Computing Finite-Horizon Policies

Recall that for the optimal policy

$$V^{*T}(s) = \max_a [u(a,s) + \Sigma_{s'} \beta V^{*(T-1)}(s') P(a,s)(s')]$$

How long does it take to compute V^{*T}?

Let $\mathcal{R}(V^{*t})$ be the running time of the computation of V^{*t}.

Naively, to compute V^{*T}, need to compute, for each action $a \in A$

$$u(a,s) + \Sigma_{s'} \beta V^{*(T-1)}(s') P(a,s)(s'). \quad (*)$$

This involves computing $V^{*(T-1)}(s')$ for each $s' \in S$.
So it looks $\mathcal{R}(V^{*T}) \geq |A| \times |S| \times \mathcal{R}(V^{*(T-1)})$.

- Similarly, $\mathcal{R}(V^{*(T-1)}) \geq |A| \times |S| \times \mathcal{R}(V^{*(T-2)}$.
- This suggests that $\mathcal{R}(V^{*T}) \geq (|A| \times |S|)^T$!

This is infeasible even for moderately large T.

We can do better

- Key observation: a lot of computations are being repeated.

Idea:

- Consider an $S \times T$ matrix
  - the (s, t) entry is $V^{*t}(s)$
  - Fill in the matrix column by column, starting with $t = 1$.
  - To fill in $(s, t+1)$ entry requires looking only at the (s', t) entries; about $O(|A| \times |S|)$ steps
  - Total running time is roughly $O(|A||S|^2|T|)$
  - OK if S isn't too large ...

Computing Infinite-Horizon Policies

Value Iteration:

- each iteration takes $O(|A||S|^2)$ steps
- If $P(a, s)(s')$ is sparse, then it could be $O(|A||S|)$.
- The Contraction Theorem says that it will take roughly $|\epsilon|$ iterations to get with ϵ of optimal.
  - $|\epsilon|$ is the length of ϵ, written in binary
  - Each iteration gets a factor β improvement
  - The number of iterations required to reach an optimal solution is polynomial in $|S|$ and the largest rewards, for a fixed discount factor β
  - However, the number of iterations can grow polynomially in $1/(1-\beta)$
    - The convergence rate can be slow if β is close to 1

As long as β is not too close to 1, this is OK if $|S|$ isn't too large ...

Policy iteration:

- each iteration takes polynomial time
- There are clearly at most $|A|^{|S|}$ iterations required to reach an optimal policy
- Each iteration gives a strictly better policy
  - Open question: is there a polynomial bound?
- In practice, value iteration is much faster per iteration, but policy iteration seems to take fewer iterations.
- Many variants of value/policy iteration have been considered, that try to combine the best features of both.

In any case, even polynomial complexity may be infeasible.

- The state space may be huge!

Example: Call Blocking

Consider a wireless network:

- The world is divided into cells
- Each cell is handled by one base station
- Each base station has an upper bound on the bandwidth it can handle
  - Different types of calls may have different bandwidth requirements
  - Voice vs. video vs. data
- Should a new call be blocked?
  - There may not be enough capacity to carry it
  - May want to allow excess capacity for people coming over from neighboring cells
    - Blocking a new call is not as bad as dropping an ongoing call

What is the optimal policy for accepting calls?

It seems reasonable to model this as an MDP, but what are the states/actions/rewards?

- Is the probability that a new call arrives time-dependent?
  - If so, need to include time in the state
- The probability of a handoff typically depends on how many calls a neighbor is handling
  - This will typically not be known
  - It can perhaps be approximated by the number of calls at the current cell.
- And whose utility are we talking about anyway?
  - The user's?
  - The carriers?
- What are the actions?
  - Time passing?
  - Need to model the arrival of calls somehow.

One Model

One possibility:

- Assume K service classes/types of call
- Service class K requires b_k channels
- There are N channels altogether

State has the form $s = (x_1, \ldots, x_K, \theta)$:

- x_i is number of calls of class i;
- $\Sigma_{k=1}^{K} b_k x_k \leq N$;
- θ is the event that occurred at the last time step (if any):
  - new-call arrival of class i
  - handoff from neighboring cell of class i
  - call departure of class i call
  - $3K + 1$ possibilities

That means that if there are even three call classes, requiring 1, 2, and 3 channels respectively, and $N = 5,000$,

then there are $c(5000)^3 10$ states (for some c I'm too lazy to compute).

Note time is not in the state included, nor is information about neighboring cells:

- This only makes the size of the state space worse

May be able to deal with state-space size by analysis or heuristics:

- For the call-blocking problem, under appropriate assumptions, the optimal policy uses a threshold: if the number of calls in the system is below the threshold, accept; otherwise reject.
  - Can discover the threshold experimentally.
- In more general problems, may want to take into account structure in the state space
  - Use ideas of (in)dependency as in Bayesian networks/influence diagrams
- Could also consider heuristics; stay tuned ...

Actions:

- If the last event is a new-call arrival or handoff, can choose to accept or block

Suppose we consider the network provider's utility:

- One goal: make money
  - Is there a fixed cost per call, or does it depend on the length of the call?
- Another goal: keep customers happy
  - Can be captured by assigning a penalty to dropping/blocking a call.

Moral: Modeling things using MDPs is wonderful in principle not always so easy in practice:

- Problems include both that of constructing the model and complexity issues

POMDPs

We have assumed that the states are fully observable

- If the agent is in state s, he knows it

Can model an agent's uncertainty about the state by using *Partially observable Markov decision processes (POMDPs)*:

- states S, actions A, and transition probabilities $P_S(a, s)$ as in MDPs
- observations O
- observation probabilities $P_O(a, s, s')(o)$
  - probability of observing o if in state s action a is chosen and the resulting state is s'

POMDPs can be solved using essentially the same techniques as MDPs

- The agent now has a probability over states

Big problem: POMDPs are exponentially harder to solve than MDPs.

Reinforcement Learning

Up to now we have assumed

- the set of states is known
- the transition probabilities are known

Even ignoring complexity issues, there are other problems:

- The agent may not know the model
- The model may change over time

Big issue:

- Exploitation vs. exploration

Multi-Armed Bandit Problem

The classic exploitation vs. exploration problem is the multi-armed bandit.

A simple version has two arms:

- Each one pays off \$1 (or 0) on each pull
- They may differ on the probability of paying off
- The probability is unknown, but constant

Actions have no long-term effects.

Multi-armed bandits can be viewed as POMDPs:

- The true state is unknown and not observed
- The observations are the results of pulling the arms

Multi-Armed Bandit: Formal Analysis

Can use dynamic programming to analyze the multi-armed bandit, if we're willing to assume a prior probability on the payoff:

We want to model an internal state that captures has been learned:

- Suppose that (n_1, w_1, n_2, w_2) denotes a state where arm i has been pulled n_i times and returned \$1 w_i times, for $i = 1, 2$.
- Note that $n_1 + n_2$ is the number of trials.
- Given a prior $\Pr_0$, the $\rho_{n_i,w_i,i}$ be the posterior probability of arm i paying off, given n_i and w_i.
    - If $\Pr_0$ is uniform, $\rho_{n_i,w_i,i} = (w_i + 1)/(n_i + 1)$.

Note:

- $V^*(n_1, w_1, n_2, w_2) = 0$ if $n_1 + n_2 = T$ (the horizon)
- If $n_1 + n_2 < T$, then

$$\begin{aligned} & V^*(n_1, w_1, n_2, w_2) \\ = \; & \max(\rho_{n_1,w_1,1}(1 + V^*(n_1 + 1, w_1 + 1, n_2, w_2)) + \\ & \quad (1 - \rho_{n_1,w_1,1})V^*(n_1 + 1, w_1, n_2, w_2), \\ & \quad \rho_{n_2,w_2,2}(1 + V^*(n_1, w_1, n_2 + 1, w_2 + 1)) + \\ & \quad (1 - \rho_{n_2,w_2,2})V^*(n_1, w_1, n_2 + 1, w_2)) \end{aligned}$$

  - Pull an arm, and then proceed to perform the best action, based on the outcome.

Complexity: $O(|BS||A|)$.

- BS is the set of belief states
- $|BS| = O(T^4)$
- This is much better than the general problem for POMDPs, but still infeasible for large T.

Multi-Armed Bandit: Heuristics

An obvious heuristic: the greedy strategy

- pick the currently-best action
- Problem: an unlucky streak on a good arm can lock you into a suboptimal choice.
- Heuristic solution: be optimistic
  - Put a high prior on the probability of payoff being high.
  - This results in lots of experimentation before you give up the belief

Another obvious heuristic: randomize

- The probability of trying an arm depends on past history, but it goes to 0 slowly. E.g.:

$$\Pr(a_i) = \frac{e^{E(a_i)/T}}{e^{E(a_1)/T} + e^{E(a_2)/T}}$$

  - $E(a_i)$ is the expected reward for action a_i
  - T is the *temperature*, a parameter that measures willing to explore
  - The idea is to lower the temperature slowly

Lots of other heuristics have been considered.

The multi-armed bandit is a simple problem, because it is easy to assign a "measure of goodness" to an action (choice of arm), based on past history.

- In general, it is much harder to decide how relevant an action was to an outcome
  - One reason for superstitious behavior

Learning an Optimal Policy

MDPs do capture dependence of outcomes on actions.

- Problem: even if you believe that the world is best described by an MDP, how do you learn to play an optimal policy.

Note that you may not even know what the MDP is.

- You could try to learn it, by learning the state space and the transition probabilities.
- You could try to learn the optimal policy without learning (all of) the model.

Predicting Values

Define

$$Q(s,a) = u(a,s) + \beta \sum_{s' \in S} P(a,s)(s')V(s')$$

- The reward if you perform a in state s and then continue optimally (assuming V is accurate)

Value iteration repeatedly updates V, using

$$V'(s) = \max_a Q(s,a)$$

TD(0) (TD = *temporal difference*) uses the same idea to update, without knowing S.

- If the agent is in state s, performs action a, gets reward r, and moves to state s'

$$\begin{aligned} V(s) &:= V(s) + \alpha(r + \beta V(s') - V(s)) \\ &= (1-\alpha)V(s) + \alpha(r + \beta V(s')) \end{aligned}$$

  - Think of α as the learning rate
  - It is slowly decreased over time

If actions are chosen according to policy π, then TD(0) converges to V^π.

TD(λ)

TD(λ) generalizes TD(0). If the agent is in state s, performs action a, gets reward r, and moves to s'

$$V(u) := V(u) + \alpha(r + \beta V(s') - V(s))e(u)$$

- V is updated for *every* state u according to its *eligibility* $e(u) = \Sigma_{k=0}^{t}(\lambda\beta)^{t-k}\delta_{u,s_k}$
  - $s_0, \ldots, s_t$ is sequence of states visited
  - λ is a parameter (that's why it's TD(λ))
  - $\delta_{u,s_k} = 1$ if $u = s_k$; 0 otherwise
  - $e^t(u) = \begin{cases} \beta\lambda e^{t-1}(u) + 1 & \text{if } u \text{ is state at time } t \\ \beta\lambda e^{t-1}(u) & \text{otherwise} \end{cases}$
    - $e(u)$ measures how often and how recently u has been visited
    - $e(u)$ decays by λ unless u is visited
  - if $\lambda = 0$, get TD(0)

TD(λ) gets computationally more expensive as λ gets larger, but it tends to converge faster.

Learning Optimal Policies

Sarsa: the analogue of TD(0).

- If the agent is in state s, performs action a, gets reward r, and moves to state s'

$$\begin{aligned} Q(s,a) &:= Q(s,a) + \alpha(r + \beta Q(s',a') - Q(s,a)) \\ &= (1-\alpha)Q(s,a) + \alpha(r + \max_{a'} \beta Q(s',a')) \end{aligned}$$

Q-learning:

- If the agent is in state s, performs action a, gets reward r, and moves to state s'

$$Q(s,a) := Q(s,a) + \alpha(r + \beta \max_a Q(s',a) - Q(s,a))$$

There are also analogues of both these approaches to TD(λ).

Key idea:

- Given Q, choose action that is within some fixed ϵ of optimal.
  - If there is more than one, choose randomly

Polynomial-Time Convergence

Q-learning and TD(λ) guarantee convergence to optimal values (and, thus, to optimal policies that are *eventually optimal*), but there is no guarantee on the speed of convergence.

- A policy π is eventually optimal if, for all ϵ, there exists a T_ϵ such that from time T_ϵ on, π has value within ϵ of optimal (treating T_ϵ as time 0).
  - With discounting, we need to use a notion of optimality like this, to allow recovery from "youthful mistakes" made while exploring.

We can get polynomial-time convergence:

Theorem: [Kearns-Singh] Given ϵ, δ, and an MDP with discount rate β and maximum immediate reward $R_{\max}$, there exists an algorithm A polynomial in $1/\epsilon$, $1/\delta$, $|S|$, $1/(1-\beta)$, and $R_{\max}$ such that with probability at least $1-\delta$, A will halt in state s and output a policy π such that $V^\pi(s) \geq V^*(s) - \epsilon$.

Large State Spaces

Up to now, I have implicitly assumed that it is possible to enumerate the state and action spaces and store tables of values over them.

- Except in small environments, this is completely impractical.
- If the actions or states come from a continuous space, this cannot be done at all.

This problem is addressed using *generalization techniques*, which allow compact storage of learned information and transfer of knowledge between "similar" states and actions.

- neural networks are often used
- *hierarchical methods* attempt to view the problem as a hierarchy of learning problems (each with a much smaller state space)

Success Stories: TD Gammon

TD-Gammon [Tesauro] plays backgammon.

- Uses TD(λ) to evaluate board positions
- Approximately 10^{20} states
  - uses a neural network to approximate the value function
- Learned to play better by playing itself repeatedly
  - 1,500,000 training games for TD Gammon 3.0
- TD Gammon 3.0 is playing at world-championship level
  - barely lost to world champion in 1998

Success Story: Elevator dispatching

If you have n elevators in an m-floor building, what is the optimal dispatch policy?

- There are various criteria to optimize:
  - average waiting time before getting on elevator
  - average time before getting to destination
  - minimizing passengers waiting too long (e.g., > 60 seconds)

In practice, elevator dispatchers are designed heuristically and evaluated on simulated buildings.

Crites and Barto used Q-learning to design an elevator dispatcher on a 4-elevator 10-floor building.

- each elevator has position, direction, speed, set of buttons indicating who wants to get off where.
- There are over 10^{22} states
- Again, neural nets were used to approximate the value functions.

Their controller performed well compared to commercial dispatchers, by all measures.

- This suggests an automatic design to learning, with improvement over time.

Other applications

There have been lots of other applications of reinforcement learning:

- call blocking
- job shop scheduling
- juggling robotic acrobat
- ...

The Effect of Information

Up to now we have (implicitly) assumed that agents have perfect recall

- They may not have perfect information (e.g., in partially observable MDPs), but you do recall everything you've done and seen.

With perfect recall, optimal policies can always be taken to be deterministic.

- [Isbell]: Not necessarily so with imperfect recall.

What about time consistency? Of course, if preferences change over time, then what seemed optimal at time 0 may no longer seem optimal at time 1.

- Theorem [Kuhn 1953]: if you have perfect recall and preferences don't change, then there is no time inconsistency
- [Piccione and Rubinstein 1996]: Not true if you have imperfect recall??!!

The model

Consider finite-horizon games.

- Without loss of generality, associate with each complete sequence of states and actions just a final payoff.
  - Think of this as incorporating the immediate rewards and taking into account the discount factor

This puts as back into familiar territory: decision trees, with payoffs associated with the leaves.

- But now think of these dynamically, rather than statically
- The decision-maker (DM) is making a decision at each node in the tree
- Clearly, the decision maker's decision now should take into account what he will do in the future
  - it's better to go through a sequence of moves to get 3 later rather than taking 2 right away, provided you can count on your future actions.

Modeling Uncertainty

Think of the nodes in the decision tree as representing complete states of the world.

- Problem: the decision-maker may be uncertain as to the true state of the world.
- This is typically represented in the game theory literature by drawing ellipses in the decision tree
  - two nodes are inside the ellipse if the agent cannot tell which of them is the true world.
- (I will argue that this representation can be misguided.)

Example: In a two-armed bandit problem, one arm is known to pay off with probability .5; the other arm is known to pay off either with probability .8 or probability .4, but the agent doesn't know which.

- There are two initial states of the world: $(.5, .4)$ and $(.5., 8)$, giving the following decision tree:

Information Sets and Strategies

The set of states in a ellipse form an *information set*.

- They represent the agent's information
- In a POMDP, the sequence of observations represents the information set: the underlying set of possible true states of the world.

A *deterministic strategy* (or *policy*) is a function from information sets to actions:

- If an agent cannot tell two states apart, he must do the same thing at both states

A *mixed strategy* is a distribution over deterministic strategies.

A *behavior* (or *randomized*) *strategy* is a distribution over possible actions at each information set.

Time Consistency

A strategy s is *time consistent* if, whenever an agent reaches an information set X in the decision tree with positive probability, then the agent does not consider some other strategy s' to be better than s, given that he has reached X.

- That is, a strategy is time consistent if the agent is not tempted to change his mind after playing it for a while.

Perfect Recall

An agent has *perfect recall* if at every node in the decision tree he recalls the actions he took and what he knew.

- That means he can tell apart two nodes that were reached by different sequences of actions or via a different sequence of information sets.
- Formally, an agent has perfect recall (in a decision situation) if, for all nodes x and y in the decision tree, if x and y are in the same information set then the same actions were taken to get to x and to y, and the same information sets were gone through.

Theorem: [Kuhn, 1953] In a finite decision problem with perfect recall:

(a) every behavior strategy is equivalent to a mixed strategy

(b) every mixed strategy is equivalent to a behavior strategy

- "Equivalent" = same distribution on final outcomes

(c) there is an optimal deterministic strategy

(d) the optimal strategy is time consistent

Proof. (Sketch)

(a) A behavior strategy generates a distribution over sequences of actions.

- each sequence determines a deterministic strategy (ignoring unreached information sets), given perfect recall
- Thus, get a mixed strategy.

For (b), group strategies by common prefix

- e.g., all strategies with prefix a, d, b

For each prefix, consider probability of next action

- This determines a behavior strategy

For (c), suppose the optimal strategy is mixed/randomized.

- Just pick the best deterministic strategy

For (d), note that if you can do better at information set X which is reached from information set X', there is already a better strategy at X'.

What happens without perfect recall?

Part (a) fails if you don't remember your actions.

Example: Suppose that at step 1, you can go left or right. At step 2, you can go left or right again, but you forget what you did at step 1:

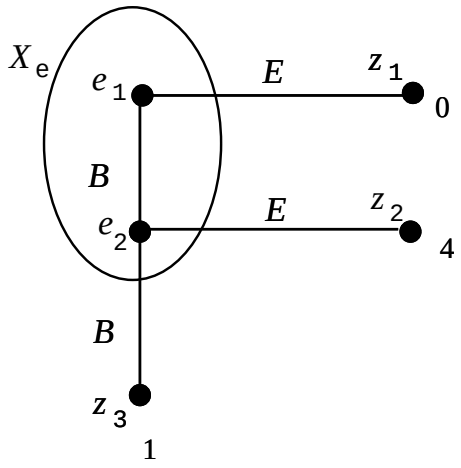
There is no behavior strategy equivalent to $\Pr(LL) = 1/2$, $\Pr(RR) = 1/2$

- What do you do at step 2?

Absentmindedness (a special case of imperfect recall) occurs when there are two points on the same path in the same information set.

Parts (b) and (c) fail with absentmindedness.

The absentminded driver game



[Piccione & Rubinstein, 1997]: To get home, an individual has to take the second exit off the highway.

- Turning at the second exit gives a reward of 4
- Turning at the first exit leads him to a bad part of town (payoff 0).
- If he misses the second exit he reaches a hotel where he can spend the night (payoff 1)

But . . . the driver is absentminded (and knows it).

- When he reaches an intersection, he's not sure if he's at the first or second exit.

What's the optimal strategy?

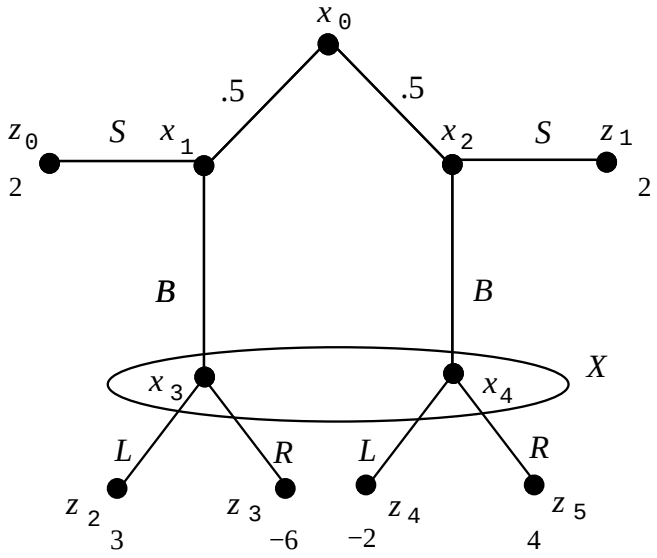
- The optimal deterministic strategy is to continue at each exit (payoff 1).
- Can do better with randomization.
  - The randomized strategy of continuing with probability 2/3 (and exiting with probability 1/3) does better:
    - With probability 4/9, get payoff of 1
    - With probability 2/9, get payoff of 4
    - With probability 1/3, get payoff of 0
    - Expected payoff: 4/3

    This randomized strategy is obviously not equivalent to a mixed strategy (since the only deterministic strategies are E and BB).
  - This strategy is in fact optimal
    - Want to choose α so as to maximize

$$\alpha^2 + 4\alpha(1-\alpha) = 4\alpha - 3\alpha^2$$

Time Consistency

Do we have time consistency with imperfect recall? Consider the left-or-right game [Piccione & Rubinstein]



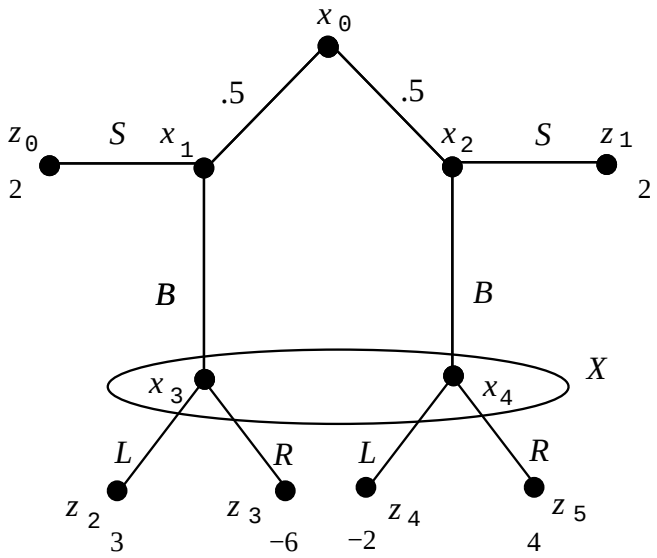
Strategy f chooses S at x_1, B at x_2, R at $\{x_3, x_4\}$.

- f is optimal

Strategy f' chooses B at x_1, S at x_2, L at $\{x_3, x_4\}$.

f seems time inconsistent:

- If you reach x_1, switching from f to f' is optimal



Claim: time inconsistency arises here because the representation does not correctly describe what's going on.

- You get time inconsistency *only if you can remember that you have switched.*
  - i.e., if you remember your strategy
  - i.e., provided you remember your strategy
- But if you remember your strategy, the information set does not correctly capture your information!

Time Absentminded Driver Again

Suppose the driver starts out using the deterministic strategy of playing B.

- When he reaches an intersection, according to PR, he places equal probability of being at intersection 1 and intersection 2. But then he should switch strategies!

Where is the 1/2 coming from?

- It's the probability of reaching e_1/e_2, conditional on reaching X_e (the info. set)
- If the strategy is played repeatedly, of the times that the driver is in X_e, half the time he is at e_1 and half the time he is at e_2
- But if he driver has the option of switching strategies, then it does not follow that this should be the probability of being at e_2.
- You need to consider how likely he is to think of switching strategies.

- If he is certain to consider it, then he will definitely be at intersection 1 when he considers it (he will never get to intersection 2).

If the driver follows the optimal protocol, according to PR, the probability of being at e_2, conditional on being at X_e, is $(2/3)/(1 + 2/3) = 2/5$.

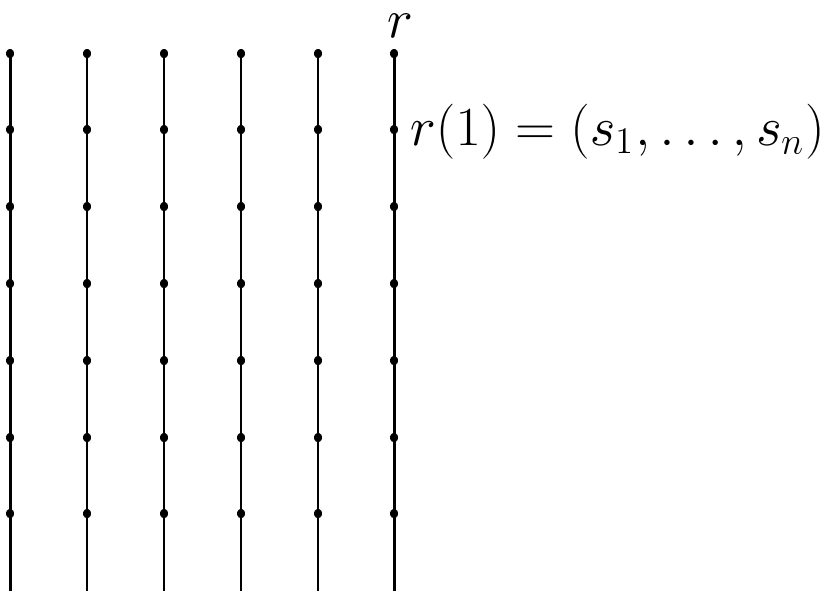
- If this were the appropriate probability, then the driver should switch strategies.
- But this is only correct if an adversary is stopping the play at random, and only once, and giving the driver the chance to switch.
  - This is a strange model of switching!

A Different Model

A *multi-agent system* [FHMV] consists of a collection of agents (player(s) in a game/processes/robots).

- Each agent has a *local* or *internal* state
  - Captures the agent's observations/information
  - Describes what's going on in the agent's head.
- The *environment* state captures other relevant features of the (external) world
  - E.g., where the agent is actually located
- The *global state* of the system is a tuple consisting of each agent's state and the environment state
- A *run* is a complete description of the system over time: a function from times to global states.
  - A run is essentially a path in the decision tree.
- A *system* is a set of runs.
  - Identifying system with its possible behaviors.

A system:



The set of points where an agent has the same local state can be viewed as an information set.

- But now the information set is not exogenously given as part of the problem description, but is determined from a description of the agent's information.
- If the agent remembers his strategy, then the strategy must be encoded in the local state.

Where do systems come from?

Actions are functions from global states to global states.

A *protocol* is a function from local states to (a probability distribution over) actions:

- what you do is a function of what you know.

A protocol is like a strategy, with local states playing the role of information sets.

Can consider system generated by a protocol: all the runs where the transitions are the ones determined by the protocol.

- Can put a distribution over this set of runs in the obvious way.

Given a decision tree (but *not* including the information sets), a description of the agent's observations, and a protocol/strategy, can construct an appropriate system.

- Must decide how to encode the agent's information.

- Must decide what the allowable protocols are.
  - Does the agent get to switch strategies?
- Must decide what the agent remembers?
  - Does he remember the strategies that were used?
- Under appropriate assumptions, the absentmindedness in the left-right game disappears.

Theorem [Halpern 1997]: There is no time inconsistency, if the problem is modeled correctly.

References

Standard text on MDPs (there are lots):

- M. L. Puterman, *Markov Decision Processes*, Wiley 1994.

Call-blocking example:

- Z. Haas, J. Y. Halpern, L. Li, S. Wicker A decision theoretic-approach to resource allocation in wireless multimedia networks, *Proc. Dial M for Mobility*, 2000

Reinforcement learning:

- R. S. Sutton, A. G. Barto, *Reinforcement Learning*, MIT Press, 1998.
- L. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: a survey, *Journal of AI Research* **4**, 1996
- M. Kearns, S. Singh, Near-optimal reinforcement learning in polynomial time, *Proc. 16th Int. Conf. on Machine Learning*, 1998.

Perfect Recall

- H. Kuhn, Extensive games and the problem of information, in *Contributions to the Theory of Games II*, Princeton U. Press, 1953.
- M. Piccione, A. Rubinstein, On the interpretation of decision problems with imperfect recall, *Games and Economic Behavior* **20**, 1997.
- J. Y. Halpern, On ambiguities in the interpretation of game trees, *Games and Economic Behavior* **20**, 1997.
- A. J. Grove and J. Y. Halpern, On the expected value of games with absentmindedness, *Games and Economic Behavior* **20**, 1997.