The joint distribution for all these random variables is
specified by the product of the conditional distributions
\[ P\left\{ \begin{array}{l}
X_{1t} = x_{1t}, X_{2t} = x_{2t} \\
U_{1t} = u_{1t}, U_{2t} = u_{2t}
\end{array} \right| X_{1}^{t-1} = x_{1}^{t-1}, X_{2}^{t-1} = x_{2}^{t-1} \right\} \\
= p_{1t}(x_{1t}|x_{1,t-1}, u_{1,t-1}),
\]
\[ = p_{2t}(x_{2t}|x_{2}, x_{1,t-1}, x_{2,t-1}, u_{1,t-1}, u_{2,t-1}),
\]
\[ q_{1t}(u_{1t}|x_{1}^{t}, u_{1}^{t-1}), q_{2t}(u_{2t}|x_{1}^{t}, x_{2}^{t}, u_{1}^{t}, u_{2}^{t-1}) \]
where
1. \( p_{1t}(x_{1t}|x_{1,t-1}, u_{1,t-1}) \) is the transition function for player 1, represented by the conditional distribution of player 1’s state given its past state and action. Note that player 1’s state is not affected by player 2.
2. \( p_{2t}(x_{2t}|x_{2}, x_{1,t-1}, x_{2,t-1}, u_{1,t-1}, u_{2,t-1}) \) is the transition function for player 2, represented by the conditional distribution of player 2’s state given both player 1 and player 2’s past state and action. Here, player 2’s state is affected by player 1.
3. \( q_{1t}(u_{1t}|x_{1}^{t}, u_{1}^{t-1}) \) is the (randomized) controller for player 1, represented by the conditional distribution of player 1’s action given the history of player 1’s states and actions. Again, note that player 1’s action is not affected by player 2.
4. \( q_{2t}(u_{2t}|x_{1}^{t}, x_{2}^{t}, u_{1}^{t}, u_{2}^{t-1}) \) is the (randomized) controller for player 2, represented by the conditional distribution of player 2’s action given the history of both players’ states and actions. Here, player 2’s action is obviously affected by player 1.

For the \( t = 0 \) case, we can assume that \( X_{1,-1} = X_{2,-1} = U_{1,-1} = U_{2,-1} = 0 \), so that we can drop \( x_{1}, x_{2}, u_{1}, u_{2}, \) and \( u_{1}, u_{2} \) arguments when convenient.

For each \( t \), let \( c_{t}(X_{1t}, X_{2t}, U_{1t}, U_{2t}) \) denote the cost at time \( t \), where \( c_{t} : X_{1} \times X_{2} \times U_{1} \times U_{2} \rightarrow \mathbb{R} \) is a fixed function. The goal is to choose controllers \( q_{1} = (q_{10}, \ldots, q_{1N}) \) and \( q_{2} = (q_{20}, \ldots, q_{2N}) \) to minimize the total expected cost from time 0 to \( N \), i.e.
\[
J(q_{1}, q_{2}) = \sum_{t=0}^{N} E[c_{t}(X_{1t}, X_{2t}, U_{1t}, U_{2t})]
\]
\[
= \sum_{t=0}^{N} \sum_{u_{1}^{t}, u_{2}^{t}} \prod_{i=0}^{t} p_{1i}(x_{1i}|x_{1,i-1}, u_{1,i-1}), \quad p_{2i}(x_{2i}|x_{1i}, x_{2,i-1}, u_{1,i-1}, u_{2,i-1}), \quad q_{1i}(u_{1i}|x_{1i}, u_{1}^{i-1}), \quad q_{2i}(u_{2i}|x_{1i}, x_{2i}, u_{1}^{i}, u_{2}^{i-1}) \]
\[ c_{t}(x_{1t}, x_{2t}, u_{1t}, u_{2t}) \]

**Dynamic Programming Solution**

To develop a practical dynamic programming solution to this problem, we need to prove the following key result.

**Theorem 1.** There exists optimal controllers \( (q_{1}^{*}, q_{2}^{*}) \) where for each \( t \), there are functions \( K_{1t} : X_{1t}^{t} \rightarrow U_{1} \) and \( K_{2t} : X_{1t}^{t} \times X_{2t}^{t} \rightarrow U_{2} \) where
\[
q_{1t}^{*}(u_{1t}|x_{1}^{t}, u_{1}^{t-1}) = \begin{cases} 
1, & u_{1t} = K_{1t}(x_{1}^{t}) \\
0, & \text{otherwise}
\end{cases}
\]
\[
q_{2t}^{*}(u_{2t}|x_{1}^{t}, x_{2}^{t}, u_{1}^{t}, u_{2}^{t-1}) = \begin{cases} 
1, & u_{2t} = K_{2t}(x_{1}^{t}, x_{2t}) \\
0, & \text{otherwise}
\end{cases}
\]

In other words, there are optimal controllers such that \( U_{1t} \) is a deterministic function of \( X_{1t}^{t} \) and \( U_{2t} \) is a deterministic function of \( (X_{1t}^{t}, X_{2t}^{t}) \).

**Proof.** Note that \( J(q_{1}, q_{2}) \) is a continuous function of \( (q_{1}, q_{2}) \). Moreover, the set of possible \( (q_{1}, q_{2}) \) is a finite Cartesian product of probability simplexes, and is therefore compact. Thus \( J(q_{1}, q_{2}) \) has a minimizer, which we will call \( (q_{1}^{*}, q_{2}^{*}) \).

We now show how to transform \( (q_{1}^{*}, q_{2}^{*}) \) into the desired form without affecting optimality. For any given \( t \) and \( (x_{1}^{t}, u_{1}^{t-1}) \), note that \( J(q_{1}^{*}, q_{2}^{*}) \) is an affine function of the variables \( q_{1i}^{*}(\cdot|u_{1i}^{t-1}) \), provided the other variables are fixed. Let \( K_{1i}(x_{1i}, u_{1i}^{t-1}) \in U_{1i} \) be the index of a variable in \( q_{1i}^{*}(\cdot|u_{1i}^{t-1}) \) whose corresponding linear coefficient achieves the lowest value. Then if we replace \( q_{1i}^{*}(\cdot|u_{1i}^{t-1}) \) with
\[
q_{1i}^{*}(u_{1i}|x_{1i}, u_{1i}^{t-1}) = \begin{cases} 
1, & u_{1i} = K_{1i}(x_{1i}, u_{1i}^{t-1}) \\
0, & \text{otherwise}
\end{cases}
\]
we can never increase \( J(q_{1}^{*}, q_{2}^{*}) \). Thus without affecting optimality, we can transform the optimal controller for player 1 into a deterministic one, and by a similar argument, we can also do so for player 2.

We have so far shown that \( U_{1t} \) needs only to be a deterministic function of \( (X_{1t}^{t}, U_{1t-1}^{t-1}) \) and \( U_{2t} \) a deterministic function of \( (X_{1t}^{t}, X_{2t}^{t}, U_{1t-1}^{t-1}) \). It is easy to see that \( U_{1t} \) only needs to be a function of \( X_{1t} \) since \( U_{10} \) is a function of only \( X_{10} \), and if \( U_{10}, \ldots, U_{i-1} \) are functions of only \( X_{10}, \ldots, X_{i-1}^{t-1} \), respectively, then \( U_{1t} \) is a function of \( (X_{1t}^{t}, U_{1t-1}^{t-1}) \), and therefore of just \( X_{1t}^{t} \). By a similar inductive argument, \( U_{2t} \) needs to be a function of only \( (X_{1}^{t}, X_{2}^{t}) \).

To complete the proof, we need to show that \( U_{2t} \) actually can be a function of only \( (X_{1}^{t}, X_{2}^{t}) \). In light of what we have just proven, we can rewrite the cost function as
\[
J(q_{1}, q_{2}) = \sum_{t=0}^{N} \sum_{u_{1}^{t}, u_{2}^{t}} \prod_{i=0}^{t} p_{1i}(x_{1i}|x_{1,i-1}, u_{1,i-1}), \quad p_{2i}(x_{2i}|x_{1i}, x_{2,i-1}, u_{1,i-1}, u_{2,i-1}), \quad q_{1i}(u_{1i}|x_{1i}, u_{1}^{i-1}), \quad q_{2i}(u_{2i}|x_{1i}, x_{2i}, u_{1}^{i}, u_{2}^{i-1}) \]
\[ c_{t}(x_{1t}, x_{2t}, u_{1t}, u_{2t}) \]
where the dependencies of \( q_{1i} \) on \( u_{1}^{i-1} \) and \( q_{2i} \) on \( (u_{1}^{i}, u_{2}^{i-1}) \) have been dropped. As before, let \( (q_{1}^{*}, q_{2}^{*}) \) be a minimizer of this cost function. Then given \( t \) and \( (x_{1}^{t}, x_{2}^{t}) \), \( J(q_{1}^{*}, q_{2}^{*}) \) can be written as the following affine function of \( q_{2t}^{*}(\cdot|x_{1}^{t}, x_{2}^{t}) \) (provided the other variables are fixed):
\[
\alpha_{t}(x_{1}^{t}, x_{2}^{t}) \sum_{u_{2t}} \beta_{t}(x_{1}^{t}, x_{2}^{t}, u_{2t}) q_{2t}^{*}(u_{2t}|x_{1}^{t}, x_{2}^{t}) + \gamma_{t}(x_{1}^{t}, x_{2}^{t})
\]
where

$$
\alpha_t(x_1^t, x_2^t) = \sum_{i=0}^{t-1} \prod_{u_2^{i-1}} p_1(x_i|x_{i-1}, u_{i-1}) + \prod_{u_2^{t-1}} p_2(x_t|x_{t-1}, u_{t-1})
$$

and

$$
\beta_t(x_1^t, x_2^t) = \sum_{u_2^t} q_1^t(u_2^t) \left[ c_t(x_t, x_1^t, u_t^t, u_2^t) + \sum_{\tau=t+1}^{N} \prod_{u_2^{\tau-1}} q_1^\tau(u_2^{\tau-1}) q_2^\tau(u_2^\tau) \right]
$$

where $x_i^{t+1}$ means the vector $(x_{i,t+1},\ldots,x_{i,T})$. Note that for $t = N$, $\beta_N(x_N^N, u_N^N)$ is actually a function of only $(x_N^N, x_{N+1}, u_{N+1})$. Thus let $K_{2N}(x_N^N, x_{N+1})$ be a $u_{2N}$ where $\beta_N(x_N^N, x_{N+1}, u_{2N})$ is minimized. Then by replacing $q_{2N}(x, u_2^N)$ with

$$
q_{2N}^*(u_2|x_1^t, x_2^t) = \begin{cases} 1, & u_2 = K_2(x_1^t, x_2^t) \\ 0, & \text{otherwise} \end{cases}
$$

we cannot increase the cost function.

Now suppose for all $\tau > t$ and $(x_1^\tau, x_2^\tau)$, $q_{2N}^*(x, u_2^\tau)$ is only a function of $(x_1^\tau, x_2^\tau)$, then it follows that $\beta_t(x_1^t, x_2^t, u_2^t)$ is only a function of $(x_1^t, x_2^t, u_2^t)$, so we can define $K_{2t}(x_1^t, x_2^t)$ to be a $u_{2t}$ where $\beta_t(x_1^t, x_2^t, u_2^t)$ is minimized. Then by replacing $q_{2t}(x, u_2^t)$ with

$$
q_{2t}^*(u_2|x_1^t, x_2^t) = \begin{cases} 1, & u_2 = K_2(x_1^t, x_2^t) \\ 0, & \text{otherwise} \end{cases}
$$

we cannot increase the cost function, as before. The theorem follows by induction.

In light of the previous theorem, we only need to consider deterministic controllers $K_1 = (K_{10},\ldots,K_{1N})$ and $K_2 = (K_{20},\ldots,K_{2N})$, where $K_{1t}$ and $K_{2t}$ are functions of the form $K_{1t} : x_1^t \rightarrow U_1$ and $K_{2t} : x_2^t \rightarrow U_2$. Moreover, by rewriting the cost function $J$ as a function $(K_1, K_2)$, we have

$$
J(K_1, K_2) = \sum_{t=0}^{N} \sum_{x_1^t, x_2^t} \prod_{i=0}^{t} p_1(x_i|x_{i-1}, K_{1,i-1}(x_{i-1})) + \prod_{i=0}^{t} p_2(x_i|x_{i-1}, K_{2,i-1}(x_{i-1}))
$$

and

$$
\gamma_t(x_1^t, x_2^t) = \sum \text{ components in } J(q_1^*, q_2^*) \text{ not containing } q_{2t}(x_1^t, x_2^t)
$$

To get the dynamic programming recursion, note that the expression for the cost can be factored as

$$
J(K_1, K_2) = \sum_{x_1^0} \left[ \sum_{x_2^0} p_2(x_2^0) c_0(x_1^0, x_2^0, K_{10}(x_1^0), K_{20}(x_2^0)) + \sum_{x_1^1} \left[ \sum_{x_2^1} p_2(x_2^1)p_1(x_1^1|x_1^0) c_1(x_1^1, x_2^1, K_{11}(x_1^1), K_{21}(x_2^1)) \right] \right]
$$

and thus the minimum value of the cost function is

$$
\min_{K_1, K_2} J(K_1, K_2) = \left[ \min_{x_1^0} \left[ \sum_{x_2^0} \left[ \sum_{x_1^1} \left[ \sum_{x_2^1} \left[ \cdots \right] \right] \right] \right] \right]
$$

with the optimizing $K_1, K_2$ on the right hand side being the optimal controllers. This leads to the following theorem:

**Theorem 2.** Define the value functions $V_t : X_t \times \mathbb{R} \rightarrow \mathbb{R}$ by the backward recursion

$$
V_N(x_1^N, \hat{p}_{2N}) = \min_{u_1^N \in U_1} \sum_{x_2^N} \min_{u_2^N(x_2^N) \in U_2} \sum_{x_1^N} \sum_{x_2^N} \hat{p}_{2N}(x_2^N) c_N(x_1^N, x_2^N, u_1^N, u_2^N(x_2^N))
$$

and

$$
V_t(x_1^t, \hat{p}_{2t}) = \min_{u_1^t \in U_1} \sum_{x_2^t} \min_{u_2^t(x_2^t) \in U_2} \sum_{x_1^t} \sum_{x_2^t} \hat{p}_{2t}(x_2^t) c_t(x_1^t, x_2^t, u_1^t, u_2^t(x_2^t))
$$
Then an optimal controller \((K_1, K_2)\) can be constructed inductively as follows: For each fixed \(x_t^N \in X_t^N\),

1. Set \(t = 0\) and \(\hat{p}_{20}(x_{20}) = p_{20}(x_{20})\) for all \(x_{20} \in X_2\).
2. Set \(K_{1l}(x_t^1)\) and \(K_{2l}(x_t^1, \cdot)\) to be an optimizing \(u_{1l} \in U_l\) and \(u_{2l} \in U_l^{X_2}\) needed to compute the value function \(V_l(x_{1l}, \hat{p}_{2l})\). If \(t = N\), then we’re done.
3. Set
\[
\hat{p}_{2l+1}(\cdot) = \sum_{x_{2l}} \hat{p}_{2l}(x_{2l})p_{2l+1}(\cdot|x_{1l}, x_{2l}, t, K_{1l}(x_t^1), K_{2l}(x_t^1, x_{2l}))
\]
(3)

4. Increment \(t\) and go back to step 2.

**Proof.** Suppose we fix \(t, x_t^1, K_{1l}(x_t^1)\), and \(K_{2l}(x_t^1, x_{2l})\) for all \(i < t\). Then by a simple backward inductive argument, the \((t+1)\)th outermost minimization in (1) is precisely \(V_l(x_{1l}, \hat{p}_{2l})\), where
\[
\hat{p}_{2l}(x_{2l}) = \sum_{x_{2l}^{-1}} \hat{p}_{2}(x_{2l}|x_{1l}, x_{2l}^{-1}, K_{2l}(x_t^1, x_{2l}^-))
\]

Thus if \(K_{1l}(x_t^1)\) and \(K_{2l}(x_t^1, \cdot)\) are chosen optimally for all \(i < t\), then choosing \(K_{1l}(x_t^1) \in U_l(1)\) and \(K_{2l}(x_t^1, \cdot) \in U_l^{X_2}\) to be the minimizers in evaluating the value function \(V_l(x_{1l}, \hat{p}_{2l})\) will also be optimal. Moreover, from the formulas of \(\hat{p}_{2l}\), it follows immediately that
\[
\hat{p}_{20}(x_{20}) = p_{20}(x_{20})
\]
\[
\hat{p}_{2l+1}(x_{2l+1}) = \sum_{x_{2l}} \hat{p}_{2l}(x_{2l})p_{2l+1}(\cdot|x_{1l}, x_{2l}, t, K_{1l}(x_t^1), K_{2l}(x_t^1, x_{2l}))
\]
so the theorem is proved.

Theorem 2 gives the algorithm that solves for the optimal decentralized controllers \(K_1\) and \(K_2\). In an actual implementation of the algorithm, we precompute and store the value functions with a piecewise-linear representation, but we compute on-the-fly only the portions of the controller as needed for the optimal actions. In particular, at time \(t\), we only need to compute \(K_{1l}(X_t^1)\) and \(K_{2l}(X_t^1, \cdot)\).

The value function \(V_l(x_{1l}, \hat{p}_{2l})\) defined by the theorem has an important interpretation—it is the minimum cost from time \(t\) to \(N\), given that \(X_{1t} = x_{1l}\) and \(\hat{p}_{2l}\) is the conditional distribution of \(X_{2t}\) given \(X_{1t}\), i.e. \(\hat{p}_{2l}\) is player 1’s knowledge of player 2’s state at time \(t\). The value function recursion in (2) also has a insightful interpretation. It expresses the current value function as optimal sum of the expected cost at time \(t\) given the conditional distribution for \(X_{2t}\), plus the expected cost of the next value function given the conditional distribution for \(X_{2,t+1}\).

It is important to know that even though player 2 knows both players’ states, the optimal strategy for him is not to simply apply the optimal centralized policy for player 2. Indeed, the value function recursion in (2) shows that player 2 must consider how his policy on this step affects player 1’s knowledge of his state on the next time step.

Because of these extra considerations taken by player 2, we in general need to simultaneously solve for the optimal actions for player 2 at each possible state in \(X_2\). This algorithm is feasible only if \(|U_2|^{|X_2|}\) is not too large. The simplest way to deal with this problem is to restrict our search for \(u_{2l}\) in (2) over a reasonably-sized subset \(S \subset U_2^{X_2}\). The idea is that even if \(|U_2^{X_2}|\) is large, there is often only a small subset of \(U_2^{X_2}\) make sense to apply, such as control-limit or monotone policies. Even if we can’t prove that the optimal choice for \(u_{2l}\) always resides in \(S\), the controller will still be optimal in the following restricted sense—the cost function is minimized for all controllers \(K_1\) and \(K_2\) subject to the constraint that \(K_{2l}(x_t^1, \cdot) \in S\) for all \(t < N\) and \(x_t^1\).

**Computing the Value Functions**

The value functions can be precomputed using well-known methods for solving centralized POMDPs. Because the value function recursion in (2) is slightly different from the standard form for centralized POMDPs, we include a brief primer on how to do this computation here.

The basic idea, due to Smallwood and Sondik (1973), is to notice that for each fixed \(x_{1t}, V_l(x_{1t}, \hat{p}_{2l})\) is a piecewise-linear-concave (PWLC) function of the form \(\min_i(a_i T p)\), and thus can be stored as the vectors \(a_i\). This is clearly true for \(t = N\). Moreover, if \(V_{t+1}(x_{1t+1}, \hat{p}_{2,t+1})\) is PWLC for all \(x_{1t+1}\), then it is clearly true by (2) that \(V_l(x_{1l}, \hat{p}_{2l})\) will be PWLC, if we show the following trivial lemma:

**Lemma 3.** Suppose \(a_1, \ldots, a_m, b_1, \ldots, b_n \in \mathbb{R}^k\), and \(Q \in \mathbb{R}^{k \times k}\). Then

1. \(\min_i(a_i^T Q p) = \min_i((Q^T a_i)^T p)\).
2. \(\min_i(a_i^T p) + \min_j(b_j^T p) = \min_{i,j}(a_i + b_j)^T p\).
3. \(\min_i \{a_i^T p, b_j^T p\} = \min_{i,j} \{a_i^T p, b_j^T p\}\).

Note that the addition of PWLC functions can dramatically increase the number of vectors representing the functions. It is therefore important to keep the representations as efficient as possible, pruning vectors that are unnecessary for the representation. A simple procedure for pruning the PWLC function \(\min_{i=1}^m \{a_i^T p\}\) is as follows:

1. Add vectors to the selected set, \(I\): Start with \(I = \{1\}\). Then for each \(i = 2, \ldots, m\), solve the linear program
   
   minimize \(a_i^T p - t\) subject to \(a_j^T p \geq t\) for all \(j \in I\)
   \[p \geq 0, \quad 1^T p = 1\]
   If the optimal value is negative, then set \(I = I \cup \{i\}\).

2. Prune vectors from the selected set, \(I\): For each \(i \in I - \{\max I\}\), solve the linear program
   
   minimize \(a_i^T p - t\) subject to \(a_j^T p \geq t\) for all \(j \in I - \{i\}\)
   \[p \geq 0, \quad 1^T p = 1\]
   If the optimal value is nonnegative, then set \(I = I - \{i\}\).
It is wise to run the pruning process after each addition in (2), as well as once after the minimization. More details can be found in Cassandra, Litman, and Zhang (1997).

Even with the pruning process, it is still possible for the representation of the value functions to get very large. In fact, Papadimitrrou and Tsitsiklis (1987) show that even with stationary transition functions, solving finite-horizon centralized POMDPs is PSPACE-complete. In other words, POMDPs are some of the hardest problems to solve in polynomial space. The main problem lies with the number of states (in our case, the number of states for player 2), since this determines the dimension of the PWLC functions. Nevertheless, exact algorithms for solving POMDPs are still useful when number of states is small, and there are recent algorithms that can approximately solve POMDPs with very large number of states—see for example Kurniawati, Hsu, and Lee (2008).

**Examples**

**Machine Replacement**

Consider the problem of designing a replacement policy for two machines, where

1. $X_{1t}$ and $X_{2t}$ denote the operating states of machines 1 and 2 at time $t$, and are nonnegative integer random variables with maximum values of $n_1$ and $n_2$, respectively. If $X_{it} = n_i$, this means that machine $i$ has failed at time $t$.

2. At each time $t$, we have a choice of whether to replace a machine or not. Let $U_{1t}$ and $U_{2t}$ indicate this choice for machines 1 and 2, where a value of 1 indicates to replace and 0 indicates to not replace.

3. $p_1(x_{1,t+1} | x_{1t}, u_{1t})$ and $p_2(x_{2,t+1} | x_{2t}, u_{2t})$ denote the transition functions for machines 1 and 2, respectively. Also, we assume that $p_i(\cdot | x_{it}, 1) = p_i(\cdot | 0, 0)$, i.e. the damage distribution one time step after replacement is the same as the damage distribution after operating a fresh machine for one time step.

4. $c_1(x_{1t}, u_{1t})$ and $c_2(x_{2t}, u_{2t})$ denote the individual costs for machines 1 and 2 during time period $[t, t+1)$. In particular, $c_i(x_{it}, 0)$ indicates the operating cost given machine $i$ has damage state $x_{it}$, while $c_i(x_{it}, 1)$ indicates the expected cost of replacing the machine with a fresh one and operating it for one time period, minus any rewards for recovering the old machine at a damage state $x_{it}$.

5. The machines are linked serially, so that if either one of the machines is being replaced or has failed, then the entire system incurs a fixed downtime cost of $D$. This provides an incentive for the machines to work together to replace at the same time. Moreover, the total cost for the system at time $t$ is

$$c(X_{1t}, X_{2t}, U_{1t}, U_{2t}) = c_1(X_{1t}, U_{1t}) + c_2(X_{2t}, U_{2t}) + D 1_{\{X_{it} = n_i \text{ or } U_{it} = 1 \text{ for any } i\}}$$

6. At time $t$, machine 1 has access to $(X_{1t}^{t-1}, U_{1t}^{t-1})$, while machine 2 has access to $(X_{2t}^{t-1}, U_{2t}^{t-1})$. By Theorem 1, we can restrict ourselves to deterministic controllers of the form

$$U_{1t} = K_{1t}(X_{1t}^t)$$

$$U_{2t} = K_{2t}(X_{1t}^t, X_{2t}^t)$$

for some functions $K_{1t}$ and $K_{2t}$.

7. To reduce the burden of computing the value functions, we will constrain the controller for player 2 to be a *control-limit policy*, so that for each fixed $x_{1t}^t$, there is a threshold $x_{2t}^*(x_{1t}^t)$ such that

$$K_{2t}(x_{1t}^t, x_{2t}) = \begin{cases} 0, & x_{2t} < x_{2t}^*(x_{1t}^t) \\ 1, & x_{2t} \geq x_{2t}^*(x_{1t}^t) \end{cases}$$

This reduces the set of possible $K_{2t}(x_{1t}^t, \cdot)$ from $2^{|X_2|}$ members to $|X_2| + 1$ members.

The goal is to choose controllers $K_1 = (K_{10}, \ldots, K_{1N})$ and $K_2 = (K_{20}, \ldots, K_{2N})$ to minimize the expected cost in the time interval $[0, N + 1)$, i.e.

$$J(K_1, K_2) = \sum_{t=0}^N E[c(X_{1t}, X_{2t}, U_{1t}, U_{2t})]$$

As a numerical example, suppose that the transition functions are

$$p_1(\cdot | 1, 0) = \begin{bmatrix} 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2 & 0.2 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.2 & 0.2 & 0.4 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0.2 & 0.2 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0.1 & 0.1 & 0.2 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.2 & 0.4 & 0.6 & 1 \end{bmatrix}$$

$$p_2(\cdot | 1, 0) = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0.5 & 0 & 0 & 0 & 0 \\ 0.2 & 0.3 & 0.5 & 0 & 0 & 0 \\ 0 & 0.2 & 0.3 & 0.5 & 0 & 0 \\ 0 & 0 & 0.2 & 0.3 & 0.5 & 0 \\ 0 & 0 & 0 & 0.2 & 0.3 & 1 \end{bmatrix}$$

Thus the machines have maximum damages of $n_1 = 7$ and $n_2 = 5$, respectively, and the *incremental* damage suffered between time steps is i.i.d. The individual costs of the machines are

$$c_1 = \begin{bmatrix} 0 & 5 \\ 0 & 5 \\ 0 & 5 \\ 0 & 5 \\ 0 & 5 \\ 15 & 20 \end{bmatrix}, \quad c_2 = \begin{bmatrix} 0 & 5 \\ 0 & 5 \\ 0 & 5 \\ 0 & 5 \end{bmatrix}$$

Thus there is no operating cost for an unbroken machine, a fixed cost of 5 for replacing a machine, and large penalty of 15 if a machine breaks. We assume a system downtime cost of $D = 5$. The time horizon is $N + 1 = 17$. 

The performance gap is small. For example, when the centralized and decentralized policies are compared, it is clear that at least for these problem parameters, the performance gap is small. For example, when the centralized policy is applied, the per-period cost is 3.714 for the centralized policy and 3.812 for the decentralized policy.

Figure 2 shows the performance gap between the centralized and decentralized policies. To provide a fair comparison, we assume that in the decentralized case, player 1 has perfect knowledge of player 2’s state only at time \( t = 0 \).

Figure 2: Performance gap between centralized and decentralized replacement policies.

The lower and upper black dots show the per-period cost of the centralized and decentralized policies, respectively. It is clear that at least for these problem parameters, the performance gap is small. For example, when \( X_{10} = X_{20} = 0 \), the per-period cost is 3.714 for the centralized policy and 3.812 for the decentralized policy.

We mentioned earlier that even though player 2 knows both states, the optimal decentralized policy for player 2 is not the same as the centralized policy. This problem is no exception. For example, if

\[
X_{10} = 3, \quad \hat{p}_{20} = [0.01 \ 0.02 \ 0.05 \ 0.1 \ 0.6 \ 0.22]^T
\]

then the optimal actions at time \( t = 0 \) are to replace machine 1, and to replace machine 2 if \( X_{20} \geq 2 \). This clearly differs from the optimal centralized policy displayed on Figure 1, which is to replace machine 2 if \( X_{20} \geq 4 \). In fact, applying the first actions yields a total cost of 83.012, while applying the second ones yields a total cost of 83.644. In hindsight, it makes sense to use the lower threshold because player 1’s belief about player 2 (however incorrect) compels him to replace machine 1, so player 2 has an incentive to take advantage of the downtime to replace machine 2. Thus simply using a centralized policy for player 2 fails to take into account player 1’s limited knowledge of player 2.

### Two Queues in Series

Figure 3 shows an admission control system for two queues connected in series, where

1. \( X_{1t} \) and \( X_{2t} \) denote the number of jobs in queues 1 and 2 at time \( t \). Queues 1 and 2 are finite and can hold a maximum of \( n_1 \) and \( n_2 \) jobs, respectively.

2. \( W_{1t} \) and \( W_{2t} \) denote the number of new jobs arriving at queues 1 and 2 at time \( t \). \( W_{1t} \) is an i.i.d. process where the probability mass function of \( W_{1t} \) is \( p_{W1} \).

3. \( U_{1t} \) and \( U_{2t} \) denote the number of new jobs actually admitted into queues 1 and 2 right after time \( t \). Thus the number of jobs in the queues right after time \( t \) are \( X_{1t} + U_{1t} \) and \( X_{2t} + U_{2t} \). Arrivals that are not admitted are discarded.

4. \( Y_{1t} \) and \( Y_{2t} \) denote the potential number of jobs served in queues 1 and 2 during the time slot \((t, t + 1]\). The actual number of jobs served are \( \min\{Y_{1t}, X_{1t} + U_{1t}\} \) and \( \min\{Y_{2t}, X_{2t} + U_{2t}\} \), respectively, so that jobs served are limited by the number of jobs in the queue. \( Y_{1t} \) and \( Y_{2t} \) have probability mass functions of \( p_{Y1} \) and \( p_{Y2} \), respectively, and are i.i.d. processes independent of each other and of the arrival process \( W_{1t} \).

5. The completed jobs from queue 1 become the new arrivals for queue 2, so that

\[
W_{2,t+1} = \min\{Y_{1t}, X_{1t} + U_{1t}\}
\]

6. At time \( t + 1 \), the number of jobs remaining in queue \( i \) is

\[
X_{i,t+1} = (X_{it} + U_{it} - Y_{it})^+
\]
and so the transition function for queue $i$ is

$$p_i(x_{i,t+1} | x_{it} + u_{it})$$

$$= \left\{ \begin{array}{ll}
    \sum_{y_{it} \geq x_{i,t} + u_{it}} p_{yi}(y_{it}), & x_{i,t+1} = 0 \\
    p_{yi}(x_{it} + u_{it} - x_{i,t+1}), & x_{i,t+1} \geq 1
\end{array} \right.$$ for each job that gets serviced by both queues. Moreover, for each time period $(t, t+1)$, there is a holding cost of $h_1(z_1) + h_2(z_2)$, where $z_1$ and $z_2$ are the number of jobs in queues 1 and 2 right after time $t$. Thus given $X_{1t} + U_{1t} = z_1$ and $X_{2t} + U_{2t} = z_2$, the expected reward obtained in time period $(t, t+1)$ is

$$RE[\min\{Y_{2t}, z_2\}] - h_1(z_1) - h_2(z_2)$$

where we define $r_2(z_2) = RE[\min\{Y_{2t}, z_2\}] - h_2(z_2)$. $r_2(z_2)$ denotes the expected reward earned by queue 2 during the period $(t, t+1)$ given that the number of jobs in the queue right after time $t$ is $z_2$.

8. At time $t$, queue 1 has access to $(X_{1t}^1, W_{1t}^1, W_{2t}^1, U_{1t}^{t-1})$, while queue 2 has access to $(X_{1t}^2, W_{1t}^2, X_{2t}^2, U_{1t}^{t-1})$. By Theorem 1, we can restrict ourselves to deterministic controllers of the form

$$U_{1t} = K_{1t}(X_{1t}^1, W_{1t}^1, W_{2t}^2)$$

$$U_{2t} = K_{2t}(X_{1t}^2, W_{1t}^2, W_{2t}^2, X_{2t})$$

for some functions $K_{1t}$ and $K_{2t}$.

9. To save computation, we only consider control-limit policies for player 2, so that for each $(x_{1t}^1, w_{1t}^1, w_{2t}^2)$, there is a threshold $x_{2t}^*(x_{1t}^1, w_{1t}^1, w_{2t}^2)$ such that

$$K_{2t}(x_{1t}^1, w_{1t}^1, w_{2t}^2, x_{2t}) = \min \{x_{2t}^*(x_{1t}^1, w_{1t}^1, w_{2t}^2) - x_{2t}, w_{2t}^2\}$$

In other words, admit jobs into queue 2 until we reach the threshold $x_{2t}^*(x_{1t}^1, w_{1t}^1, w_{2t}^2)$. Let $S(w_{2t})$ denote the set of allowable $K_{2t}(x_{1t}, w_{1t}, w_{2t}, \cdot)$.

The goal is to choose $K_1 = (K_{10}, \ldots, K_{1N})$ and $K_2 = (K_{20}, \ldots, K_{2N})$ to maximize the expected $(N+1)$-period reward

$$J(K_1, K_2) = \sum_{t=0}^{N} (E[r_2(X_{2t} + U_{2t})] - E[h_1(X_{1t} + U_{1t})])$$

Using (2), the value function recursion for this problem is

$$V_N(x_{1N}, w_{1N}, w_{2N}, \hat{p}_{2N}) =$$

$$\sum_{x_{2N}} \max_{u_{2N}(x_{2N}) \leq w_{2N}} r_2(x_{2N} + u_{2N}(x_{2N}))$$

$$- \min_{u_{1N} \leq u_{2N}} h_1(x_{1N} + u_{1N})$$

$$V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t}) =$$

$$\max_{u_{1t} \leq w_{1t}} \left[ \sum_{x_{2t}} \hat{p}_{2t}(x_{2t}) r_2(x_{2t} + u_{2t}(x_{2t})) \right.$$

$$- h_1(x_{1t} + u_{1t})$$

$$+ \sum_{x_{2t}} \hat{p}_{2t}(x_{2t}) p_1(x_{1t+1} | x_{1t} + u_{1t}) w_1(w_{1t+1}) \left] \right.$$}

$$\sum_{x_{2t}}$$

We can simplify the computation of the value functions if we note that (5) takes the form

$$V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t}) =$$

$$\max_{u_{1t} \leq w_{1t}} Q_i(x_{1t} + u_{1t}, w_{1t}, \hat{p}_{2t})$$

where $Q_i(x_{1t} + u_{1t}, w_{1t}, \hat{p}_{2t})$ is just the quantity inside the brackets in (5). We can thus compute the value function recursion as follows:

1. Using the PWLC representation of $V_{t+1}$, compute the PWLC representations of $Q_i(z_1, u_{2t}, \hat{p}_{2t})$ for all $z_1 \leq \{0, \ldots, n_1\}$ and allowable $u_{2t}$.

2. Set $V_i(x_{1t}, 0, 0, \hat{p}_{2t}) = Q_i(x_{1t}, 0, \hat{p}_{2t})$ for all $x_{1t} \leq n_1$.

3. For each fixed $x_{1t}$, compute the PWLC representations of $V_i(x_{1t}, 0, w_{2t}, \hat{p}_{2t})$ using the recursion

$$V_i(x_{1t}, 0, w_{2t}, \hat{p}_{2t}) =$$

$$\max \left\{ V_i(x_{1t}, 0, w_{2t} - 1, \hat{p}_{2t}), \max_{u_{1t} \leq w_{1t}} Q_i(x_{1t}, u_{1t}, \hat{p}_{2t}) \right\}$$

Since $V_i(x_{1t}, 0, w_{2t}, \hat{p}_{2t}) = V_i(x_{1t}, n_2, \hat{p}_{2t})$ for all $w_{2t} \geq n_2$, the computation can stop when $w_{2t} = n_2$.

4. For each fixed $x_{1t}$ and $w_{2t}$, compute the PWLC representations of $V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t})$ using the recursion

$$V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t}) = \max \{ V_i(x_{1t} + w_{1t}, 0, w_{2t}, \hat{p}_{2t}), \}$

$$\max_{u_{1t} \leq w_{1t}} Q_i(x_{1t}, u_{1t}, \hat{p}_{2t}) \}$

Since $V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t}) = V_i(x_{1t}, n_2, \hat{p}_{2t})$ for all $w_{1t} \geq n_1 - x_{1t}$, the computation can stop when $w_{1t} = n_1 - x_{1t}$.

This method allows one to compute $V_i(x_{1t}, w_{1t}, w_{2t}, \hat{p}_{2t})$ for all $(x_{1t}, w_{1t}, w_{2t})$ with essentially the same complexity as computing $V_i(0, n_1, n_2, \hat{p}_{2t})$. 

7
As a numerical example, consider the case both queues have a capacity of $n_1 = n_2 = 5$ jobs, and when
\[
\begin{align*}
  p_{W1} &= [0.36 \ 0.36 \ 0.18 \ 0.1] \\
  p_{Y1} &= [0.2 \ 0.6 \ 0.2] \\
  p_{Y2} &= [0.3 \ 0.4 \ 0.3] \\
  h_1 = h_2 &= [0 \ 1 \ 2 \ 3 \ 4 \ 5] \\
  R &= 12
\end{align*}
\]
The time horizon is $N + 1 = 7$. The optimal per-period rewards for the centralized and decentralized cases assuming $X_{10} = X_{20} = W_{20} = 0$ are given in the following table:

<table>
<thead>
<tr>
<th>$W_{10}$</th>
<th>Centralized</th>
<th>Decentralized</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.2535</td>
<td>3.2466</td>
</tr>
<tr>
<td>1</td>
<td>4.3270</td>
<td>4.3170</td>
</tr>
<tr>
<td>2</td>
<td>4.6809</td>
<td>4.6654</td>
</tr>
<tr>
<td>3</td>
<td>4.6809</td>
<td>4.6654</td>
</tr>
</tbody>
</table>

Conclusions and Further Work

To summarize, we considered a general two-player finite-horizon decentralized MDP where player 1 has access to only its own states, and player 2 has access to the both states but cannot affect player 1. We found a dynamic programming recursion which allows us to solve the problem using centralized POMDP methods. The key result that enables the practicality of the method is Theorem 1, which states that the optimal controller for player 2 only needs to be a deterministic function of player 1’s history and player 2’s current state. Without this simplification, solving the problem would be hopelessly complex.

As shown by dynamic programming recursion in Theorem 2, the optimal controller takes player 1’s current state as well as its belief of player 2’s current state and solves for two things simultaneously: player 1’s optimal action, and the set of player 2’s optimal actions for each possible states. The crucial insight gained by the recursion is that even though player 2’s knows the entire state of the system, his optimal strategy is not to simply apply the optimal centralized policy. The reason is that he still needs to take into account how his policy (known by both players) will affect player 1’s belief of his state on the next time step.

We then applied this dynamic programming solution to decentralized versions of replacement and queueing problems. In order to solve the problems practically, we constrained the controllers for player 2 to be control-limit policies. For at least the problem instances we considered, the optimal decentralized policies performed within 1-2 percent of the optimal centralized policies.

Future work includes showing under what conditions control-limit or monotone policies for player 2 are optimal, and extensions to more general network topologies.

References


