F Problems

This section covers some of the decision problems used throughout this book. Table F.1 summarizes some of the important properties of these problems.

Table F.1. Problem summary. The `DecisionMakingProblems.jl` package implements these problems.

Problem	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$	γ
Hex world	—	varies	6	—	0.9
2048	—	∞	4	—	1
Cart-pole	—	$(\subset \mathbb{R}^4)$	2	—	1
Mountain car	—	$(\subset \mathbb{R}^2)$	3	—	1
Simple regulator	—	$(\subset \mathbb{R})$	$(\subset \mathbb{R})$	—	1 or 0.9
Aircraft collision avoidance	—	$(\subset \mathbb{R}^3)$	3	—	1
Crying baby	—	2	3	2	0.9
Machine replacement	—	3	4	2	1
Catch	—	4	10	2	0.9
Prisoner's dilemma	2	—	2 per agent	—	1
Rock-paper-scissors	2	—	3 per agent	—	1
Traveler's dilemma	2	—	99 per agent	—	1
Predator-prey hex world	varies	varies	6 per agent	—	0.9
Multicaregiver crying baby	2	2	3 per agent	2 per agent	0.9
Collaborative predator-prey hex world	varies	varies	6 per agent	2 per agent	0.9

F.1 Hex World

The *hex world problem* is a simple MDP in which we must traverse a tile map to reach a goal state. Each cell in the tile map represents a state in the MDP. We can attempt to move in any of the six directions. The effects of these actions are stochastic. As shown in figure F.1, we move 1 step in the specified direction with a probability of 0.7, and we move 1 step in one of the neighboring directions, each with a probability of 0.15. If we bump against the outer border of the grid, then we do not move at all, at a cost of 1.0.

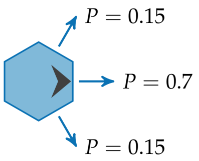

Figure F.1. Actions in the hex world problem have probabilistic effects.

Taking any action in certain cells gives us a specified reward and then transports us to a terminal state. No further reward is received in the terminal state. The total number of states in the hex world problem is thus the number of tiles plus 1, for the terminal state. Figure F.2 shows an optimal policy for two hex world problem configurations used throughout this book. We refer to the larger instance as "hex world" and to the smaller, simpler instance as "straight-line hex world."[1] The straight-line hex world formulation is used to illustrate how reward is propagated from its single reward-bearing state on the rightmost cell.

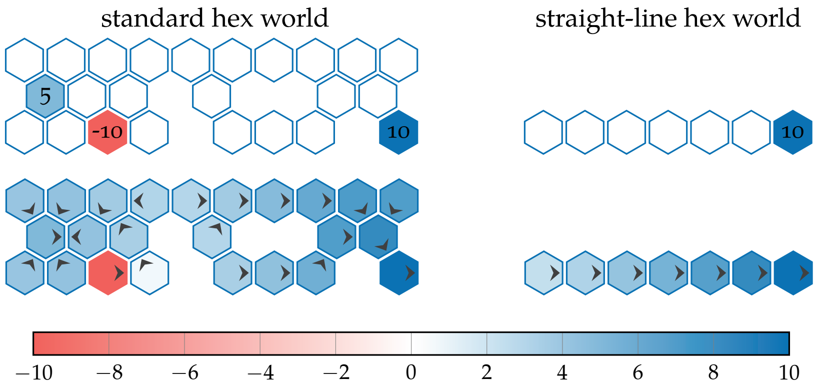

Figure F.2. The standard hex world and straight-line hex world problems. The top row shows the base problem setup and colors hexes that have terminal rewards. The bottom row shows an optimal policy for each problem, colored according to the expected value, with arrows indicating the action to take in each state.

F.2 2048

The *2048 problem* is based on a popular tile game played on a 4 × 4 board.[2] It has discrete state and action spaces. The board is initially empty except for two tiles, each of which can have value 2 or 4. A randomly selected starting state is shown in figure F.3.

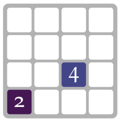

Figure F.3. A random starting state in the 2048 problem consists of two tiles, each with value 2 or 4.

The agent can move all tiles `left`, `down`, `right`, or `up`. Choosing a direction pushes all the tiles in that direction. A tile stops when it hits a wall or another tile of a different value. A tile that hits another tile of the same value merges with that tile, forming a new tile with their combined value. After shifting and merging, a new tile of value 2 or 4 is spawned in a random open space. This process is shown in figure F.4.

The game ends when we can no longer shift tiles to produce an empty space. Rewards are obtained only when merging two tiles, and they are equal to the

[1] The straight-line formulation is similar to the *hall problem*, a common benchmark MDP. See, for example, L. Baird, "Residual Algorithms: Reinforcement Learning with Function Approximation," in *International Conference on Machine Learning* (*ICML*), 1995.

[2] This game was developed by Gabriele Cirulli in 2014.

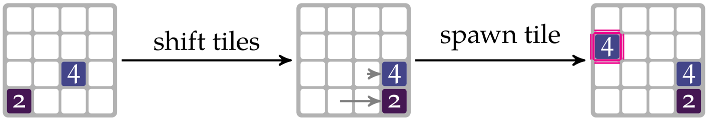

Figure F.4. An action in 2048 shifts all tiles in the chosen direction and then spawns a new tile in an empty space.

merged tile's value. An example state-action transition with a merge is shown in figure F.5.

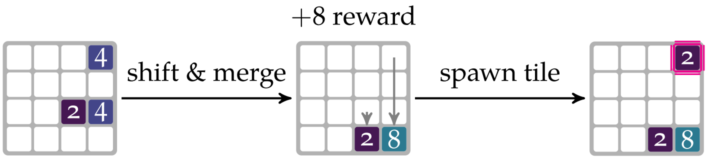

Figure F.5. Here, the `down` action is used to shift all tiles, resulting in the merging of two 4 tiles to produce an 8 tile and receive a reward of 8.

A common strategy is to choose a corner and alternate between the two actions that lead in that direction. This tends to stratify the tiles such that the larger-valued ones are in the corner and the newly spawned tiles are in the periphery.

F.3 Cart-Pole

The *cart-pole problem*,[3] also sometimes called the *pole balancing problem*, has the agent move a cart back and forth. As shown in figure F.6, this cart has a rigid pole attached to it by a swivel, so that as the cart moves back and forth, the pole begins to rotate. The objective is to keep the pole vertically balanced while keeping the cart within the allowed lateral bounds. As such, 1 reward is obtained each time step in which these conditions are met, and transition to a terminal zero-reward state occurs whenever they are not.

The actions are to either apply a left or right force F to the cart. The state space is defined by four continuous variables: the lateral position of the cart x, its lateral velocity v, the angle of the pole θ, and the pole's angular velocity ω. The problem involves a variety of parameters, including the mass of the cart m_{cart}, the mass of the pole m_{pole}, the pole length ℓ, the force magnitude $|F|$, gravitational acceleration g, the time step Δt, the maximum x deviation, the maximum angular deviation, and friction losses between the cart and the pole or between the cart and its track.[4]

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[4] We use the parameters implemented in the OpenAI Gym. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016. arXiv: 1606.01540v1.

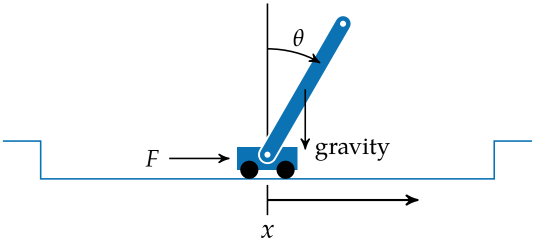

Figure F.6. In the cart-pole problem, a vehicle must alternate between accelerating left and right in order to balance a pole. The pole is not allowed to fall past a given angle, and the cart is not allowed to travel outside of given limits.

Given an input force F, the angular acceleration of the pole is

$$\alpha = \frac{g\sin(\theta) - \tau\cos(\theta)}{\frac{\ell}{2}\left(\frac{4}{3} - \frac{m_{\text{pole}}}{m_{\text{cart}}+m_{\text{pole}}}\cos(\theta)^2\right)} \tag{F.1}$$

where

$$\tau = \frac{F + \omega^2 \ell \sin\theta/2}{m_{\text{cart}} + m_{\text{pole}}} \tag{F.2}$$

and the lateral cart acceleration is

$$a = \tau - \frac{\ell}{2}\alpha\cos(\theta)\frac{m_{\text{pole}}}{m_{\text{cart}} + m_{\text{pole}}} \tag{F.3}$$

The state is updated with Euler integration:

$$\begin{aligned} x &\leftarrow x + v\Delta t \\ v &\leftarrow v + a\Delta t \\ \theta &\leftarrow \theta + \omega\Delta t \\ \omega &\leftarrow \omega + \alpha\Delta t \end{aligned} \tag{F.4}$$

The cart-pole problem is typically initialized with each random value drawn from $U(-0.05, 0.05)$. Rollouts are run until the lateral or angular deviations are exceeded.

F.4 Mountain Car

In the *mountain car problem*,[5] a vehicle must drive to the right, out of a valley. The valley walls are steep enough that blindly accelerating toward the goal with insufficient speed causes the vehicle to come to a halt and slide back down. The agent must learn to accelerate left first in order to gain enough momentum on the return to make it up the hill.

[5] This problem was introduced in A. Moore, "Efficient Memory-Based Learning for Robot Control," Ph.D. dissertation, University of Cambridge, 1990. Its popular, simpler form, with a discrete action space, was first given in S. P. Singh and R. S. Sutton, "Reinforcement Learning with Replacing Eligibility Traces," *Machine Learning*, vol. 22, pp. 123–158, 1996.

The state is the vehicle's horizontal position $x \in [-1.2, 0.6]$ and speed $v \in [-0.07, 0.07]$. At any given time step, the vehicle can accelerate left ($a = -1$), accelerate right ($a = 1$), or coast ($a = 0$). We receive -1 reward every turn, and terminate when the vehicle makes it up the right side of the valley past $x = 0.6$. A visualization of the problem is given in figure F.7.

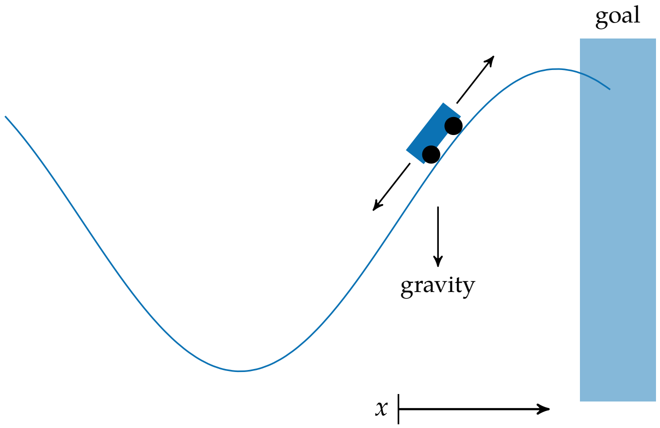

Figure F.7. In the mountain car problem, a vehicle must alternate between accelerating left and right in order to power itself up a hill. The goal region is shown in blue.

Transitions in the mountain car problem are deterministic:

$$
\begin{aligned}
v' &\leftarrow v + 0.001a - 0.0025\cos(3x) \\
x' &\leftarrow x + v'
\end{aligned}
$$

The gravitational term in the speed update is what drives the underpowered vehicle back toward the valley floor. Transitions are clamped to the bounds of the state space.

The mountain car problem is a good example of a problem with delayed return. Many actions are required to get to the goal state, making it difficult for an untrained agent to receive anything other than consistent unit penalties. The best learning algorithms are able to efficiently propagate knowledge from trajectories that reach the goal back to the rest of the state space.

F.5 Simple Regulator

The *simple regulator problem* is a simple linear quadratic regulator problem with a single state. It is an MDP with a single real-valued state and a single real-valued action. Transitions are linear Gaussian, such that a successor state s' is drawn from the Gaussian distribution $\mathcal{N}(s + a, 0.1^2)$. Rewards are quadratic, $R(s, a) = -s^2$, and do not depend on the action. The examples in this book use the initial state distribution $\mathcal{N}(0.3, 0.1^2)$.

Optimal finite horizon policies cannot be derived using the methods from section 7.8. In this case, $\mathbf{T}_s = [1]$, $\mathbf{T}_a = [1]$, $\mathbf{R}_s = [-1]$, $\mathbf{R}_a = [0]$, and w is drawn from $\mathcal{N}(0, 0.1^2)$. Applications of the Riccati equation require that $\mathbf{R}_a$ be negative definite, which it is not.

The optimal policy is $\pi(s) = -s$, resulting in a successor state distribution centered at the origin. In the policy gradient chapters, we learned parameterized policies of the form $\pi_{\boldsymbol{\theta}}(s) = \mathcal{N}(\theta_1 s, \theta_2^2)$. In such cases, the optimal parameterization for the simple regulator problem is $\theta_1 = -1$ and θ_2 is asymptotically close to zero.

The optimal value function for the simple regulator problem is also centered about the origin, with reward decreasing quadratically:

$$\begin{aligned} U(s) &= -s^2 + \frac{\gamma}{1-\gamma} \mathbb{E}_{s \sim \mathcal{N}(0, 0.1^2)}\left[-s^2\right] \\ &\approx -s^2 - 0.010 \frac{\gamma}{1-\gamma} \end{aligned}$$

F.6 Aircraft Collision Avoidance

The *aircraft collision avoidance problem* involves deciding when to issue a climb or descend advisory to an aircraft to avoid an intruder aircraft.[6] There are three actions corresponding to no advisory, commanding a 5 m/s descend, and commanding a 5 m/s climb. The intruder is approaching us head on, with a constant horizontal closing speed. The state is specified by the altitude h of our aircraft measured relative to the intruder aircraft, our vertical rate $\dot{h}$, the previous action a_{prev}, and the time to potential collision t_{col}. Figure F.8 illustrates the problem scenario.

Given action a, the state variables are updated as follows:

$$h \leftarrow h + \dot{h}\Delta t \tag{F.5}$$

$$\dot{h} \leftarrow \dot{h} + (\ddot{h} + \nu)\Delta t \tag{F.6}$$

$$a_{\text{prev}} \leftarrow a \tag{F.7}$$

$$t_{\text{col}} \leftarrow t_{\text{col}} - \Delta t \tag{F.8}$$

[6] This formulation is a highly simplified version of the problem described by M. J. Kochenderfer and J. P. Chryssanthacopoulos, "Robust Airborne Collision Avoidance Through Dynamic Programming," Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371, 2011.

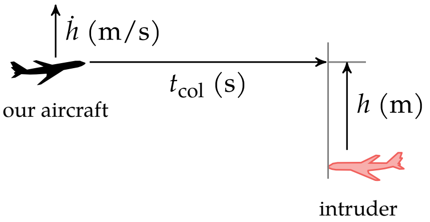

Figure F.8. State variables for the aircraft collision avoidance problem.

where $\Delta t = 1\,\text{s}$ and ν is selected from a discrete distribution over -2, 0, or $2\,\text{m/s}^2$ with associated probabilities 0.25, 0.5, and 0.25. The value $\ddot{h}$ is given by

$$\ddot{h} = \begin{cases} 0 & \text{if } a = \text{no advisory} \\ a/\Delta t & \text{if } |a - \dot{h}|/\Delta t < \ddot{h}_{\text{limit}} \\ \text{sign}(a - \dot{h})\ddot{h}_{\text{limit}} & \text{otherwise} \end{cases} \tag{F.9}$$

where $\ddot{h}_{\text{limit}} = 1\,\text{m/s}^2$.

The episode terminates when taking an action when $t_{\text{col}} < 0$. There is a penalty of 1 when the intruder comes within 50 m when $t_{\text{col}} = 0$, and there is a penalty of 0.01 when $a \neq a_{\text{prev}}$.

The aircraft collision avoidance problem can be efficiently solved over a discretized grid using backward induction value iteration (section 7.6) because the dynamics deterministically reduce t_{col}. Slices of the optimal value function and policy are depicted in figure F.9.

F.7 Crying Baby

The *crying baby problem*[7] is a simple POMDP with two states, three actions, and two observations. Our goal is to care for a baby, and we do so by choosing at each time step whether to feed the baby, sing to the baby, or ignore the baby.

The baby becomes hungry over time. We do not directly observe whether the baby is hungry; instead, we receive a noisy observation in the form of whether the baby is crying. The state, action, and observation spaces are as follows:

$$\mathcal{S} = \{\text{sated}, \text{hungry}\}$$
$$\mathcal{A} = \{\text{feed}, \text{sing}, \text{ignore}\}$$
$$\mathcal{O} = \{\text{crying}, \text{quiet}\}$$

[7] The version of the crying baby problem presented in this text is an extension of the original, simpler crying baby problem in M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

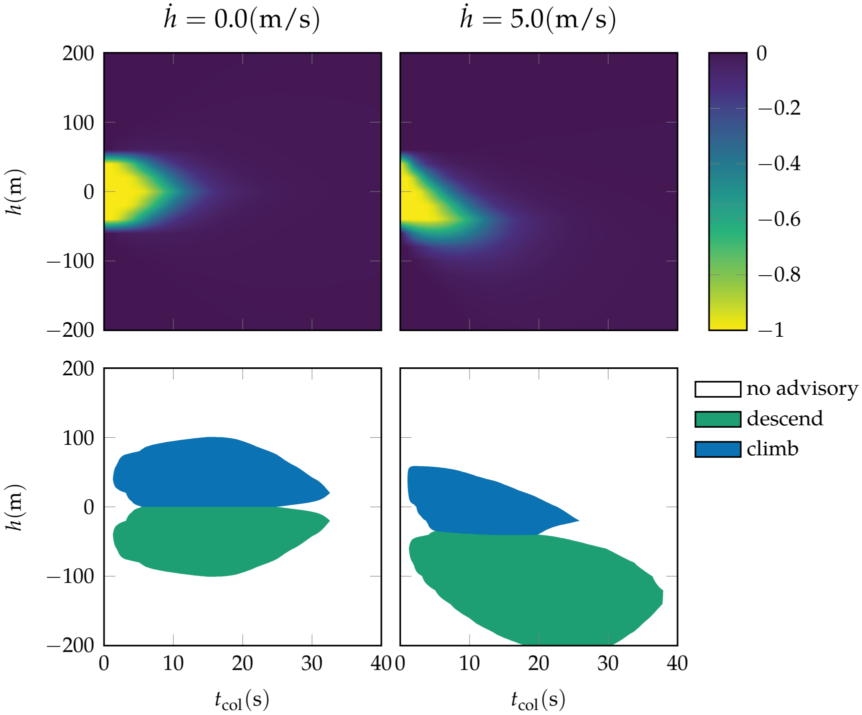

Figure F.9. Optimal value function slices (top) and policy slices (bottom) for the aircraft collision avoidance problem. The value function and policy are symmetric about 0 when the vertical separation rate is 0, but are skewed when the vertical separation rate is nonzero. Overall, our aircraft need not take action until the intruder aircraft is close.

Feeding will always sate the baby. Ignoring the baby risks a sated baby becoming hungry, and ensures that a hungry baby remains hungry. Singing to the baby is an information-gathering action with the same transition dynamics as ignoring, but without the potential for crying when sated (not hungry) and with an increased chance of crying when hungry.

The transition dynamics are as follows:

$$\begin{aligned}
T(\text{sated} \mid \text{hungry}, \text{feed}) &= 100\,\% \\
T(\text{hungry} \mid \text{hungry}, \text{sing}) &= 100\,\% \\
T(\text{hungry} \mid \text{hungry}, \text{ignore}) &= 100\,\% \\
T(\text{sated} \mid \text{sated}, \text{feed}) &= 100\,\% \\
T(\text{hungry} \mid \text{sated}, \text{sing}) &= 10\,\% \\
T(\text{hungry} \mid \text{sated}, \text{ignore}) &= 10\,\%
\end{aligned}$$

The observation dynamics are as follows:

$$\begin{aligned}
O(\text{cry} \mid \text{feed}, \text{hungry}) &= 80\,\% \\
O(\text{cry} \mid \text{sing}, \text{hungry}) &= 90\,\% \\
O(\text{cry} \mid \text{ignore}, \text{hungry}) &= 80\,\% \\
O(\text{cry} \mid \text{feed}, \text{sated}) &= 10\,\% \\
O(\text{cry} \mid \text{sing}, \text{sated}) &= 0\,\% \\
O(\text{cry} \mid \text{ignore}, \text{sated}) &= 10\,\%
\end{aligned}$$

The reward function assigns -10 reward if the baby is hungry, independent of the action taken. The effort of feeding the baby adds a further -5 reward, whereas singing adds -0.5 reward. As baby caregivers, we seek the optimal infinite-horizon policy with discount factor $\gamma = 0.9$. Figure F.10 shows the optimal value function and associated policy.

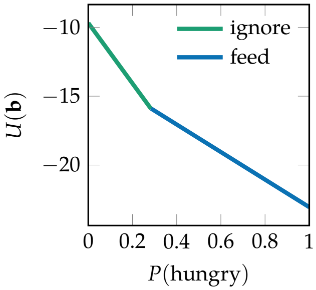

Figure F.10. The optimal policy for the crying baby problem. This infinite horizon solution does not recommend singing for any belief state. As shown in figure 20.3, it is optimal to sing in some finite-horizon versions of this problem.

F.8 Machine Replacement

The *machine replacement problem* is a discrete POMDP in which we maintain a machine that creates products.[8] This problem is used for its relative simplicity and the varied size and shape of the optimal policy regions. The optimal policy for certain horizons even has disjoint regions in which the same action is optimal, as shown in figure F.11.

[8] R. D. Smallwood and E. J. Sondik, "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973. The original problem formulation includes *salvage values*, or *terminal rewards* that are equal to the number of working parts. We do not model terminal rewards separately in this book. Terminal rewards could be included in our framework by explicitly including the horizon in the problem state.

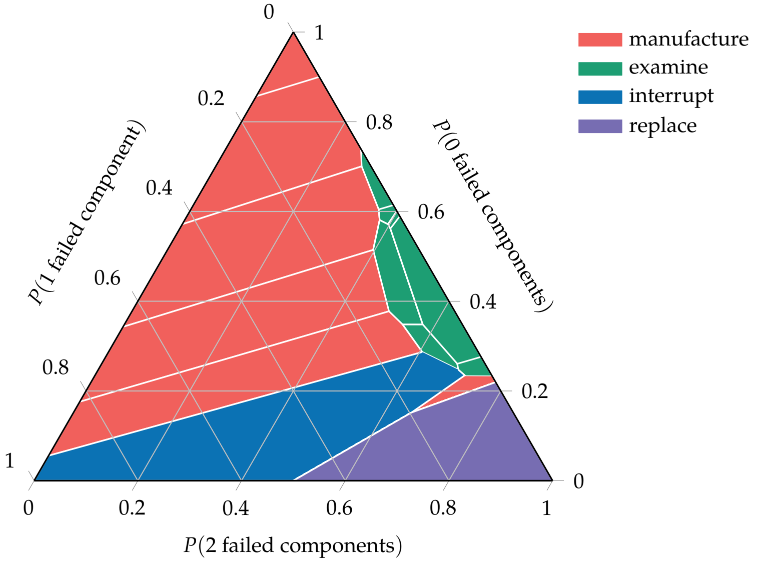

Figure F.11. The 14-step optimal policy for the machine replacement problem has disjoint regions where manufacturing is optimal. Each polygon corresponds to the region in which a particular alpha vector dominates.

The machine produces products for us when it is working properly. Over time, the two primary components in the machine may break down, together or individually, leading to defective products. We can indirectly observe whether the machine is faulty by examining the products, or by directly examining the components in the machine.

The problem has states $\mathcal{S} = \{0, 1, 2\}$, corresponding to the number of faulty internal components. There are four actions, used prior to each production cycle:

1. *manufacture*, manufacture product and do not examine the product,
2. *examine*, manufacture product and examine the product,
3. *interrupt*, interrupt production, inspect, and replace failed components, and
4. *replace*, replace both components after interrupting production.

When we examine the product, we can observe whether it is defective. All other actions observe only nondefective products.

The components in the machine independently have a 10 % chance of breaking down with each production cycle. Each failed component contributes a 50 % chance of creating a defective product. A nondefective product nets 1 reward, whereas a defective product nets 0 reward. The transition dynamics assume that component breakdown is determined before a product is made, so the manufacture action on a fully-functional machine does not have a 100 % chance of producing 1 reward.

The manufacture action incurs no penalty. Examining the product costs 0.25. Interrupting the line costs 0.5 to inspect the machine, causes no product to be made, and incurs a cost of 1 for each broken component. Simply replacing both components always incurs a cost of 2, but it does not have an inspection cost.

The transition, observation, and reward functions are given in table F.2. Optimal policies for increasing horizons are shown in figure F.12.

Action	$T(s' \mid s, a)$	$O(o \mid a, s')$	$R(s, a)$
manufacture	$\begin{array}{cc} & s' \\ s & \begin{bmatrix} 0.81 & 0.18 & 0.01 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$	$\begin{array}{cc} & o \\ s' & \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \end{array}$	$s \begin{bmatrix} 0.9025 \\ 0.475 \\ 0.25 \end{bmatrix}$
examine	$s \begin{bmatrix} 0.81 & 0.18 & 0.01 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{bmatrix}$	$s' \begin{bmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}$	$s \begin{bmatrix} 0.6525 \\ 0.225 \\ 0 \end{bmatrix}$
interrupt	$s \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$s' \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$	$s \begin{bmatrix} -0.5 \\ -1.5 \\ -2.5 \end{bmatrix}$
replace	$s \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$s' \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$	$s \begin{bmatrix} -2 \\ -2 \\ -2 \end{bmatrix}$

Table F.2. The transition, observation, and reward functions for the machine replacement problem.

F.9 Catch

In the *catch problem*, Johnny would like to successfully catch throws from his father, and he prefers catching longer-distance throws. However, he is uncertain about the relationship between the distances of a throw and the probability of a successful catch. He does know that the probability of a successful catch is the same, regardless of whether he is throwing or catching; and he has a finite number of attempted catches to maximize his expected utility before he has to go home.

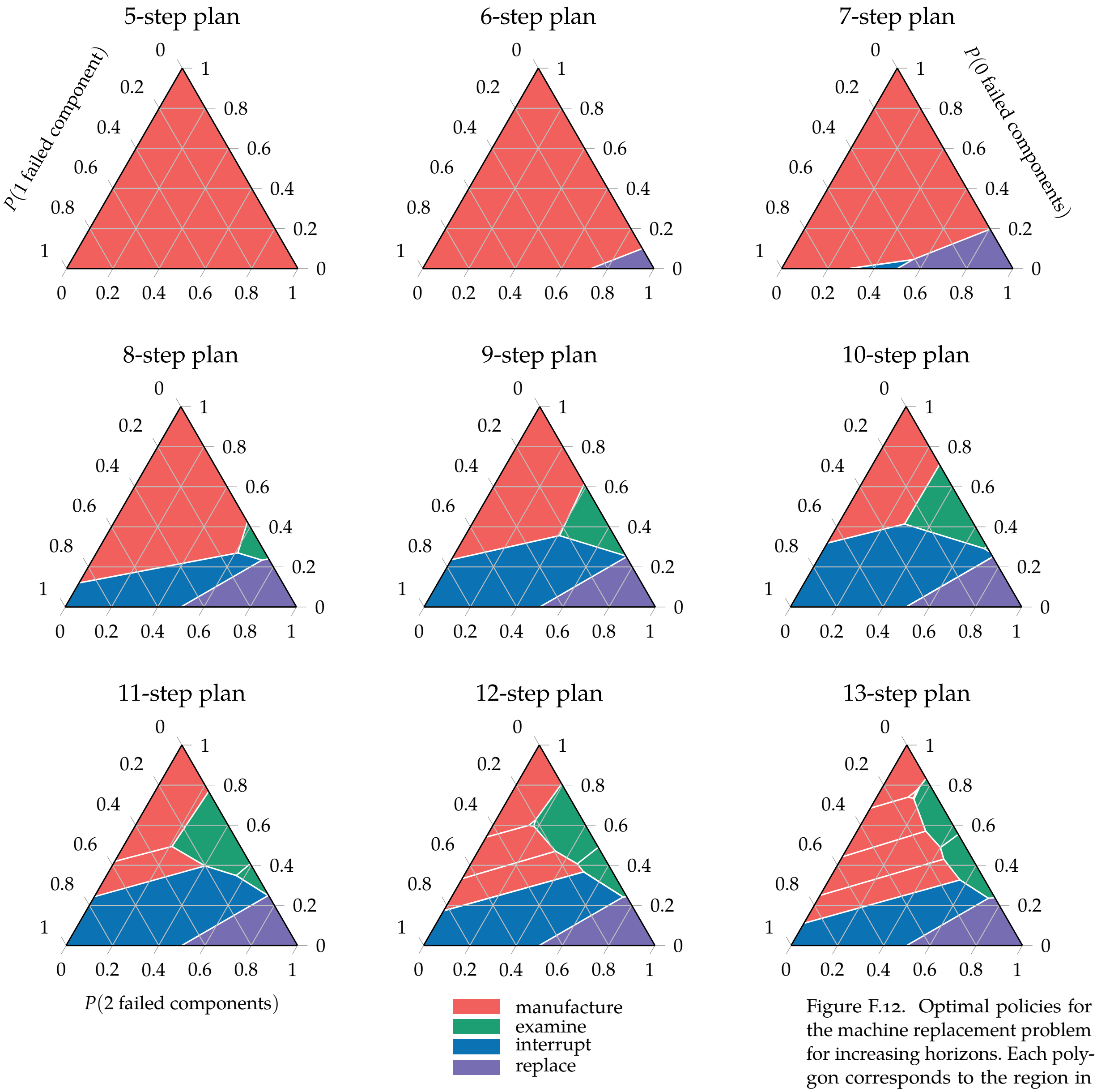

Figure F.12. Optimal policies for the machine replacement problem for increasing horizons. Each polygon corresponds to the region in which a particular alpha vector dominates.

As shown in figure F.13, Johnny models the probability of successfully catching a ball thrown a distance d as

$$P(\text{catch} \mid d) = 1 - \frac{1}{1 + \exp\left(-\frac{d-s}{15}\right)} \tag{F.10}$$

where the proficiency s is unknown and does not change over time. To keep things manageable, he assumes that s belongs to the discrete set $\mathcal{S} = \{20, 40, 60, 80\}$.

The reward for a successful catch is equal to the distance. If the catch is unsuccessful, then the reward is zero. Johnny wants to maximize the reward over a finite number of attempted throws. With each throw, Johnny chooses a distance from a discrete set $\mathcal{A} = \{10, 20, \ldots, 100\}$. He begins with a uniform distribution over $\mathcal{S}$.

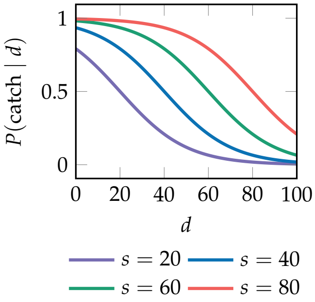

Figure F.13. The catch probability as a function of throw distance d for the four proficiencies in $\mathcal{S}$.

F.10 Prisoner's Dilemma

The *prisoner's dilemma* is a classic problem in game theory involving agents with conflicting objectives. There are two prisoners that are on trial. They can choose to cooperate, remaining silent about their shared crime, or defect, blaming the other for their crime. If they both cooperate, they both serve time for one year. If agent i cooperates and the other agent $-i$ defects, then i serves four years and $-i$ serves no time. If both defect, then they both serve three years.[9]

The game has two agents, $\mathcal{I} = \{1, 2\}$ and $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$, with each $\mathcal{A}^i = \{\text{cooperate}, \text{defect}\}$. The table in figure F.14 expresses the individual rewards. Rows represent actions for agent 1. Columns represent actions for agent 2. The rewards for agent 1 and 2 are shown in each cell: $R^1(a^1, a^2)$, $R^2(a^1, a^2)$. The game can be played once or repeated any number of times. In the infinite horizon case, we use a discount factor of $\gamma = 0.9$.

F.11 Rock-Paper-Scissors

One common game played around the world is *rock-paper-scissors*. There are two agents who can each choose either rock, paper, or scissors. Rock beats scissors, resulting in a unit reward for the agent playing rock and a unit penalty for the agent playing scissors. Scissors beats paper, resulting in a unit reward for the agent playing scissors and a unit penalty for the agent playing paper. Finally,

[9] A. W. Tucker gave the name to this game and formulated the story. It was based on the original problem formulation of Merrill Flood and Melvin Dresher at RAND in 1950. A history is provided by W. Poundstone, *Prisoner's Dilemma*. Doubleday, 1992.

		agent 2	
		cooperate	defect
agent 1	cooperate	$-1,-1$	$-4,0$
	defect	$0,-4$	$-3,-3$

Figure F.14. The rewards associated with the prisoner's dilemma.

paper beats rock, resulting in a unit reward for the agent playing paper and a unit penalty for the agent playing rock.

We have $\mathcal{I} = \{1, 2\}$ and $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2$ with each $\mathcal{A}^i = \{\text{rock}, \text{paper}, \text{scissors}\}$. Figure F.15 shows the rewards associated with the game, with each cell denoting $R^1(a^1, a^2)$, $R^2(a^1, a^2)$. The game can be played once or repeated any number of times. In the infinite horizon case, we use a discount factor of $\gamma = 0.9$.

		agent 2		
		rock	paper	scissors
agent 1	rock	$0,0$	$-1,1$	$1,-1$
	paper	$1,-1$	$0,0$	$-1,1$
	scissors	$-1,1$	$1,-1$	$0,0$

Figure F.15. The rewards associated with the rock-paper-scissors game.

F.12 *Traveler's Dilemma*

The *traveler's dilemma* is a game where an airline loses two identical suitcases from two travelers.[10] The airline asks the travelers to write down the value of their suitcases, which can be between \$2 and \$100, inclusive. If both put down the

[10] K. Basu, "The Traveler's Dilemma: Paradoxes of Rationality in Game Theory," *American Economic Review*, vol. 84, no. 2, pp. 391–395, 1994.

same value, then they both get that value. Otherwise, the traveler with the lower value gets their value plus $2 and the traveler with the higher value gets the lower value minus $2. In other words, the reward function is as follows:

$$R_i(a_i, a_{-i}) = \begin{cases} a_i & \text{if } a_i = a_{-i} \\ a_i + 2 & \text{if } a_i < a_{-i} \\ a_{-i} - 2 & \text{otherwise} \end{cases} \tag{F.11}$$

Most people tend to put down between $97 and $100. However, somewhat counterintuitively, there is a unique Nash equilibrium of only $2.

F.13 Predator-Prey Hex World

The *predator-prey hex world problem* expands the hex world dynamics to include multiple agents consisting of predators and prey. A predator tries to capture a prey as quickly as possible, and a prey tries to escape the predators as long as possible. The initial state of the hex world is shown in figure F.16. There are no terminal states in this game.

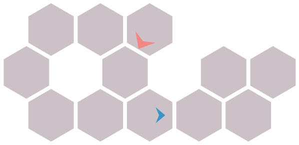

Figure F.16. The initial state in the predator-prey hex world. The predator is red and the prey is blue. The arrows indicate potential actions taken by the individual agents from their initial cells.

There is a set of predators $\mathcal{I}_{\text{pred}}$ and a set of prey $\mathcal{I}_{\text{prey}}$, with $\mathcal{I} = \mathcal{I}_{\text{pred}} \cup \mathcal{I}_{\text{prey}}$. The states contain the locations of each agent: $\mathcal{S} = \mathcal{S}^1 \times \cdots \times \mathcal{S}^{|\mathcal{I}|}$, with each $\mathcal{S}^i$ equal to all hex locations. The joint action space is $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^{|\mathcal{I}|}$, where each $\mathcal{A}^i$ consists of all six hex directions of movement.

If a predator $i \in \mathcal{I}_{\text{pred}}$ and prey $j \in \mathcal{I}_{\text{prey}}$ share the same hex with $s_i = s_j$, then the prey is devoured. The prey j is then transported to a random hex cell, representing its offspring appearing in the world. Otherwise, the state transitions are independent and are as described in the original hex world.

One or more predators can capture one or more prey if they all happen to be in the same cell. If n predators and m prey all share the same cell, the predators receive a reward of m/n. For example, if two predators capture one prey together, they each get a reward of $1/2$. If three predators capture five prey together, they each get a reward of $5/3$. Moving predators receive unit penalty. Prey can move with no penalty, but they receive a penalty of 100 for being devoured.

F.14 Multicaregiver Crying Baby

The *multicaregiver crying baby problem* is a multiagent extension of the crying baby problem. For each caregiver $i \in \mathcal{I} = \{1, 2\}$, the states, actions, and observations are as follows:

$$\mathcal{S} = \{\text{hungry}, \text{sated}\} \tag{F.12}$$

$$\mathcal{A}^i = \{\text{feed}, \text{sing}, \text{ignore}\} \tag{F.13}$$

$$\mathcal{O}^i = \{\text{crying}, \text{quiet}\} \tag{F.14}$$

The transition dynamics are similar to the original crying baby problem, except that either caregiver can feed to satisfy the baby:

$$T(\text{sated} \mid \text{hungry}, (\text{feed}, \star)) = T(\text{sated} \mid \text{hungry}, (\star, \text{feed})) = 100\,\% \tag{F.15}$$

where $\star$ indicates all possible other variable assignments. Otherwise, if the actions are not feed, then the baby transitions between sated to hungry as before:

$$T(\text{hungry} \mid \text{hungry}, (\star, \star)) = 100\,\% \tag{F.16}$$

$$T(\text{sated} \mid \text{sated}, (\star, \star)) = 50\,\% \tag{F.17}$$

The observation dynamics are also similar to the single-agent version, but the model ensures that both caregivers make the same observation of the baby, but not necessarily of each other's choice of caregiving action:

$$O((\text{cry}, \text{cry}) \mid (\text{sing}, \star), \text{hungry}) = O((\text{cry}, \text{cry}) \mid (\star, \text{sing}), \text{hungry}) = 90\,\% \tag{F.18}$$

$$O((\text{quiet}, \text{quiet}) \mid (\text{sing}, \star), \text{hungry}) = O((\text{quiet}, \text{quiet}) \mid (\star, \text{sing}), \text{hungry}) = 10\,\% \tag{F.19}$$

$$O((\text{cry}, \text{cry}) \mid (\text{sing}, \star), \text{sated}) = O((\text{cry}, \text{cry}) \mid (\star, \text{sing}), \text{sated}) = 0\,\% \tag{F.20}$$

If the actions are not to sing, then the observations are as follows:

$$O((\text{cry}, \text{cry}) \mid (\star, \star), \text{hungry}) = O((\text{cry}, \text{cry}) \mid (\star, \star), \text{hungry}) = 90\,\% \tag{F.21}$$

$$O((\text{quiet}, \text{quiet}) \mid (\star, \star), \text{hungry}) = O((\text{quiet}, \text{quiet}) \mid (\star, \star), \text{hungry}) = 10\,\% \tag{F.22}$$

$$O((\text{cry}, \text{cry}) \mid (\star, \star), \text{sated}) = O((\text{cry}, \text{cry}) \mid (\star, \star), \text{sated}) = 0\,\% \tag{F.23}$$

$$O((\text{quiet}, \text{quiet}) \mid (\star, \star), \text{sated}) = O((\text{quiet}, \text{quiet}) \mid (\star, \star), \text{sated}) = 100\,\% \tag{F.24}$$

Both caregivers want to help the baby when the baby is hungry, assigning the same penalty of -10.0 for both. However, the first caregiver favors feeding and the second caregiver favors singing. For feeding, the first caregiver receives an extra penalty of only -2.5, while the second caregiver receives an extra penalty of -5.0. For singing, the first caregiver is penalized by -0.5, while the second caregiver is penalized by only -0.25.

F.15 Collaborative Predator-Prey Hex World

The *collaborative predator-prey hex world* is a variant of the predator-prey hex world in which a team of predators chase a single, moving prey. The predators must work together to capture a prey. The prey moves randomly to a neighboring cell that is not occupied by a predator.

Predators also only make noisy local observations of the environment. Each predator i detects whether a prey is within a neighboring cell $\mathcal{O}^i = \{\text{prey}, \text{nothing}\}$. The predators are penalized with a -1 reward for movement. They receive a reward of 10 if one or more of them capture the prey, meaning that they are in the same cell as the prey. At this point, the prey is randomly assigned a new cell, signifying the arrival of a new prey for the predators to begin hunting again.