PROFESSOR: OK. Today we're going to finish up with Markov chains. And the last topic will be dynamic programming. I'm not going to say an awful lot about dynamic programming. It's a topic that was enormously important in research for probably 20 years from 1960 until about 1980, or 1990.

And it seemed as if half the Ph.D. theses done in the control area and the operations research were in this area. Suddenly, everything seemed to be done, could be done. And strangely enough, not many people seem to know about it anymore. It's an enormously useful algorithm for solving an awful lot of different problems. It's quite a simple algorithm.

You don't need the full power of Markov chains in order to understand it. So I do want to at least talk about it a little bit. And we will use what we've done so far with Markov chains in order to understand it. I want to start out today by reviewing a little bit of what we did last time about eigenvalues and eigenvectors.

This was a somewhat awkward topic to talk about, because you people have very different backgrounds in linear algebra. Some of you have a very strong background, some of you have almost no background. So it was a lot of material for those of you who know very little about linear algebra. And probably somewhat boring for those of you use it all the time.

At any rate, if you don't know anything about it, linear algebra is a topic that you ought to understand for almost anything you do. If you've gotten to this point without having to study it, it's very strange. So you should probably take some extra time out, not because you need it so much for this course. We won't use it enormously in many of the things we do later.

But you will use it so many times in the future that you ought to just sit down, not to learn abstract linear algebra, which is very useful also, but just to learn how to use the topic of solving linear equations. Being able to express them in terms of matrices. Being able to use the eigenvalues and eigenvectors, and matrices as a way of understanding these things.

So I want to say a little more about that today, which is why I've called this a review plus of eigenvalues and eigenvectors. It's a review of the topics we did last time, but it's looking at it in a somewhat different way. So let's proceed with that. We said that the determinant of an M by M matrix is given by this strange formula.

The determinant of a is the sum over all permutations of the integers 1 to M of the product from i equals 1 to M of the matrix element a sub i mu of i. Mu of i is the permutation of the number i. i is between one and M, and mu of i is a permutation of that. Now if you look at the matrix, which has the form, which is block upper diagonal.

In other words, there's a matrix here, a square matrix a sub t, which is a transient matrix. There's a recurrent matrix here, and there's some way of getting from the transient states to the recurring states. And this is the general form that a unit chain has to have. There are a bunch of transient states, there are a bunch of recurring states.

And the interesting thing here is that the determinant of a is exactly the determinant of a sub t times the determinant a sub r. I'm calling this a instead of the transition matrix p because I want to replace a by p minus lambda i, so I can talk about the eigenvalues of p.

So when I do that replacement here, if I know that the determinant of a is this product of determinants, then the determinant of p minus lambda i is the determinant of pt minus lambda it, where it is just a crazy way of saying a diagonal matrix. A diagonal t by t matrix, because this is a t by t matrix, also. i sub r is an r by r matrix, where this is a square r by r matrix also.

Now, why is it that this determinant is equal to this product of determinants here? Well, before explaining why this is true, why do you care? Well, because we know that if we have a recurring matrix here, we know that it has-- I mean, we know a great deal about it. We know that any square matrix, r by r matrix has r different eigenvalues.

Some of them might be repeated, but they're always r eigenvalues. This matrix here has t eigenvalues. OK. This matrix here, we know has r plus t eigenvalues. You look at this formula here and you say aha, I can take all the eigenvalues here, add them to all the eigenvalues here, and I have every one of the eigenvalues here.

In other words, if I want to find all of the eigenvalues of p, all I have to do is define the eigenvalues of p sub t, add them to the eigenvalues of p sub r, and I'm all done. So that really has simplified things a good deal. And it also really says explicitly that if you understand how to deal with recurrent Markov chains, you really know everything.

Well, you also have to know how to deal with a transient chain, but the main part of it is dealing with this chain. This has little r different eigenvalues, and all of those are eigenvalues, excuse me, p sub r has little r eigenvalues. They're given by the roots of this determinant here. And all of those are roots here. OK, so why is this true?

Well, the reason for it is that this product up here, look at this. We're taking the sum over all permutations. But which one of those permutations can be non-zero? If I start out by saying that a sub t is t by t, then I know that this might be anything. These have to be zeroes here. If I choose some permutation of down here, of sum i, which is greater than t.

In other words, if I choose mu o i to be some element over here. If I choose mu of i to be less than our equal to t, and i to be greater than t, what happens? I get a term which is equal to zero. That term in this product is zero. And none of those products can be zero. So the only way I can get non zeros here is when I'm dealing with an i which is less than or equal to t.

Namely an i here. I have to choose a mu of i, a column which is less than t, also. If I'm dealing with an i which is greater than t, namely and i up here, then, well, it looks like I can choose anything there. But look. I've already used up all of these columns here by the five by the non-zero terms here. So I can't do anything but use a smaller i, smaller than t up here.

So when I look at the permutations that are non zero, the only permutations that are non zero are those where mu of i is less than t if i less than t, and mu of i is less than or equal to t if i is less than or equal to t. And mu of i is greater than t if i is greater than t. Now, how does that show that this is equal here? Well, let's look at that a little bit.

I didn't even try to do it on the slide because the notation is kind of horrifying. But let's try to write this the following way. Determinant of a is equal to the sum, and now I'll write it as a sum over mu of 1 up to t. And the sum over mu of t plus 1 up to, well, t plus r, let's say. OK, so here I have all of the permutations of the numbers 1 to t.

And here I have all the permutations of the numbers t plus 1 up. And for all of those, I'm going to ignore this plus minus. You can sort that out for yourselves. And then I have a product from i equals 1 to t. And then a product from i equals t plus 1 up to m. Excuse me. i sub i, mu of i times product of a of i.

Mu of i for i equals t plus 1 up to t plus r. OK? So I'm separating this product here into a product first of the terms i less than or equal to t, and then for the terms i greater than t. For every permutation I choose using the i's that are less than or equal to t, I can choose any of the permutation using mu of i greater than t that I choose to use.

So this breaks up in this way. I have this sum, I have this sum. I have these two products, so I can break this up as a sum over mu of 1 to t of plus minus product from i equals 1 to t of ai, mu of i times the sum over mu of t plus 1 up to t plus r ai mu of i. Product. OK. So I've separated that into two different terms.

STUDENT: T equals [INAUDIBLE].

PROFESSOR: What?

STUDENT: T plus r equals big m?

PROFESSOR: T plus r is big m, yes. Because I have t terms here, and I have r terms here. OK, so the interesting thing here is having this non-zero term here doesn't make any difference here. I mean, this is more straightforward if you have a block diagonal matrix.

It's clear that the eigenvalues of a block diagonal matrix are going to be the eigenvalues of 1 plus the eigenvalues of the other. Here we have the eigenvalues of this, and the eigenvalues is this. And what's surprising is that as far as the eigenvalues are concerned, this has nothing whatsoever to do with it. OK.

The only thing that this has to do with it is it says something about the sums of this matrix here, because the sums of these rows are now less than 1. They all have to be, some of them, at least, have to be less than or equal to 1. Because you do have this way of getting from the transient elements to the non transient elements.

But it's very surprising that these elements, which are critically important, because those are the things that get you from the transition states to the recurrent states have nothing to do in the eigenvalues whatsoever. I don't know why. I can't give you any insights about that, but that's the way it is.

That's an interesting thing, because if you take this transition matrix, and you keep at and a sub r fixed, and you play any kind of funny game you want to with those terms going from the transient states to the non transient states, it won't change any eigenvalues. Don't know why it doesn't. OK, so where do we go with that? Well, that's what it says.

The eigenvalues of p, or the t eigenvalues of pt, and the r eigenvalues of PR. It also tells you something about simple eigenvalues, and these crazy eigenvalues, which don't have enough eigenvectors to go along with them. Because it tells you that a piece of r has all of its eigenvectors, and a piece of t has all of its eigenvectors. Then you don't have any of this crazy [INAUDIBLE] form thing, or anything.

OK If pi is a left eigenvector of this recurrent matrix, then if you look at the vector, starting was zeros, and then I guess I should really say, well, if pi sub 1 up to pi sub r as a left eigenvalue of this r by r matrix, then if I start out with t zeroes, and then put in pi 1 to pi r, this vector here has to be a left eigenvector of all of p. Why is that?

Well, if I look at a vector, which starts out with zeroes, and then has this eigenvector pi, and I multiply that vector by this matrix here, I'm taking these terms, multiplying them by the columns of this matrix, these zeros knock out all of these elements here. These zeroes knock out all of these elements. So I start out with zeroes everywhere here. That's what this says.

And then when I'm dealing with this part of the matrix, the zeros knock out all of this, and I just have pi multiplying piece of r. So if I have an eigenvalue lambda, it says I have the eigenvalue lambda times a vector zero times pi. It says that if I have an eigenvector, a left eigenvector of this recurrent matrix, then that turns into, if you put some zeroes up in front of, it turns into an eigenvector of the whole matrix.

If we look at the eigenvalue 1, which is the most important thing this, is the thing that gives you the steady state factor, this is sort of obvious. Because the steady state vector is where you go eventually, and eventually where you go is you have to be in one of these recurrent states, eventually.

And the probabilities within the recurrent set of states are the same as the probabilities if you didn't have this transient states at all. so this is all sort of obvious, as far as the steady state factor pi. But it's a little less obvious as far as the other vectors. The left eigenvectors, a piece of t, I don't understand them at all.

They aren't the same as the left eigenvectors of, well, the left eigenvectors of the eigenvalues of p sub t. I didn't say this right here. The left eigenvectors of p corresponding to the left eigenvectors of p sub t. I don't understand how they work, and I don't understand anything you can derive from them. They're just kind of crazy things, which are what they happen to be.

And I don't care about them. I don't know anything to do with them. But these other eigenvectors are very useful. OK. We can extend this to as many different recurrent sets of states as you choose. Here I'm doing it with a Markov chain, which has two different sets of recurrent states. They might be periodic, they might be ergodic, it doesn't make any difference.

So the matrix p has these transient states up here. Here we have those transition states would just go to each other, where the transition probabilities starting with the transient state and going to a transition state. Here we have the transitions, which go from transient states to this first set of recurrent states.

Here we have the transitions, which go from a transient state to the second state of recurrent states. OK. The same way as before, the determinant of this whole thing here, and this determinant, the roots of that are in fact the eigenvalues of p, are the product of the determinant of pt minus lambda it times the product of this, times this determinant here.

This has little t eigenvalues. This has little r eigenvalues. This has little r prime eigenvalues, and if you add up t plus little r plus little r prime, what do you get? You get jM, excuse me, capital M, which is the total number of states in the Markov chain.

So the eigenvalues here are exactly the eigenvalues here plus the eigenvalues here, plus the eigenvalues here. And you can find the eigenvectors, the left eigenvectors for these states in exactly the same way as before. OK. Yeah?

STUDENT: So again, the eigenvalues can be repeated both within t, r, r prime, and in between the--

PROFESSOR: Yes.

STUDENT: There's nothing that says [INAUDIBLE].

PROFESSOR: No. There's nothing that says they can't, except you can always find the left eigenvectors, anyway, of this are, in fact, these things in the form. If pi is a left eigenvector of p sub r, then zero followed by pi followed by zero. In other words, little t zeros followed by r, followed by the eigenvector pi, followed by little r prime zeroes here, this has to be a left eigenvector of t.

So this tells you something about whether you're going to have a Jordan form or not, one of these really ugly things in it. And it tells you that in many cases, you just can't have them. If you have them, they're usually tied up with this matrix here. OK, so that, I don't know. Was this useful? Does this clarify anything? Or if it doesn't, it's too bad. OK.

So now we want to start talking about rewards. Some people call these costs. If you're an optimist, you call it rewards. If you're a pessimist, you call it costs. They're both the same thing. If you're dealing with rewards, you maximize them. If you're dealing with costs, you minimize them. So mathematically, who cares?

OK, so suppose that each state i of a Markov chain is associated with a given reward, or a sub i. In other words, you think of this Markov chain, which is running along. You go from one state to another over time. And while this is happening, you're pocketing some reward all the time. OK. You invest in a stock. Strangely enough, these particular stocks we're thinking about here I this Markov property.

Stocks really don't have a Markov property, but we'll assume they do. And since they have this Markov property, you win for a while, and you lose for a while. You win for a while, you lose for a while. But we have something extra, other than just the Markov chains. We can analyze this whole situation, knowing how Markov chains behave.

There's not much left besides that, but there are an extraordinary number of applications of this idea, and dynamic programming is one of them. Because that's just one added extension beyond this idea of rewards. OK. The random variable x of n. That's a random quantity. It's the state at time n.

And the random reward of time n is then the random variable r of xn that maps xn equals i into ri for each i. This is the same idea of taking one random variable, which is a function of another random variable. The one random variable takes on the values one up to capital M.

And then the other random variable takes on a value which is determined by the state that you happen to be in, which is this random states. So specifying our sub i specifies what the set of rewards are, what the reward is in each given state. Again, we have this awful problem, which I wish we could avoid in Markov chains, of using the same word state to talk about the set of different states.

And also to talk about the random state at any given time. But hopefully by now you're used to that. In our discussion here, the only thing we're going to talk about are expected rewards. Now, you know that expected rewards, or expectations are a little more generally than you would think they would be, because you're going to take the expected value of any sort of crazy thing.

If you want to talk about any event, you can take the indicator function of that event, and find the expected value of that indicator function. And that's just the probability of that event. So by understanding how to deal with expectations, you really have the capability of finding distribution functions, or anything else you want to find. OK.

But anyway, since we're interested only in expected rewards, the expected reward at time n, given that x zero is i is the expected value of r of xn given x zero equals i, which is the sum over j of the reward you get if you're in state j at time n times p sub ij, super n, which we've talked about ad nauseum for the last four lectures now.

And this is the probability that the state at time n is j, given that the state at time zero is i. So you can just automatically find the expected value of r of xn. And it's by that formula. Now, recall that this quantity here is not all that simple. This is the ij element of the product of the matrix, of the nth product of the matrix p. But, so what? We can at least write a nice formula for it now.

The expected aggregate reward over the n steps from m to m plus n minus 1. What is m doing in here? It's just reminding us that Markov chains are homogeneous over time. So, when I talk about the aggregate reward from time m the m plus n minus 1, it's the same as the aggregate reward from time 0 up to time n minus 1. The expected values are the same. The actual sample functions are different.

OK, so if I try to calculate this aggregate reward conditional on xm equals i, mainly conditional on starting in state i, then this expected aggregate reward, I use that as a symbol for it, is the expected value of r of xm, given xm equals i. What is that? Well, that's ri. I mean, given that xm is equal to i, this isn't random anymore. It's just the source sub i.

Plus the expected value of r of xm plus 1, which is the sum over j, of pij times r sub j. That's the time m plus 1 given that you're in state i at time m, and so forth, up until time n minus 1, where the expected reward, then, is a piece of ij.

Probability of being in state j at time n minus 1 given that you started off in state i at time 0 times r sub j. And since expectations add, we have this nice, convenient formula here. We're doing something I normally hate doing, which is building up a lot of notation, and then using that notation to write extremely complicated formulas in a way that looks very simple.

And therefore you will get some sense of what we're doing is very simple. These quantities in here, again, are not all that simple. But at least we can write it in a simple way. And since we can write it in a simple way, it turns out we can do some nice things with it. OK. So where do we go from all of this?

We have just said that the expected reward we get, expected aggregate reward over n steps, namely from m up to m plus n minus 1. We're assuming that if we start at time m, we pick up a reward at time n. I mean, that's just an arbitrary decision. We might as well do that, because otherwise we just have one more transition matrix sitting here. OK, so we start at time m.

We pick up a reward, which is conditional on the state we start in. And then we look at the expected reward for time m and time m plus 1, m plus 2, up to m plus n minus 1. Since we started at m, we're picking up n different rewards. We have to stop at time m plus n minus 1. OK, so that's this expected aggregate reward. Why do I care about expected aggregate reward?

Because the rewards at any time n are sort of trivial. What we're are interested in is how does this build up over time? You start to invest in a stock. You don't much care what it's worth at time 10. You care how it grows. You care about its value when you want to sell it, and you don't know when you're going to sell it, most of the time.

So you're really interested in these aggregate rewards that you. You'll see when we get to dynamic programming what you're interested in that, also. OK. If the Markov chain is an ergotic unit chain, then successive terms of this expression tend to a steady state gain per step.

In other words, these terms here , when n gets very large, if I run this process for very long time, what happens to p sub ij to n minus 1? This tends towards the steady state vector pi sub j. And it doesn't matter where we started. The only thing of importance is where we end up. It doesn't matter how high this is. So we have a sum over j, of pi sub j times r sub j.

After a very long time, the expected gain per step is just a sum of pi j times our r sub j. That's what's important after a long time. And that's independent of the starting state. So what we have here is a big, messy transient, which is a sum of a whole bunch of things. And then eventually it just settles down, and every extra step you do, you just pick up an extra factor of g as an extra reward.

The reward might, of course, be negative, like in the stock market over the last 10 years, or up until the last year or so, who was negative for a long time. But that doesn't make any difference. This is just a number, and this is independent of starting state. And p sub in can be viewed a transient ni, which is all this stuff at the beginning.

The sum of all these terms at the beginning plus something that settles down over a long period of time. How to calculate that transient, how to combine it with the steady state gain. Then those talk a great deal about that. What we're trying to do today is to talk about dynamic programming without going into all of this terrible mess about dealing rewards words in a very systematic and simple way.

You can read about that later. What we're aiming at is to talk about dynamic programming a little bit, and then get off to other things. OK. So anyway, we have a transient, plus we have a steady state gain. The transient is important. And it's particularly important if g equals zero.

Namely if your average gain per step is nothing, then what you're primarily interested in is how valuable is it to start in a particular state? If you start in one state versus another state, you might get a great deal of reward in this one state, whereas you make a loss in some other state. So it's important to know which state is worth being in. So that's the next thing we try to look at.

How does the state affect things? This brings us to one example which is particularly useful. And along with being a useful example, well, it's a nice illustration of Markov rewards. It's also something which you often want to find. And when we start talking about renewal processes, you will find that this idea here is a nice connection between Markov chains and renewal series.

So it's important for a whole bunch of different reasons. OK. Suppose for some arbitrary unit chain, namely we're saying one set of recurring states. We want to find the expected number of steps, starting from a given state i, until some particular state 1 is first entered. So you start at one state. There's this other state way over here.

This state is recurrent, so presumably, eventually you're going to enter it. And you want to find out, what's the expected time that it takes to get to that particular state? OK? If you're a Ph.D. student, you have this Markov chain of doing your research. And at some point, you're going to get a Ph.D. So we can think of this as the first pass each time to your first Ph.D. I mean, if you want to get more Ph.D.'s, fine, but that's probably a different Markov chain. OK. So anyway, that's the problem we're trying to solve here.

We can view this problem as a reward problem. We have to go through a number of steps if we want to view it as a reward problem. The first one, first step is to assign one unit of reward to each successive state until you enter state 1. So you're bombing through this Markov chain, a frog jumping from lily pad to lily pad. And finally, the frog gets to the lily pad with the food on it.

And the frog wants to know, is it going to start before he gets to this lily pad with the food on it? So, if we're trying to find the expected time to get there, here what we're really interested in is a cost, because the frog is in danger of starving. Or on the other hand, there might be a snake lying under this one lily pad. And then he's getting a reward for staying alive.

You can look at these things whichever way you want to. OK. We're going to assign one unit of reward to successive state until state 1 is entered. 1 is just an arbitrary state that we've selected. That's where the snake is underneath a lily pad, or that's where the food is, or what have you. Now, there's something else we have to do.

Because if we're starting out at some arbitrary state i, and we're trying to look for the first time that we enter state 1, what do you do after you enter state 1? Well eventually, normally you're going to go away from state 1, and you're going to start picking up rewards again. You don't want that to happen.

So you do something we do all the time when we're dealing with Markov chains, which is we start with one Markov chain, and we say, to solve this problem I'm interested in, I've got to change the Markov chain. So how are we going to change it? We're going to change it to say, once we get in state 1, we're going to stay there forever.

Or in other words, the frog gets eaten by the snake, and therefore its remains always stay at that one lily pad. So we change the Markov chain again. The frog can't jump anymore. And the way we change it is to change the transition probabilities out of state 1 to p sub 1, 1, namely the probability given you're in state 1, of going back to state 1 in the next transition is equal to 1.

So whenever you get to state 1, you just stay there forever. We're going to say r1 equal to zero, namely the reward you get in state 1 will be zero. So you keep getting rewards until you go to state 1. And then when you go to state 1, you don't get any reward. You don't get any reward from any time after that. So in fact, we've converted the problem.

We've converted the Markov chain to be able to solve the problem that we want to solve. Now, how do we know that we haven't changed the problem in some awful way? I mean, any time you start out with a Markov chain and you modify it, and you solve a problem for the modified chain, you have to really think through whether you changed the problem that you started to solve.

Well, think of any sample path which starts in some state i, which is not equal to 1. Think of the sample path as going forever. In the original Markov chain, that sample path at some point, presumably, is going to get to state 1. After it gets to state 1, we don't care what happens, because we then know how long it's taken to get to state 1.

And after it gets to state 1, the transition probabilities change. We don't care about that. So for every sample path, the time that it takes the first pass each time to state 1 is the same in the modify chain as it is in the actual chain. The transition probabilities are the same up until the time when you first get to state 1.

So for first pass each time problems, it doesn't make any difference what you do after you get to state 1. So to make the problem easy, we're going to set these transition probabilities in state 1 to 1, and we're going to set the reward equal to zero. What do you call a state which has p sub i, i equal to 1?

You call it a trapping state. It's a trapping state because once you get there, you can't get out. And since we started out with a unit chain, and since presumably state 1 is a recurrent state in that unit chain, eventually you're going to get to state 1. But once you get there, you can't get out.

So what you've done is you've turned the unit chain into another unit chain where the recurrent set of states has only this one state, state 1 in it. So it's a trapping state. Everything eventually leads to state 1. All roads lead to Rome, but it's not obvious that they're leading to Rome. And all of these states eventually lead to state 1, but not for quite a while sometimes. OK.

So the probability of an initial segment until 1 is entered is unchanged, and expected first pass each time is unchanged. OK. A modified Markov chain is now an ergotic unit chain. It has a single recurrent state. State 1 is a trapping state, we call it. ri is equal to 1 for i unequal to 1, and r1 is equal to zero.

This says that a state 1 is first entered at time l, and the aggregate reward from 0 to n is l for all m greater than or equal to l. In other words, after you get to the trapping state, you stay there, and you don't pick up any more reward from then on.

One of the things that's maddening about problems like this, at least that's maddening for me, because I can't keep those things straight, is the difference between n and n plus 1, or n and n minus 1. There's always that strange thing, we've started at time m, we get reward at time m. So if we're looking at m transitions, as we go from m the m plus n minus 1. And that's just life.

If you try to do it in a different way, you wind up with a similar problem. You can't avoid it. OK, so what we're trying to find is the expected value of v sub i of n, and the limit as n goes to infinity, we'll just call that v sub i without the n on it. And what we want to do is to calculate this expected time until we first enter state one. We want to calculate that for all of the other states i.

Well fortunately, there's a sneaky way to calculate this. For most of these problems, there's a sneaky way to calculate these limits. And you don't have to worry about the limit. So the next thing I'm going to do is to explain what this sneaky way is. You will see the same sneaky method done about 100 times from now on until the end of course. We use it all the time.

And each time we do it, we'll get a better sense of what it really amounts to. So for each state unequal to the trapping state, let's start out by assuming that we start at time zero, and state i. In other words, what this means is first we're going to assume that x sub 0 equals i for some given i. We're going to go through whatever we're going to go through, then we'll go back and assume that x sub 0 is some other i.

And we don't have to worry about that, because i is just a generic state. So we'll do it for everything at once. There's a unit reward at time 0. r sub i is equal to 1. So we start out at time zero and state i. We pick up our reward of 1, and then we go on from there to see how much longer it takes to get to state 1.

In addition to this unit reward at time zero, which means it's already taken us one unit of time to get the state 1, given that x sub 1 equals j, namely, given that we go from state i to state j, the remaining expected reward is v sub j. In other words, if it's times 0, I'm in some state i.

Given that I go to some stage j, the next unit of time, what's the remaining accepted expected time they get to state 1? The remaining expected time is just v sub j, because that's the expected time. I mean, if v sub j is something where it's very hard to get to state 1, then we really lost out. If it's something which is closer to state 1 in some sense, then we've gained.

But what we wind up with is the expected time to get to state 1 from state i is one. That's the instant reward that we get, or the instant cost that we pay, plus each of the possible states we might get to. There's a cost to go, or reward to go from that particular j. So this is the formula we have to solve. What's this mean? It means we have to solve this formula for all i.

If I solve it for all i, and I've solved this for all i, then that's the linear equation in the variables v sub 1 up to v linear equations in i equals 2, up to m. We also have decided that v sub 1 is equal to 0. In other words, if we start out in state 1, you expect the time to get to state 1 is 0. We're already there. OK. So we have to solve these linear equations.

And if your philosophy on solving linear equations is that of, I shouldn't say a computer scientist because I don't want to indicate that they are any different from any of the rest of us, but for many people, your philosophy of solving linear equations is to try to solve it. If you can't solve it, it doesn't have any solution. And if you're happy with doing that, fine.

Some people would rather spend 10 hours asking whether in general it has any solution, rather than spending five minutes solving it. So either way, this expected first passage time, we've just stated what it is. Starting in state i, it's 1 plus the time to go for any other state you happen to go to.

If we put this in vector form, you put things in vector form because you want to spend two hours finding the general solution, rather than five minutes solving the problem. If you have 1,000 states, then it works the other way. It takes you multiple hours to work it out by hand, and it takes you five minutes by looking at the equation.

So sometimes you win, and sometimes you lose by looking at the general solution. If you look at this as a vector solution, the vector v where v1 is equal to zero, and the other v's are unknowns, is the vector r, the vector r is 0. 0 reward in state 1. Unit reward in all other states, because we're trying to get to this end. And then we have the matrix here, t times v.

So we want to solve this set of linear equations, and what do we know about this set of linear equations? We have an ergotic unit chain. We know that p has an eigenvalue, which is equal to 1. We know that's a simple eigenvalue. So that in fact, when we write v equals r plus pv as zero equals r plus p minus i times v.

And we try to ask whether v has any solution, what's the answer? Well, this matrix here has an eigenvalue of 1. Since it has an eigenvalue of one, and since it's a simple eigenvalue, there's a space of solutions to this equation. The space of solutions is the vector of all ones and the vector of all anything else. In other words, it's a vector of v times any constant alpha.

Now we've stuck this in here, so now we want to find out what's the set of solutions now. We observe v plus alpha e also satisfies this equation if we found another solution. So if we found a solution, we have a one dimensional family of solutions.

Well, since this eigenvalue is a simple eigenvalue, the space of vectors for which r is equal to p minus i times v as a one dimensional space, and therefore there has to be a unique solution to this question. OK. So in fact, in only 15 minutes, we've solved the problem in general, so that you can deal with matrices of 1,000 states, as opposed to two states.

And you still have the same answer. OK. So this equation has a simple solution, which says that you can program your computer to solve this set of linear equations, and you're bound to get an answer. And the answer will tell you how long it takes to get to this particular state. OK. Let's go one to aggregate rewards with a final reward. Starting to sound like-- yes?

STUDENT: I'm sorry, for the last example, how are we guaranteed that it's ergotic? Like, I possible you enter a loop somewhere that can never go to your trapping state, right?

PROFESSOR: But I can't do that because there always has to be a way of getting to the trapping state, because there's only one recurrent state. All these other states are transient now.

STUDENT: No, but I mean-- OK, like, let's say you start off with a general Markov chain.

PROFESSOR: Oh, I start off with a general Markov chain? You're absolutely right. Then there might be no way of getting from some starting state to state 1, and therefore, the amount of time that it takes you to get from that state to the starting state is going to be infinite. You can't get there.

So in fact, what you have to do with a problem like this is to look at it first, and say, are you in fact dealing with a unit chain? Or do you have multiple recurrent sets? If you have multiple recurrent sets, then the expected time to get into one of the recurrent states, starting from either a transient state, or from some other recurrent set is infinite.

I mean, just like this business we were going through at the beginning. What you would like to do is not have to go through a lot of calculation when you have, or a lot of thinking when you have multiple recurrent sets of states. You just know what happens there. There's no way to get from this recurrent set to this recurrent set. So that's the end of it.

STUDENT: OK. So like it works when you have the unit chain, and then you choose your trapping state to be one instance [INAUDIBLE].

PROFESSOR: Yes. OK. Good. Now, yes?

STUDENT: The previous equation is true for any reward. But it's not necessary--

PROFESSOR: Yeah, it is true for any set of rewards, yes. Although what the interpretation would be of any set of rewards is if you have to sort that out. But yes. For any r that you choose, there's going to be one unique solution, so long as one is actually a trapping state, and everything else leads to one.

OK, so why do I want to put a-- ah, good.

STUDENT: I feel like there's a lot of the rewards that are designed for it, designed with respect to being in a particular state.

PROFESSOR: Yes.

STUDENT: But if the rewards are actually in transition, so for example, if you go from i to j, there are going to be a different number from j to j. How do you deal with that?

PROFESSOR: How do I deal with that? Well, then let's talk about that. And in fact, it's fairly simple so long as you're only talking about expected rewards. Because if I have a reward associated with-- if I have a reward rij, which is the reward for transition i to j, then if I take the sum of rij times p summed over j, what this gives me is the expected reward associated with state j, with state i.

Now, you have to be a little bit careful with this because before we've been picking up this reward as soon as we get to state i, and here suddenly we have a slightly different situation where you have a reward associated with state i but you don't pick it up until the next set. So this is where this problem of i or i plus 1 comes in.

And you guys can do that much better than I can, because at my age I start out with an age of 60 and an age of 61 is the same thing. I mean, these are-- OK. So anyway, the point of it is, if you have rewards associated with transitions you can always convert that to rewards associated with states. Oh, I didn't really get to this.

What I've been trying to say now for a while is that sometimes, for some reason or other, after you go through and end steps of this Markov chain, when you get to the end, you want to consider some particularly large reward for having gotten to the end, or some particularly large cost of getting to the end, or something which depends on the state that you happen to be in.

So we will assign some final reward which in general can be different from the reward that we're picking up at each of the other states. We're going to do this in a particular way. You would think that what we would want to do is, if we went through in steps, we would associate this final reward with the n-th step. We're going to do it a different way.

We're going to go through n steps, and then the final reward is what happens on the state after that. So we're really turning the problem of looking at n steps into a problem of looking at n plus 1 steps. Why do we do that? Completely arbitrary. It turns out to be convenient when we talk about dynamic programming, and you'll see why in just a minute.

So this extra final state is just an arbitrary thing that you add, and we'll see the main purpose for it in just a minute. OK. So we're going to now look at what in principle is a much more complicated situation than what we were looking at before, but you still have this basic mark off condition which is making things simple for you.

So the idea is, you're looking at a discrete time situation. Things happen in steps. There's a finite set of states which don't change over time. At each unit of time, you're going to be in one of the set of m states, and at each time l, there's some decision maker sitting around who looks at the state that you're in at time l.

And the decision maker says I have a choice between what reward I'm going to pick up at this time and what the transition probabilities are for going to the next state. OK, so it's kind of a complicated thing. It's the same thing that you face all the time. I mean, in the stock market for example, you see that one stock is doing poorly, so you have a choice.

Should I sell it, eat my losses, or should I keep on going and hope it'll turnaround? If you're doing a thesis, you have the even worse problem. You go for three months without getting the result that you need, and you say, well, I don't have a thesis. I can't say something about this. Should I go on for one more month, or should I can it and go on to another topic?

OK, it's exactly the same situation. So this is really a very broad set of situations. The only thing that makes it really different from real life is this Markov property sitting there and the fact that you actually understand what the rewards are and you can predict them in advance.

You can't predict what state you're going to be in, but you know that if you're in a particular state, you know what your choices are in the future as well as now, and all you have to do at each unit of time is to make this choice between various different things. You see an interesting example of that here.

If you look at this Markov chain here, it's a two state Markov chain. And what's the steady state probability of being in state one? Anybody? It's a half, yes. Why is it a half, and why don't you have to solve for this? Why can you look at it and say it's a half? Because it's completely symmetric. 0.99 here, 0.99 here, 0.01 here, 0.01 here.

These rewards here had nothing to do with the Markov chain itself. The Markov chain is symmetric between states one and two, and therefore, the steady state probabilities have to be one half each. So here's something where, if you happen to be in state two, you're going to stay there typically for a very long time.

And while you're studying there for a very long time, you're going to be picking up rewards one unit of reward every unit of time. You work for some very stable employer who pays you very little, and that's a situation you have. You're sitting here, you have a job but you're not making much, but still you're making something, and you have a lot of job security.

Now, we have a different choice when we're sitting here with a job in state two, we can, for example, you can go to the cash register and take all the money out of it and disappear from the company. I don't advocate doing that, except, it's one of your choices. So you pick up a big reward of 50, and then for a long period of time you go back to this state over here and you make nothing in reward for a long period of time while you're in jail.

And then eventually you pop back here, and if we assume the judicial system is such that it has no memory, [INAUDIBLE] you can cut into the cash register, and, well, OK. So anyway, this decision two, you're looking for instant gratification here. You're getting a big reward all at once, but by getting a big reward with probability one, you're going back to state zero.

From state zero, it takes a long time to get back to the point where you can get a big reward again, so you wonder, is it better to use this policy or is it better to use this policy? Now, there are two basic ways to look at this problem. I think it's important to understand what they are before we go further.

One of the ways is to say, OK, let's suppose that I work out which is the best policy and I use it forever. Namely, I use this policy forever or I use this policy forever. And if I use this policy forever, I can pretty easily work out what the steady state probabilities of these two states are. I can then work out what my expected gain is per unit time and I can compare this with that.

And who thinks that this is going to be better than that and who thinks that this is going to be better than that? Well, you can work it out easily. It's kind of interesting because the steady state gain here and here are very close to the same. It turns out that this is just a smidgen better than this, only by a very small amount. OK.

See, what happens here is that here, you tend to go for about 100 steps here. So you pick up every reward of about 100 if you use this very simple minded analysis. Then for 100 steps, you're sitting here, you're getting no reward, so you think we ought to get every reward of one half on the average, and that's exactly what you do get here.

And here, you get this big reward of 50, but then you go over here and you spend 100 units of time in purgatory and then you get back again, you get another reward of 50 and then spend hundreds units of time in purgatory. So again, you're getting pretty close to a half of a unit of reward, but it turns out, when you work it out, that here is just a smidgen. It's 1% less than a half, so this is not as good as that.

But suppose that you have a shorter time horizon. Suppose you don't want to wait for 1,000 steps to see what's going on, so you don't want to look at the average. Suppose this was a gambling game. You have your choice of these two gambling options, and suppose you're only going to be playing for a short time. Suppose you're going to be only playing for one unit of time.

You can only play for one unit of time and then you have to stop, you have to go home, you have to go back to work, or something else. And you happen to be sitting in state two. What do you want to do if you only have one unit of time to play. Well, obviously, you want to get the reward of 50, because delayed gratification doesn't work here, because you don't get any opportunity for that gratification later.

So you pick up the big reward at first. So when you have this problem of playing for a finite amount of time, whatever kind of situation you're in, what you would like to do is say, for this finite amount of time that I'm going to play, what's my best strategy then?

Dynamic programming is the problem, which is the algorithm which finds out what the best thing to do is dynamically. Namely, if you're going to stop in 10 steps, stop in 100 steps, stop in one step, it tells you what to do under all of those circumstances. And the stationary policy tells you what to do if you're going to play forever.

But in a situation like this where things happen rather slowly, it might not be the relevant thing to deal with. A lot of the notes deal with comparing the stationary policy with this dynamic policy. And I'm not going to do that here because, well, we have too many other interesting things that we want to deal with. So we're just going to skip all of that stuff about stationary policies.

You don't have to bother to read it unless you're interested in it. I mean, if you're interested in it, by all means, read it. It's a very interesting topic. It's not all that interesting to find out what the best stationary policy is. That's kind of simple. What's the interesting topic is what's the comparison between the dynamic policy and the stationary policy.

But all we're going to do is worry about what the dynamic policy is. That seems like a hard problem, and someone by the name of Bellman figured out what the optimal solution to that dynamic policy was. And it turned out to be a trivially simple algorithm, and Bellman became famous forever.

One of the things I want to point out to you, again, I keep coming back to this because you people are just starting a research career. Everyone in this class, given the formulation of this dynamic programming problem, could develop and would develop, I'm pretty sure, the dynamic programming algorithm.

Developing the algorithm, understanding what the problem is is a trivial matter. Why is Bellman famous? Because he formulated the problem. He said, aha, this dynamic problem is interesting. I don't have to go through the stationary problem. And in fact, my sense from reading his book and from reading things he's written is that he couldn't have solved the stationary problem because he didn't understand probability that well.

But he did understand how to formulate what this really important problem was and he solved it. So, all the more credit to him, but when you're doing research, the time you spend on formulating the right problem is far more important than the time you spend solving it. If you start out with the right problem, the solution is trivial and you're all done.

It's hard to formulate the right problem, and you learn to formulate the problem not by playing all of this calculating things, but by setting back and thinking about the problem and trying to look at things in a more general way. So just another plug. I've been saying this, I will probably say it every three or four lectures throughout the term. OK.

So let's go back and look at what the problem is. We haven't quite formulated it yet. We're going to assume this process of random transitions combined with decisions based on the current state. In other words, in this decision maker, the decision maker at each unit of time sees what state you're in at this unit of time.

And seeing what state you're in at this given unit of time, the decision maker has a choice between how much reward is to be taken and along with how much reward is to be taken, what the transition probabilities are for the next state. If you rob the cash register, your transition probabilities are going to be very different than if you don't rob the cash register.

By robbing the cash register, your transition probabilities go into a rather high transition probability that you're going to be caught. OK, so you don't want that. So you can't avoid the problem of having the rewards at a given time locked into what the transition probabilities are for going to the next state, and that's the essence of this problem. OK.

So, the decision maker observers the state and chooses one of a finite set of alternatives. Each alternative consists of recurrent reward which we'll call r sub j of k, the alternative is k, and a set of transition probabilities. pjl of k, one less than or equal to a l less than or equal to m for going to the next state.

OK, the notation here is horrifying, but the idea is very simple. I mean, once you get used to the notation, there's nothing complicated here at all. OK, so in this example here, well, we already talked about that. We're going to start out at time m.

We're going to make a decision at time m, pick up the associated reward for that decision, and pick the transition probabilities that we're going to use at that time m, and then go on to the next state. We're going to continue doing this until time m plus n minus 1. Mainly, we're going to do this for n steps of time.

After the n-th decision-- you make the n-th decision at m plus n minus t-- there's a final transition based on that decision. The final transition is based on that decision, but the final reward is fixed ahead of time. You know what the final reward is going to be, which happens at time m plus n.

So the things which are variable is how much reward do you get at each of these first n time units, and what probabilities you choose for going through the next state. Is this still a Markov chain? Is this still Markov? You can talk about this for a long time. You can think about it for a long time because this decision maker might or might not be Markov.

What is Markov is the transition probabilities that are taking place in each unit of time. After I make a decision, the transition probabilities are fixed for that decision and that initial state and had nothing to do with the decisions that had been made before that or the states you've been in before that.

The Markov condition says that what happens in the next unit of time is a function simply of those transition probabilities that had been chosen. We will see that when we look at the algorithm, and then you can sort out for yourselves whether there's something dishonest here or not. Turns out there isn't, but to Bellman's credit he did sort out correctly that this worked, and many people for a long time did not think it worked.

So the objective of dynamic programming is both to determine the optimal decision at each time and to determine the expected reward for each starting state and for each number and steps. As one might suspect, now here's the first thing that Bellman did. He said, here, I have this problem. I want to find out what happens after 1,000 steps. How do I solve the problem?

Well, anybody with any sense will tell you don't solve the problem with 1,000 steps first. Solve the problem with one step first, and then see if you find out anything from it and then maybe you can solve the problem with two steps and then maybe something nice will happen, or maybe it won't.

When we do this, it'll turn out that what we're really doing is we're starting at the end and working our way back, and this algorithm is due to Richard Bellman, as I said. And he was the one who sorted out how it worked. So what is the algorithm? We're going to start out making a decision at time 1. So we're going to start at time n.

We're going to start in a given state i. You make a decision, decision k at time m. This provides a reward at time m, and the selected transition probabilities lead to a final expected reward. These are these final rewards which occur at time n plus 1. It's nice to have that n because it's what let's us generalize the problem. So this was another clever thing that went on here.

So the expected optimal aggregate reward for a one step problem is the sum of the reward that you get at time m plus this final reward you get at time n plus 1, and you're maximizing over the different policies you have available to you. So it looks like a trivial problem, but the optimal reward with a one step problem is just this.

OK, next you want to consider the two step problem. What's the maximum expected reward starting at xm equals i with decisions at times m and n plus 1. You make two decisions. Now, before, we just made one decision at time m. Now we make a decision at time m and at time n plus 1, and finally we pick up a final reward at time n plus 2.

Knowing what that final reward is going to be is going to affect the decision you make at time n plus 1, but it's a fixed reward which is a function of the state. You can adjust the transition probabilities of getting to those different rewards.

The key to dynamic programming is an optimal decision at time n plus 1 can be selected based only on the state j at time n plus 1. This decision, given that you're in state j at time n plus 1, is optimal independent of what you did before that, which is why we're starting out looking at what we're going to do with time n plus 1 before we even worry about what we're going to do with time n.

So, whatever decision you made at time n, you observe what state you're at time n plus 1 and the maximal expected reward over times n plus 1 and n plus 2, given that you happen to be in state j is just maximal over k as the reward you're going to get by choosing policy k and the expected value of the final reward you get if you're using this policy k.

This is just dj* of 1 and u as you just found. In other words, you have the same situation at time n plus 1 as you have at time n. Well, surprisingly, you've just solved the whole problem. So we've seen that what we should do at time n plus 1 is do this maximization.

So the optimal reward, aggregate reward over times m, n plus 1, and n plus 2 is what we get maximizing over our choice at time m of the reward we get at time m plus the decision plus the transition probabilities which we've decided on which get us to this reward at time n plus 1 and n plus 2. We found out what the reward is for times n plus 1 and n plus 2 together.

That's the reward to go, And we know what that is, so we have this same formula we used before. Why do we want to look at these final rewards now? Well, you can view this as a final reward in state m. It's the final reward which tells you what you get both from state n plus 1 and n plus 2.

And, going quickly, if we look at playing this game for three steps, the optimal reward for the three step game is the immediate reward optimized over k plus the rewards at n plus 1, n plus 2, and n plus 3, which we've already found. And in general, the optimal reward at time n-- when you play the game for n steps, the optimal reward is maximum here.

So, all you do in the algorithm is, for each value of n when you start with n equal to 1, you solve the problem for all states and you maximize over all policies you have a choice over, and then you go on to the next larger value of n, you solve the problem for all states and you keep on going. If you don't have many states, it's easy. If you have 100,000 states, it's kind of tedious to run the algorithm.

Today it's not bad, but today we look at problems with millions and millions of states or billions of states, and no matter how fast computation gets, the ingenuity people to invent harder problems always makes it hard to solve these problems. So anyway, that's the dynamic programming algorithm. And next time, we're going to start on renewal processes.