local rewards. Since we will show our complexity for this more restricted version of the reward function, this will suffice for the more general case as well.

We note that there need be no observations in such a problem; since we presume local full observability, each agent's observations are just their local states. Furthermore, it is presumed that each agent can observe its own available actions in any state; a local policy is thus a mapping from local states to available actions.

For such cases, Guo presents a planning algorithm based on heuristic action-set pruning, along with a learning algorithm that combines pruning with Q-learning (on the latter, see [17, chap. 6]). While empirical results show that these methods are capable of solving potentially large instances, we again know very little analytically about the actual difficulty of solving problems with state-dependent actions. An NP-hardness lower bound is given [14] for the overall class, by reducing a normal-form game problem to the state-dependent but this is potentially quite weak, since no upper bound has been shown, and even the operative algorithmic complexity of the solution methods given is not well understood. We rectify this situation, showing that in fact the problem of determining whether a given instance of such a problem has a policy with non-negative value is also just as hard as the general case.[11]

Theorem 7. Factored, finite-horizon, n-agent Dec-MDPs with local full observability, independent rewards, and state-dependent action-sets are NEXP-complete.

Once more, we can rely upon the general upper bound on the complexity of Dec-POMDPs (Theorem 1). These sorts of Dec-MDPs are obviously special cases of the general model, which can easily incorporate state-based actions simply by allowing every action in every global state.

Proof of Lower Bound. This reduction will be very like the prior one, for event-driven interactions (Theorem 5). Again, we will use the trick of "recording" certain actions of each agent in the state-space of the other, allowing us to have purely local rewards and local full observability. This time, however, we will use the idea of action-sets that are dependent upon the global state, rather than event-based dependencies, to enforce the dynamics we desire.

State-Space. We again give a reduction of any TILING problem to a 2-agent instance of the Dec-MDP with state-dependent actions. The state-space is exactly as for the prior reduction, and is divided the same way into the local, fully observable states for each agent. We do not repeat the description here, noting only that we again start with state $s_0 = \langle sel, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ; 0, 0, 0, t_0, \bullet \rangle$.

Actions and Transitions. As before, both agents possess actions $\{0, 1\} \cup L$. Unless otherwise specified, these actions are available for both agents in every state; exceptions, along with a set of additional actions for each agent are detailed below.

Again, our reduction works identically to the previous one through the first three phases (**Select**, **Generate**, and **Query**) of the problem, leading to the initial state of the **Echo** phase, which is again $s' = \langle echo, 0, A; T, q_0, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$. This phase is again divided into four sub-steps, A through D, with details as follows.

A: In this sub-step, the agents can choose whichever actions they like (all are available), and the state transitions to the next sub-step deterministically. The action of agent 2 is "recorded" by its own pos_2 variable, which in previous reductions was unused in this phase of the problem, remaining fixed ($pos_2 = 0$).

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle echo, k, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$\hat{s}'_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle echo, k, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}'_2 = \langle echo, k, B; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$,
$\mathbf{a_2} = 1$ if $a_2 = 1$ and is 0 otherwise.

[11] Again, in the original literature, the reward function for the problem is an arbitrary function of local rewards. Thus, the problem is more generally a *partially observable stochastic game* (POSG), and the initial complexity results are formulated in terms of equilibria, rather than optimal cooperative policies. Here, we show that the more restricted, purely additive, solution concept suffices.

B: We add four (4) actions to agent 1's action set A_1, $a^{00}, a^{01}, a^{10}, a^{11}$. Intuitively, we will use a^{ij} to allow us to record the situation where agent 1 echoes back bit-value i, given that agent 2 has echoed back bit-value j (in sub-step **A**). Formally, we enforce this by restricting its action-set function B_1:

$$B_1(\langle echo, *, B; *, *, *, *, *, *, *, *, *; *, *, 0, *, * \rangle) = \{a^{00}, a^{10}\}$$
$$B_1(\langle echo, *, B; *, *, *, *, *, *, *, *, *; *, *, 1, *, * \rangle) = \{a^{01}, a^{11}\}$$

Given this limitation, we can then record agent 1's own choice of bit in its own pos_1 variable, and agent 2's choice in agent 1's act_2 variable:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle echo, k, B; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$\hat{s}'_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,
$\mathbf{a_1} = 1$ if $a_1 \in \{a^{10}, a^{11}\}$, and 0 otherwise,
$\mathbf{act_2} = 1$ if $a_1 \in \{a^{01}, a^{11}\}$, and 0 otherwise.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle echo, k, B; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}'_2 = \langle echo, k, C; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$.

C: In this sub-step, we use the same trick to record the prior sub-step's action for agent 1, using the local state-variable for agent 2, act_1. We add two actions, $a^{\bullet 0}$ and $a^{\bullet 1}$ to agent 2's action set A_2, and force it to choose exactly one of them, based on the variable in agent 1's own local state-space that records its prior action (from sub-step **B**).

$$B_2(\langle echo, *, C; *, *, *, *, 0, *, *, *; *, *, *, *, * \rangle) = \{a^{\bullet 0}\}$$
$$B_2(\langle echo, *, C; *, *, *, *, 1, *, *, *; *, *, *, *, * \rangle) = \{a^{\bullet 1}\}$$

Given this restriction, we can then force the recording of agent 1's choice of bit in agent 2's act_1 variable:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle echo, k, C; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,
$\hat{s}'_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle echo, k, C; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}'_2 = \langle echo, k, D; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \mathbf{act_1} \rangle$,
$\mathbf{act_1} = 1$ if $a_2 = a^{\bullet 1}$, and 0 otherwise.

D: Finally, the system transitions to the next step of **Echo**, or to **Test**, as before. The variables for the origin (o) and FSA state (q) are again updated in the local state of agent 1, based on recorded actions of both agents.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle echo, k, D; o, q, i_2, v_2, \mathbf{a_1}, \mathbf{t_1}, \mathbf{act_2}, \circ \rangle$,
$\hat{s}'_1 = \langle p, k', A; o', FSA(q, \mathbf{a_1}, \mathbf{act_2}), i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$$p, k' = \begin{cases} echo, k+1 & \text{if } 0 \leq k < (2 \log n) - 1 \\ test, 0 & \text{if } k = (2 \log n) - 1 \end{cases}$$

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle echo, k, D; i_1, v_1, \mathbf{a_2}, \mathbf{t_2}, \mathbf{act_1} \rangle$,
$\hat{s}'_2 = \langle p, k', A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$o' = T \Leftrightarrow (o = T \;\&\; \mathbf{a_1} = \mathbf{act_2} = 0)$.

Thus, just as before we used event-based dependencies, we have now used restrictions on the sets of actions to allow (indeed, *to force*) each agent to record the other agent's actions in certain sub-steps of **Echo**, again allowing us to later produce a fully factored local reward function. To do so only requires the addition of a fixed number of actions, independent of the state-space size, and cannot upset the reduction process by increasing specification size too much.

We use similar techniques in the **Test** phase of the problem, this time expanding each step to two separate sub-step (one less than in the prior reduction), and recording necessary information using action-set selection techniques.

A. In this sub-step, we record the tile-choice of agent 2 into the $choice_2$ variable, which is part of agent 1's local state-space. To do so, we first limit agent 1's action-set to exactly one tile-type selection actions from L, directly corresponding to the tile chosen by the other agent.

(Note that since we use actions that already exist in the original problem specification, this step does not require any increase in the problem size at all.)

$$B_1(\langle test, 0, A; *, *, *, *, *, *, *, *, *; *, *, *, \mathbf{t_2}, * \rangle) = \{a \in L \mid a = \mathbf{t_2}\}.$$

Then, we have state transitions as follows:

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle test, 0, A; o, q, i_2, v_2, 0, \mathbf{t_1}, \bullet, \circ \rangle$,
$\hat{s}'_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle$,
where: $\mathbf{t_2} = a_1$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle test, 0, A; i_1, v_1, 0, \mathbf{t_2}, \bullet \rangle$,
$\hat{s}'_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle$.

B. As before, the **Test** phase terminates, moving to an absorbing final state.

$P(\hat{s}_1, a_1, \hat{s}'_1) = 1$ iff:
$\hat{s}_1 = \langle test, 0, B; o, q, 0, 0, 0, \mathbf{t_1}, \bullet, \mathbf{t_2} \rangle$,
$\hat{s}'_1 = \langle test, 0, A; T, q_0, 0, 0, 0, t_0, \bullet, \circ \rangle$.

$P(\hat{s}_2, a_2, \hat{s}'_2) = 1$ iff:
$\hat{s}_2 = \langle test, 0, B; 0, 0, 0, t_0, \bullet \rangle$,
$\hat{s}'_2 = \langle test, 0, A; 0, 0, 0, t_0, \bullet \rangle$.

Rewards, Observations, and Time Horizon. The reward function for this problem is essentially identical to that for the previous reduction (page 19), as the state-spaces are the same. The only difference is that we only have two sub-steps in the **Test** phase, rather than three. Our new reward function simply replaces the rewards previously given in sub-step **B** with those previously given in sub-step **C**, and eliminates the reference to an action for agent 2 in sub-step **A**, since that agent is now idle. As it is otherwise identical, we do not repeat it here. Observations are likewise as before (under the general presumption that agents always know what action-choices they have). The time horizon $T = (10 \log n) + 5$, which is one step less than the prior problem, simply because we have eliminated a single sub-step in **Test**.

Correctness. Given the functionally identical reward functions, this reduction produces exactly the same non-negative policies as before. Thus, agents must still echo bits back reliably or risk negative rewards, and there is a policy with non-negative expected value just in case there is a consistent solution to the original TILING instance. □

References

[1] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 32–37, Stanford, California, 2000.

[2] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[3] Daniel S. Bernstein. *Complexity Analysis and Optimal Algorithms for Decentralized Decision Making*. PhD thesis, University of Massachusetts, Amherst, 2005.

[4] Harry R. Lewis. Complexity of solvable cases of the decision problem for predicate calculus. In *Proceedings of the Nineteenth Symposium on the Foundations of Computer Science*, pages 35–47, Ann Arbor, Michigan, 1978.

[5] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[6] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147:5–34, 2003.

[7] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[8] Christos H. Papadimitriou and John Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[9] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 394–402, Montréal, Québec, Canada, 1995.

[10] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V. Goldman. Solving transition independent decentralized MDPs. *Journal of Artificial Intelligence Research*, 22:423–455, November 2004.

[11] Christos H. Papadimitriou and John Tsitsiklis. On the complexity of designing distributed protocols. *Information and Control*, 53:211–218, 1982.

[12] Christos H. Papadimitriou and John Tsitsiklis. Intractable problems in control theory. *SIAM Journal on Control and Optimization*, 24(4):639–654, 1986.

[13] Raphen Becker, Victor Lesser, and Shlomo Zilberstein. Decentralized Markov decision processes with event-driven interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 302–309, New York, New York, 2004.

[14] AnYuan Guo. *Planning and Learning for Weakly-Coupled Distributed Agents*. PhD thesis, University of Massachusetts, Amherst, 2006.

[15] AnYuan Guo and Victor Lesser. Planning for weakly-coupled partially observable stochastic games. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1715–1716, Edinburgh, Scotland, 2005.

[16] AnYuan Guo and Lesser Victor. Stochastic planning for weakly-coupled distributed agents. In *Proceedings of the Fifth Joint Conference on Autonomous Agents and Multiagent Systems*, pages 326–328, Hakodate, Japan, 2006.

[17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 2000.