PROF.

PATRICK WINSTON: It's too bad, in a way, that we can't paint everything black, because this map coloring problem sure would be a lot easier.

So I don't know what we're going to do about that.

How long is this going to take?

Here's what we're going to do.

We're going to wait till either all the laptops are closed, or this program terminates, whichever comes first.

So how long is this going to take?

I actually don't know.

I think it'll take more than the life time of the universe, but I'm not sure.

So let's take a look at a slightly easier map coloring problem.

We'll stop this one and change the map to something I call [? simplicia. ?] There's 26 states, one for each letter in the alphabet.

And what we're going to do is we're going to do a depth first search for a suitable coloring of this map.

We're going to go in order, A, B, C, D, E. And as I've suggested here, at each level we're going to rotate the color choices so we don't over use any one color.

So if we launch this particular search, this depth first attempt to color this map.

There it goes.

And maybe we should wait until it terminates.

Or maybe we should just let it run and it'll terminate, perhaps, sometime within the lecture.

Or maybe we should just let it run over the weekend.

Or how long do you think we would have to wait, if we want to come back and watch it terminate?

At roughly 30 frames a second-- I'm calculating it all in my head, I don't want to bet my life on it-- but I think about

5,000 years.

And if we want to use as many states as there are in the United States, in this demonstration, you get up to numbers like 10 to the 17th years-- 17th, 18th, 16th, I'm not exactly sure.

I did a rough calculation.

And, of course, you could do some parallels into that, and it's not as bad as chess, where you need all the atoms in the universe and you still can't do it, and things like that.

Acting as computers.

But still, it's pretty horrendous.

The problem is, well the problem is illustrated by this diagram I put in back of me.

If you do a depth first search and you have a problem like Texas.

Texas has always, always been a problem, ever since they joined the Union.

I could pick on other states, too.

Kentucky, Tennessee, if you're from those states you represent something even worse.

But Texas is down there.

And the trouble is, if you assign a color to Arizona, Oklahoma, Arkansas, and Louisiana first, and then wait around to Texas last, then you get yourself into a bind by your fourth choice, that you don't discover until you're 48th choice.

So what happens then, is that you start developing this tree like this, you get to those states surrounding Texas.

Texas is last state assigned a color and there's nothing left for it.

And that problem was there on the fourth choice, and you don't discover it until your 50th choice.

So you develop a horrendous, impossible, impossible search.

So you simply can't do it that way.

But now, not to worry.

I've come equipped with the idea of constraint propagation.

So we could just take a country with four states, like this.

Each can be labeled red, green, blue and yellow.

So just like in a case of line drawings, we'll pile up all the possible things that the value can be-- red, green, blue and yellow.

Red, green, blue and yellow.

Red, green, blue and yellow.

And we start up constraint propagation.

So we say for the upper left hand corner state, is there any reason to believe that R is impossible?

Well we look at our neighbors and see what kind of constraint flows in from them.

And sure, this guy could be green, and this guy could be blue.

They don't have to be red, so there's nothing that rules out red.

And there's nothing the rules out blue.

And there's nothing that rules out yellow.

And there's nothing that rules out green.

So constraint propagation just sits there with its finger up its nose, doing nothing.

So it doesn't look like we can use either depth first search, or constraint propagation.

So we could just give up and cry.

But maybe there's some other approach that will help.

So let me actually work the Texas problem.

So with apologies to Houston and Tyler, here's a map of Texas.

There we are.

And here's, roughly speaking, Arizona is over here somewhere.

And we've got Oklahoma in there.

And old Bill Clinton's state, Arkansas.

And then Louisiana sticks out there a little bit.

So there's our map of that particular part of the country.

So we've got Arizona here, Oklahoma here, Arkansas here, and Louisiana here, and Texas here.

And we have elected to assign colors to these states, in that order.

So this is one, two, three, four.

And we're going to do that.

We're going to rotate our color choices, just so we don't over use any one color.

But we're going to also have a look at Texas as we go around.

Because Texas is a state that borders on the States they were choosing colors for.

So the only possible colors that Texas could be are red, green, blue, and yellow.

So as we make our choices around here, we'll say that-- we don't have to adhere to any particular style-- we can say that Arizona is going to get the colored red, R. That's going to rule out R over here for Texas, because no adjacent states can have the same color.

Then we go over to Oklahoma, and we're rotating our color choices, so we'll say that can be green.

And that's fine, because it's consistent with the red here, but it rules out the possibility that Texas could be green.

And then we go over here to Arkansas, red, green, blue.

That's fine, that's consistent with the green on Oklahoma, but if we look at its neighbors we know that Texas is forever forbidden to be blue, now.

So now we go over to Louisiana, and remember, we're rotating our color choices because we don't want to overuse them.

So this means that the first choice we're going to make here, for Louisiana, is yellow.

And that's fine because it's consistent with Arkansas, but it's not so fine because it's now ruled out the last possibility for Texas.

So even though Texas is going to be the 48th state that we color, we're going to say, at this point, there's no need in going on.

We'd better back up.

Because there's nothing left for Texas when we get around to coloring it.

So that means that this yellow is ruled out here.

This yellow reappears.

We select the next color in my line for Louisiana, which happens to be red.

And now that's consistent with this yellow that's still left for Texas.

And it's also consistent with the blue that's up here for Arkansas.

So that's cool.

I wonder if maybe we could make an algorithm out of that, and solve problems like this.

Do you see, sort of, the intuition of what we're doing?

We're actually using the martial arts principle, again.

Because the whole problem is that local constraints, undiscovered local constraints, are causing downstream problems.

So we're going to use the enemy's powers against him, and we're going to look at those local constraints as we go and make sure they're not going downstream, not going to get us later on.

So now I'm going to look like I'm getting a little formal, but I'm just getting a little bit more formal.

Because I want to have some language that I can use to describe what's going on, so that it's clear what the choices are.

So to start off with, we're going to have to have some vocabulary.

So let's start up our vocabulary here.

We're going to have a notion of a variable v. And that's something that can have an assignment.

There's nothing complicated about that.

And a value.

A value x is something that can be in assignment.

It's a little bit circular, but we're all in computer science so you know what I mean.

So the next thing is a little slightly less obvious, and that's the notion of a domain, d.

And that's going to be a bag of values.

OK.

So one more thing.

A constraint.

That's a constraint c, is a limit on-- in our examples it's mostly going to be pairs of variables, pairs of variable values.

But in general, it could be variable values.

So if we go back here to Texas we could say, OK, how does our vocabulary drape itself over that configuration?

And the answer is, the states have the role of variables, the colors have the role of values.

And the domains are the remaining color possibilities that we can still use on a particular state.

And the constraint, in this case, is the simple map coloring constraint that no states that share a boundary can have the same color.

So states are variables, colors are values, domains are bags of colors, and constraints-- there's only one-- adjacent states can't have the same color.

So that's how it fits with this vocabulary.

So now, what did I actually do here?

Well what I actually did here, I'm going to now formalize a little by writing it down in pseudo code.

So here we are, we're going to have a look at what we did here with our intuition, and we're going to reduce it to a procedure.

And here's the procedure.

Remember, we're doing depth first search on this stuff.

I did a depth first search.

We're going to do a depth first search, and for each depth first search assignment-- OK, so here I am, I'm labeling Arizona, and then Oklahoma, and then Arkansas, and then Louisiana.

When I give each one of those a label, a color, I'm going to do this procedure.

Every time I make one of those assignments.

The last one that caused trouble was coloring Louisiana yellow.

Each time I put one of those colors down, each time I make an assignment, I'm going to do this procedure.

So for each depth first search assignment, for each variable v, considered.

Now you don't know what I mean by considered.

But when I put a label, when I put up a value, show something as a color for Louisiana, I thought about Texas.

So I was considering the variable, Texas, when I made the assignment for Louisiana.

Now I'm going to be a little bit vague about what I mean by considered, right now.

Because there are lots of choices about how much stuff you actually consider.

So let me just say consider, and then we'll open that up and talk about the options in a moment, so for each variable v considered for-- let's call that variable v sub i-- for each x sub i, for each value in the domain of that variable, consider each of the things that still surviving.

For each of those, for each constraint c, that's between x sub i, and some x sub j, where x sub j is an element of the domain of j.

Now that sounds awfully fancy, but this just says, in the case of Texas up there, whenever I consider one of the values that's still remaining as a choice for Texas, I want to consider all of the constraints between that variable and the adjacent states.

And I want to be sure that anything I leave in the domain is OK for some selection in the other states, some remaining choices in the other states.

So that's why we're getting pretty nested here.

But we're doing depth first search.

We are considering the variables in a certain collection of variables.

For each one of those where considering all the values that still remain in the domains of those variables.

And then for each of those values, we're checking to see if it satisfies this some constraint, satisfies the constraint that are placed upon it.

So for each of these constraints if there does not exist an x sub j, such that, the constraint between x sub i and x sub j is satisfied, well if in that adjacent place there's nothing that is consistent with a value, then we've got to get rid of it.

If that's true.

If there does not exist some value in an adjacent variable such that that constraint is satisfied, we're hosed.

We've got to get rid of that value.

So we're going to remove x sub i from d sub i.

OK.

Now, that's fine.

That's sort of what I did with Texas.

As soon as I plopped down a value for Louisiana I said, well what are the possible values in Texas?

Red, green, blue and yellow.

Let's consider red.

Let's consider the constraints between Texas and all adjacent states.

One of those constraints says it can't be the same color as Arizona.

The only color I've got available for Arizona, since I've already made the assignment is red.

Red is not consistent with red, so I've got to get rid of it.

So it looks complicated, but it's just intuition.

So what do we do if we get to a situation where the domain is empty?

That means whenever we get around to making assignment to it, there won't be anything left.

So if that ever happens, if the domain ever becomes empty, then what do we do?

We've got to back up.

So the intuition is clear.

This is the algorithm.

The algorithm when you work through it, think about whether it makes sense and what not.

How it fits with Texas.

Yeah, it sure does.

All we're doing is we're making these death first assignments.

And in the neighborhood of those depth first assignments we're looking around to see if the values that are possible include something.

And if they don't include anything, we know we made and irrevocable blunder, and we have to back up.

So that's the essence of the idea.

Now, how well does it work?

Well a little bit depends on what we choose for considered.

There are lots of choices for what we consider.

So let me enumerate some of those choices and then we'll have a look and see what they do.

Oh I guess one possibility is to consider nothing.

Let's try it out.

So our type of search is going to be no checks.

What do you think is going to happen?

We're not even checking the assignment.

That's pretty fast.

Unfortunately, since we haven't even check the most recent assignment, we get lots of places where there are states that are adjacent to each other that have the same color.

That's no good.

So another thing we can do is consider everything everything.

That's no good, because that would say, as soon as we color our first state, we check to make sure that all 47 other states can be colored.

That seems like it over doing it a little bit.

But in any case, at least we want to check the assignment.

So if we go back here and check the assignment, let's see what happens.

Type assignments, assignments only.

Boom.

Aw, gees, that's where I got in trouble before.

This is the thing is going to run for 17 billion years at nanosecond or something like that.

It's only a billion years if you run it at nanosecond speed, so I guess maybe you could do that.

Have a fast computer.

But this isn't going to work because of our unfortunate choice of Texas as the last state to be considered, and the

unfortunate coloring of the four surrounding states right up front.

And our unfortunate decision to rotate the color so as to avoid overdoing any one color.

So this doesn't work.

We know we went to the trouble of working out the business with Texas by hand, using the domain reduction algorithm.

Better make a note that this is the domain reduction algorithm.

And what we're going to do is we're going to check the neighbors of the assignments.

Just like we did here.

We checked Texas each time we made one of those four choices, because it's a neighbor of all of the choices of the states that we made.

So one thing to do is to check neighbors.

This is one, two, three, now let's see what happens.

Check neighbors only, go.

Shoot, I don't know.

It's OK with Texas, right?

Because it didn't color the states around Texas with all of the four color choices.

But it's still getting into trouble in other places.

Like the states like Missouri, Kentucky, Virginia, Tennessee, states with lots of boundaries.

So I don't know whether this is going to-- oh, there it finally worked.

It went through a lot of effort, though.

For the sake of comparison, we might make a note that it ran into 9,139 dead ends.

But it did do some good.

It didn't take a length of time longer than the remaining part of the universe.

But if it's a good idea to check the neighbors, if we make a change to the neighbors, what might that suggests that we do in addition?

Well it might suggest that if we make a change to a neighbor, that we check its neighbors, too, make sure they're all right.

So another choice is to propagate.

So propagate through variables with reduced domains.

Let's see how that works.

Wait a minute.

I must have made a mistake.

Let's try that again.

Oh, maybe we better slow it down.

All that grey stuff is showing the limit of the propagation.

Man it's, let's see at four second of 40, that's about 10 seconds.

Boom.

Not bad.

Zero dead ends.

And it was a lot faster.

I didn't happen to notice how many constraints were checked on that other thing, I think it was around 20,000 or so.

This is a lot less.

So this looks like a good idea.

But why did I label it number five?

Well because there's something between this and number three.

So number four is, through v with d reduced to one value.

So we're not going to propagate through all of the variables which have their domains shrunk a little bit.

We're only going to propagate through those that have the greater shrinkage, all the way down to a single value.

So let's see how that might work.

Anybody want to place any bets on this one?

Let's see.

We checked 2,623 constraints last time.

Let's see what happens this time.

You can see that the extent of the gray is less, because it's not propagating so far.

And as we breathily await the answer, I'd say we've found our winner.

As this does a couple of dead ends, but the number of constraint checked is less than 1,000.

So in general, with problems this, you have all of these choices for what you consider.

You don't want to consider nothing, because then you're not honoring your constraints.

You'll certainly want to consider the things you just made assignments for, because otherwise you'll construct a solution that violates a constraint.

You don't want to do everything, because that's excessive work.

And so checking the neighbors is a good idea, but it's always better in practice.

In practice, inevitably it's the case that it's better to do some propagation through the things that you've changed.

How much propagation?

It doesn't seem to do much good to propagate through things are just changed.

But it does seem to do some good to propagate through the things that have changed and been reduced to a single value.

So as soon as you get a neighbor of some assignment you just made that has its domain reduced to a single value, then you check its neighbors, too.

So you check the neighbors, of the neighbors, of the neighbors, of the neighbors, on and on and on, as long as you've found a domain being reduced.

And not only being reduced, but reduced to a signal value.

All right?

So that's the demand reduction algorithm.

And I guarantee you a problem like that.

And I know you don't know how to work those problems yet, because this is a little bit abstract.

And to work these problems in the exam setting you need to know a little bit about how to keep track of the variable values that remain in the domain, and that sort of thing.

And you'll learn more about that in your recitations, and in this mega recitation, and in the tutorials.

So we could go home except that there are few little flourishes to deal with here.

And those flourishes, include some dirty, filthy little secrets.

For example, I've chosen, as my classroom example, to pick on Texas.

And arranged for this situation to be especially ugly.

So I could arrange the states in a different way.

We have highly constrained states, that have a lot of bordering states around them.

And we have other states, like Maine, up there, that only borders on one other state.

So I don't know.

Will, what do you think?

Should we arrange the states for our death first search in the order of least constrained to most constrained, or most constrained to least constrained?

In other words, should we start with Missouri, or Tennessee, or Kentucky, or something like that?

Or should we start with Maine?

What do you think?

You have a 50% chance of getting it right, just by [? looking ?] at points.

WILL: Start with the most first.

PROF.

PATRICK WINSTON: He thinks we should start with the most constraint first.

Do we have a volunteer who wants to suggest that we start with the least constraint first?

That's the way I always work on stuff.

I'm working on a book or something, I have 500 things to fix, I'll always choose to work on the easiest stuff first, so that I feel like I'm making the list a lot smaller.

Leave the hardest things to last.

But we don't have any volunteers who want to bet on that idea of least constraint first?

OK.

Jason wants to suggest that we should work on least constraint first.

Well we have ground truth, because we can just try it out.

I guess we'll stick with our shrinking to one value thing, here.

But we will reorder things so that we have the least constrained first.

So right away, we got a color for Maine.

Maybe we ought to speed this up a little bit.

Well, that's a good idea.

Jason suggested this and we only 1,732 constraints and we had 59 dead ends.

So let's try the other way around, and we'll go back to four frames a second.

So we're working, kind of from the middle of the country out, with this one.

We're going to deal with Maine, I guess, last.

Which is better.

Too bad, I think it looks like this is better.

In fact, let's not be so aggressive with the use of constraint propagation.

Let's just check the assignments only.

If we go back to an arrangement where we have least constrained first, and we'll crank up the speed.

Well actually, we would have to crank it up pretty big, because the states like Missouri, Tennessee, Kentucky, they're going to be like Texas.

And so were kind of back to the length of the universe type problem, here.

With the least constraint first, and no use of constraints, other than to check the current assignment.

So let's stop that, though, and check the most constrained first, assignments only.

I don't know how long's this going to take.

That's the dirty little secret.

If we had arranged our states from most constrained to least constrained, ordinary depth first search with none of this stuff we talked about today would work just fine.

All right.

So it's a little bit like games.

Do you use progressive deepening, or do you use alpha beta?

And the answer is both.

You use everything you've got to deal with the problems.

And depending on the problem, one or another of the things you incorporate into your approach will work just

great, if you're lucky.

So now, I promise that this is useful not only for people who want to color maps.

God, who wants to do that?

We know it can be done with four colors.

But it's also useful for doing all kinds of resource planning problems.

So I want to show you a resource planning problem, and I want you think about-- while I'm doing it-- think about whether it's actually analogous to the map coloring problem.

All right?

So here's the deal.

You have just landed a summer job with the Jet Green, a new airline.

And Jet Green is a low cost, no frills, hardly any maintenance type of airline.

And they want to fly mostly between Boston and New York.

Occasionally they want to fly to Los Angeles.

And they're trying to get by with the smallest number of airplanes.

So that's why we have a kind of resource allocation problem with Jet Green.

So I'm going to write down what their schedule looks like.

They have one flight, F1, that goes from Boston to JFK, like so.

It's an early in the day flight.

Then they want to have another one, F2, that flies from JFK to Boston.

And then they want to have another flight a little later in the day that flies from Boston to JFK.

And a little later than that, they want to have another flight that goes from JFK to Boston.

They're going to start off mostly as a shuttle airline in the beginning.

So that's F1, F2, F3, and F4.

And they have a fifth flight, F5, that goes from Boston to Los Angeles, that takes a long time.

So it looks like this on the schedule.

Of course we have time going that way.

So that's Boston to LAX.

Now your job is to determine if they can fly this schedule with four aircraft?

And naturally you don't want to over use any one aircraft, because you would like to have even wear on them.

Right?

So as you make your choices, you'll rotate the aircraft.

So you'll assign to this one to A1, aircraft number one.

This one will be A2.

This one will be A3.

And this one will be A4.

And, oops, there's no aircraft left for the flight to Los Angeles, because you only have four.

So, it's obvious, right?

This is 100%, exactly, the map coloring problem, even down to the four choices.

Because, you have the constraint, the no single physical aircraft can fly two flights at the same time.

Just like no two adjacent states can be colored the same.

So there's a no same time constraint, like so.

So if you were assigning aircraft to these flights, you would get down to F4, the fourth flight, and you would say, well, let's see, this guy down here can be A1, A2, A3, or A4.

But if I choose A4 for that fourth flight, then there would be nothing left in its domain.

So you've thus set the problem up to be identical to the map coloring problem.

And, of course, you can enrich it with other kinds of constraints.

So, for example, you might have-- this is a not same time constraint-- and these, I mean, this is at JFK.

And it flies out of JFK, so maybe you can use the same aircraft for those.

But not if they're right up against each other, because you have a minimum ground time rule.

So there's a minimum ground time constraint here.

And there's a minimum ground time constraint here.

There's a minimum ground time constraint here.

And if these are at the same city, then you've got to allow enough time for them to fly between the two cities that are involved.

So the constraints can get a little bit more complicated, but the idea is the same.

So you say to me, I don't trust you, show me.

OK.

So let me show you.

Oh, by the way, there's one more way to make this map coloring problem easier, right?

Let's see.

The arrangement is going to be alphabetical, the type is going to be assignments only, and we know that's a loser.

But not if we use a whole lot of colors.

So it's the use of four colors that got us into trouble.

Now that's an aside, but it'll be coming back in a moment or two.

Scheduling, here's that problem.

Boom.

You can almost see it working just like the map coloring thing.

But that's maybe too easy.

Let's do this one.

See this is just like [? simplicia. ?] We've kind of got a goofy arrangement here that's guaranteed to lose at the bottom because of choices made at the top.

But that's OK.

We can that we can stop this, and we can change to check neighbors only.

And boom, there it goes.

Or alternatively, let me see if I can do it this way.

Most constrained first, type will be assignments only.

Boom.

That worked fine, too.

So you might choose to have a slightly harder problem, like this.

And we can search away.

Actually I don't know if this will complete or not.

This is a randomly generated example.

But you're going to lose your summer job if you can't figure out whether you can do this, or not.

So what are you going to do?

[? Elliott, ?] you got any thoughts about how you're going to save your job?

So here's the question you've been asked.

How many airplanes does Jet Green need?

And you decide, well, four seemed to work before, so we'll try four here.

You're not sure if it's going to terminate or not, I mean, in your lifetime, let alone in your summer job.

[? Elliott, ?] let me give you a hint.

Look at the outline.

The outline up here, on the board, the last item.

[? ELLIOTT: [INAUDIBLE] ?] PROF.

PATRICK WINSTON: Yeah, what's that mean?

What's the maximum number of airplanes we're going to need?

Suppose we've got five flights, what's the maximum number of airplanes we would ever need?

Five.

What's the minimum number of airplanes we'll need?

One.

So let's try it with a small number of airplanes and a large number of airplanes.

So that showed us very fast that we can't do it with one airplane.

That showed us very fast we can do it with ten airplanes.

SPEAKER 1: [INAUDIBLE] amount of overlappage from up [INAUDIBLE].

PROF.

PATRICK WINSTON: Volunteer?

What are we going to do to find the actual number, as fast as possible.

And at least give our boss a reasonable answer, even if not necessarily the exact number very fast?

It's easy, right?

We're going to start up here with one computer, and we're going to start down here with another computer, and see what happens.

So let's see if we can do it with nine.

Let's see if we can do it with eight.

Let's see if we can do it with seven.

These take almost zero time, right?

Because they're under constraint.

Wow, that's good, seven.

Let's try six.

Actually let's try two.

It loses fast.

Let's try three.

I don't know.

Maybe if you let it run one long enough three will work.

I doubt it.

While we're at it though, we might as well go back here and try it with six.

Remember seven worked real fast.

Gees, six, that was six, right?

Yeah.

So let's try it with five.

OK.

So it runs real fast with five.

It terminates real quick with two, so we got three and four left.

So we could tell our boss, a la any time algorithm, that you're not real sure, but you know it's going to be either

three or four.

And then, you got two computers.

You can let both run and see if either one terminate.

So you have three and four.

My guess is that three will eventually give up.

But of course, there's another little problem here.

We haven't used the most constraint first.

If we did that, we might be able to do it even faster.

Actually, I don't think I can make that switch without getting another random assignment, but let's see what happens.

Maybe so.

SPEAKER 2: [INAUDIBLE] PROF.

PATRICK WINSTON: Oh, I already had most constraint first?

OK.

So it didn't help to actually switch.

And I think I've got a new schedule to work here.

So that's the end of the story.

You can do good resource allocation if you do several things are once.

Number one, you always want to use most constraint first.

Number two you want to propagate through domains produced to a single algorithm.

And number three, if you really try to figure out what the minimum number of resources needed is, you do this under over business and you can quickly converge on a narrow range where the search is taking a long time, and be sure that it lies within that narrow range.

Because when you over resource, it's fast to complete, and when you under resource, it's fast to terminate.

So you can just squeeze it right down into a very small range.

And that is the end of the story.

Enjoy your holiday on Monday.

We'll have two classes next week on Wednesday and Friday, as advertised.