ConvexOptimizationII-Lecture18

Instructor (Stephen Boyd):Well, let me – you can turn off all amplification in here. So yeah, so it's still – you still have amplification on in here so you can – oh, well, we'll let them figure that out. Let's see, couple of announcements. It's actually kind of irritating, frankly. I wonder if we can just cut that off? Oh, the advertisement went away, that's good.

All right, I'll let them figure out how to turn off the amplification in a room that's got, like, a depth of 15 feet. So yes! No? All right. Anyway, I'll let them worry about it. Okay, first just a couple of announcements to – I keep getting hopefully. What do you think? Are we there? Getting better. It's funny. You'd think you could just turn it off.

But anyway, all right, okay. So a couple of announcements is, as you know, the final project reports are due tomorrow. So that's one deadline. The other is on Monday; I think it's been agreed. We're gonna have our poster session.

We haven't fixed the time exactly yet but we're thinking of something like 5:00-7:00 and we're gonna figure out about getting some food. So we'll – but we haven't announced that. That'll be on the website and all that kind of stuff when we figure everything out.

And there's already been some questions about what's the format of the posters and we don't know. So some people are gonna go ahead and do the fancy thing and make big posters which you're more than welcome to do but you don't have to. You can just print out 12 black and white slides, or whatever, and, like, you know, tack them to the poster board. And I don't remember what the size of the poster board is but maybe that'll also be forthcoming on the website or maybe be an announcement or something like that.

So the format, which is 12 slides, is – that's fixed and I've said it many times but, you know, please use our template. You're welcome not to, you don't have to, but then you better have better taste than we do. So, I mean, even – and there's certain things you just can't do because it's not acceptable and it's amateurish and stuff like that. And you all know what I'm talking about.

So, okay, so we're gonna finish up our absolutely last topic which is branch and bound methods. These are methods for global optimization. So in global optimization you have a non-convex problem but you're going to give up – you're not gonna give up a globalness. That's not a word, I just made it up.

But you're not gonna give up globalness – globality? You're gonna give up neither globalness nor globality but you are gonna give up speed. So what's gonna happen is these are gonna be methods that are slow but they don't lie. They will produce a – at any point you can stop them and exactly like a convex optimization method they will have a lower bound and they will have an upper bound.

And they're not approximate, they are correct and they terminate when the lower – when the upper bound minus the lower bound gets less than some tolerance and that's global. So this is a huge field. You can take entire classes on it. So this is just your taste of global optimization and it's basically just to give you the rough idea of how these things work.

There are other methods but they're all kind of related to this and then bits and pieces get very sophisticated. And even here I'll show you where there's lots of room for personal expression in these methods.

So, okay, so last time we started talking about this and the basic idea is this. It's gonna rely on two methods in your problem. You're gonna somehow – you're gonna parse up the region, you know what, there's actually amplification still on in here but it's cool. Actually it's not, but I mean, it can't be that hard to turn off the amplification.

Okay, so you're gonna have two methods which over a region of the feasible set given some description of a reason of a subset of the feasible set will compute a lower bound and an upper bound.

The lower bound, of course, is gonna be – that's gonna be the sophisticated one. Well, it can be sophisticated. That'll often be computed a Lagrange relaxation, by a duality, something like that. Oh, dear. Ah, the pleasures of – wow, now it's – I used to – well, a long time ago I did rock and roll sound so I know about this. These are bad monitors anyway.

So, okay, so the upper bound can also – can be done by a variety of methods ranging from extremely stupid and simple, for example, here's an upper bound. If your region is a rectangle you simply go to the middle of the center of the rectangle and you evaluate the objective there and the constraint functions. If that point is infeasible, you return the upper bound plus infinity, which is valid but not that useful. If it is feasible you turn the objective value at the center of the interval as the upper bound.

So that's, you know, they can't – it can't get any simpler than that. It's just evaluating the function at the center. You can do other things, too. Like, you can run a local optimization method starting from there you can do whatever you like to get a better point. Anything will work.

Matter of fact, these can have a huge effect on a real method and would be if you actually implemented some of this you would do this. We're not even asking you to do this on your last homework problem.

Okay, so this is the basic idea is you're gonna partition the feasible set into convex sets and then find lower and upper bounds for each. So that's the idea – and we'll quit. And we'll actually look at two specific examples of the general idea and once you've seen these two, which are a bit different from each other, I mean, actually they're kind of the same, you'll figure out how to do this in much more general – in the general case because it's really the ideas and not the details that matter here.

So here's the way it's gonna work. First we'll do – we'll just do simply unconstrained, non-convex minimization. Actually it's silly; it could be constrained because you could build right into F the constraints by assigning F the value plus infinity outside the feasible set.

Okay, so we're gonna optimize over this m-dimensional rectangle to some prescribed accuracy epsilon. It's gonna work like this, we're gonna have Q methods a – well, sorry, I'll get to the Q methods in a minute but we're gonna define phi min of Q as the global optimum of F, the global minimum of F over the rectangle.

So we're just looking to compute F*, that's what we're actually gonna compute here. Okay, so we're gonna have two upper and lower bound functions. So it's gonna be a lower bound function and upper bound function. They must be above and below this minimum value and they have to be – one attribute they have to have is they have to be tight as the size of the rectangles shrink.

That basically even the stupidest method would satisfy this, so, the simplest lower bound, I mean, extremely bad would be the following. The simplest upper bound is the value of F at the middle of the rectangle and the simplest lower bound by far is to subtract from that number a Lipschitz constant, the diameter of the rectangle or the radius of the – diameter of the rectangle divided by 2 multiplied by a known Lipschitz constant on F. That does the trick.

Now that stupid one has the property that this – has this property that if a rectangle gets small enough then the gap between these two is less than your epsilon. By the way, you could use that as your safety backup lower bound, meaning that you can run a sophisticated lower bound based on convex optimization or something like that and then use this one if it's better. And that means that it inherits the proof in that case. Such a method would inherit – such an algorithm would inherit the proof and so you would be safe from the proof police in that case.

So if there was a raid and someone said, "Can you absolutely prove your method works?" You'd point to a line of code and say, "Yes, it does." So that's really the only reason you would do that, by the way, would safety from them.

Okay, all right, so the idea is that these should be easy to – they should be cheap to compute because, of course, they could just be equal to this and that's – but then they're impossible to compute. So, in fact, the tradeoff here is gonna be something like this. You want cheaply computable – you want bounds that are cheap to compute but good.

Good means merely that they are close enough to each other in the instances you'll look at that branch and bound will run quickly. So that's sort of the key to everything here is getting cheaply computable bounds.

Okay, so let's look at the branch and bound algorithm, it works like this, basic one in this case goes like this. You – I call my lower bound method on the initial rectangle and I call my upper bound method and I call these U1 and L1.

Obviously I can terminate if U1 minus L1 is less than epsilon because then I'm done and I'm absolutely done. By the way, this method can also return a point that achieves this upper bound. So – if you wanted to say, "Okay, I'm terminating and I want to return an X then it will be the job of this thing to return the bound."

By the way, it's the job of the lower bound to return a certificate proving it. So if you want to make the algorithm sort of really formal and correct and all that when it quits it should quit returning two things, a certificate that the upper bound – that this value – that this upper bound is actually truly and upper bound.

Best certificate being to demonstrate a point which has that objective value and which is feasible and a lower bound. We're gonna find out what the lower bound looks like in a minute. It's much more sophisticated than merely a dual feasible point which is what you have in convex optimization.

Okay, so that's – we'll see what this is in a minute. Okay, now if this does not hold – so in other words, your upper and lower bounds are not less than your tolerance what you're gonna do is you're gonna split the rectangle into two subrectangles.

Okay? So – and you're gonna do that by choosing a variable to split on. So a rectangle, of course, is something that looks between – it's the rectangle is described by two vectors, an L and a U, a lower and an upper bound. You choose a component to split on and then in fact in more fancy versions you could decide how you want to split it.

Do you want to split it evenly or not evenly and this is where all the personal expression goes into these methods. So if you had some clever method where there was any reason you didn't want to split it evenly that would be fine.

But you'll partition it like this and then you'll call the lower bound and upper bound methods on these children. And, in fact, this is – let me just draw a picture here to show how this works in R2. I mean, it's silly in R2 but here's the point.

So here's your initial rectangle like this and you evaluate the upper and lower bound and if they're not within tolerance you split it. And you could either split it along these variable and you're gonna partition two rectangles, you could split it along X1 or along X2 like this. And let's say you decide to split it along X1 you can further decide how you want to split it. But here it doesn't say, it doesn't specify so there's a splitting like that.

Now what you do is you calculate the lower bound here and the upper bound here and let me just do a – I'm gonna fill in some numbers just so we can see how this looks. And, in fact, if I do this example you'll kind of understand absolutely everything about this. So

let's start with this one. I call the upper bound and I get ten, I call the lower bound and I get six.

So that's the – so we know that the optimal value of the function in here is somewhere between six and ten. That's all we know, you don't know anything else. What can you say about the function value like at this point here? What is it?

Student:[Inaudible].

Instructor (Stephen Boyd):It's bigger than – well, what do you mean it's bigger than six and ten?

Student: It's [inaudible].

Instructor (Stephen Boyd):Ah-ha, okay. I thought you were gonna say that. It was a trap. I wanted you to say that. It's false. Half of it is true. It's bigger than six because the minimum I can provide that this six means I can prove that on this rectangle the function never gets smaller than six. Why? Because I have a lower bound on the function of six. Okay?

What's the – so I can tell you if I pick a point here the function value here is bigger than size. I cannot say it's less than ten. What can I say though? I can say this, there is a point in here, maybe not that one, but there is a point in here which has a function value of less than or equal to ten, that I can say.

By the way, I'm skipping, you know, I'm assuming the [inaudible] and the [inaudible] are achieved in all that. You know, I mean, it's true anyway, the F is – let's make F continuous or whatever. It doesn't matter.

So there is a point and, in fact, it would be the job of the upper bound routine to return that point in case it were challenged. Actually, in case our – suppose our epsilon were five in which case we're done. We've calculated the global optimum within five it would be the job of – so there is a point, let's call it there.

But this doesn't really matter. Okay? So, all right. So now we're gonna partition – we're gonna go like this and I'm gonna call my lower bound function here and here and my upper bound function. And let's actually – if I go through a couple of – I'll just ask you some questions and I think then you'll know everything there is.

And let me switch these around so the lower bound comes first. Okay. So here let's see what would happen – well, let me ask you a couple questions. Let's call the lower bound function and let's suppose it were five here. Any comments? That's the lower bound. Is it a valid lower bound?

Student: It's a valid lower bound.

Instructor (Stephen Boyd):Yes, it's a valid lower bound but –

Student: It's of no use.

Instructor (Stephen Boyd):- it's of no use. And, in fact, if I got five there what could I do with the five? Immediately replace it with –

Student: Six.

Instructor (Stephen Boyd):- six. Okay. So the point is we don't require – we do not require that the lower bound and upper bound function should return better answers than their parents. But, in fact, we can just quietly copy – in other words, calculating this lower bound was a complete waste of time if we get five. Everybody got that?

So if you get this – I could just replace it with six because basically – and if someone challenged you and says, "How do you know the function is bigger than six in this whole region?" You say, "Look, I know it's bigger than six in a bigger region. Therefore it's bigger than 6 in here." Okay, good. So that's one discussion.

Okay, so let's suppose it's seven here and eight is the upper bound. Okay? So any comments? By the way, what if this were 11? Be very careful. So what's that?

Student:[Inaudible].

Instructor (Stephen Boyd):It could happen. You can't say anything stupid has happened yet but I can now. Right? Let's look at that. What can you say now? Does it bother you? How much does it bother you? Is it impossible or merely stupid? But which one is it? Are these valid? Could these be valid bounds? Yeah, sure. Okay.

So you can insist on the following, when you call the upper bound method on the children at least one of them is as good as the upper bound in the parent. Everybody got that? The reason is this, the – in fact, when you call the upper bound on the parent you could ask for which – for the location of the point that achieved that number then that child – whichever child contains that point can't have an upper bound that's worse.

So this can't be. One of these two can be modified to be 10. Everybody follow? Okay, by the way, it's just – once you understand all this you understand everything. I mean, but you actually really have to think about these things because they're not obvious.

So, I mean, well, they are obvious. Sorry. It's just they're confusing but you have to draw this picture and then think about it very carefully.

All right, let's do one more. How's that? It's fine. Okay? And now let me ask you, suppose this is what the two children – so you've taken the two children, you've called the lower and upper bound method on both children and they've returned these bounds.

Now I want to know what's the global lower – what's the new lower bound? What can you say if someone tells you – asks you, you just – let's see, in the first iteration you called both of these methods once each, right? Now you've done it three times. So you've got three times the work invested now and the question is, "What is the new lower bound – what do you know how that you didn't know before?"

Student: It's between seven and nine.

Instructor (Stephen Boyd):It's between seven and nine.

Student:[Inaudible].

Instructor (Stephen Boyd):Okay. Hang on, let's just go very slow. What's the new lower bound?

Student: Seven.

Instructor (Stephen Boyd):Seven. Okay. And the argument – and that's better than it used to be which was six. So originally it was six, meaning that whatever else happened we knew that the minimum had to be bigger than or equal to six. Now we know it's bigger than seven which is the minimum of these two numbers. Right?

Because in this region we trust these methods. It says that the function value is never smaller than seven here, it's never smaller than eight here. By the way, do you know which side the minimum is gonna be in?

No, not yet. We're getting there, okay? No, you don't know. You still don't know. But your new global lower bound is seven. What's your new global upper bound? Well, it can't be – well, it could be 12, it's valid. I started by saying, look, the optimum value is absolutely less than or equal to ten. Now what can I say?

Student:[Inaudible].

Instructor (Stephen Boyd):Ten?

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah, no, we've agreed on it could be either side. You know there's still amplification on? Well, it's a good way to end a year of –

Student:[Inaudible].

Instructor (Stephen Boyd):What? What? What? What? It's good now. Okay, great, great. Let's just – okay. Yeah, so what's the new upper bound?

Student: Nine.

Instructor (Stephen Boyd):Nine. Explain it to me.

Student: There's a point [inaudible].

Instructor (Stephen Boyd):Good. Okay. So if I told you, "No, no, no, no, no, the global low – you know, no, no, no, no, F* the minimum on the whole thing is 9.5." Then this thing's lying. So there's a point on the left whose value is nine and therefore the global – so we went from six nine after this iteration to seven nine. So that was our – that's literally the – so the interval of ignorance was this and it went down to this.

Everybody got this? Okay. So, I mean, these are not complicated you just have to go to a quiet place figure out the mins and the maxes and all the logic. It's elementary but as you can see you actually have to think about it for a minute to figure out what it all means.

And let me just do one more thing just for fun. Let's do this. That's 9.5, okay? Now what can you say now?

Student:[Inaudible].

Instructor (Stephen Boyd):What can you say?

Student:[Inaudible].

Instructor (Stephen Boyd):The optimal point is on the left side, okay. And what are the new bounds?

Student:[Inaudible].

Instructor (Stephen Boyd):Seven and nine. So which is the same we had a moment ago except we've actually now made an additional statement which is that the optimal solution must or can lie in the left-hand side. Must, right? Okay, and the – what's the argument? What's your argument? How do you argue it?

Student: That's the point of the left side that has a value lower than the lower bound on the right.

Instructor (Stephen Boyd):Yeah. So the point is that there is a point in here which has a value of nine. Every point over here has a value that exceeds 9.5 and therefore this whole thing goes away and we're back down to – and now we've actually learned something at least in terms of the location of it.

And so this is called pruning. We're gonna get to all of this but if you understand all these ideas of which is about five or six, they're all quite elementary and all that, you understand branch and bound. That's – basically that picture, that's all of it.

So but let's go over it now. Okay, so here's what happens. Okay, so you're gonna split it, you're gonna update the lower bound so the lower bound is gonna be the following. It's going to be the minimum of the lower bound; the new global lower bound is the minimum of the lower bound's returned by the children.

And the upper bound is also the minimum of the upper bound returned by the children. And by the way if you wanted to also put in the U1 and L1 you could do that as well here. The new – because you can't – you could query the children, get two answers back and effect you could have no progress whatsoever towards improving things. Right? None.

It's nothing in the semantics of the lower and upper bound methods that requires you to make progress. And so I won't go back to that example and show how that works.

Okay, you find the partition and repeat steps three and four and let me just ask a couple of – well, we'll get to that. Okay, so that's how this works. Okay, now as I said you can assume without loss of generality that the upper bound is not increasing an LI is non – is also non-increasing. Okay? So that's right.

So – have I got that right? So as you go down the lower – the upper bound is gonna go down. By the way it has the option of staying level if you don't get anything better. And LI is – I want to say that LI is non-decreasing, don't I? I do. That's a typo. LI is non-decreasing. Okay.

So what happens is when you repeat this several times, what you do is you have a binary tree and so let me just show you over here what that's gonna look like in this case. So let's go back to this and let's suppose now I choose to split this guy and I split it like so and then I split this one and I get that – something like that. Then what I'm really developing in this case is a tree, a binary tree. And so it would look something like that.

Let me draw this, I'll have to make it a very fat tree. So you get something that looks like that and this node corresponds to this original rectangle. This node corresponds to this left rectangle, right? And I could even label this cut as L and R for left and right, okay?

Then this node corresponds to this rectangle which I then split and that happens to be top and bottom. So I can call this Top and Bottom, for example. And then these nodes – this node corresponds do this rectangle and the bottom corresponds to this rectangle. This one was further split like that and then this would be L and R, okay.

So this partially developed binary tree corresponds to this partition here and let me just make a couple of comments about it. So the leaves of this tree correspond exactly to this set of rectangles, okay. So that's what the leaves are. By the way, the non-leaf nodes correspond to rectangles also in here but ones that are further split, right? So that's what this thing looks like.

Okay, on every node here I've calculated a lower – every node here I've called the lower and the upper bound method. So I have a lower bound in all of these and if you have a lower bound – if you look at the leaves, right? The leaves constitute a rectangle partition of the original set.

In other words, it's a set of rectangles whose union gives you the original rectangle and whose intersection has – intersection of any pair has non-empty interior. That's a rectangular partition. Did I say non-empty? I meant to say empty interior, okay. So that's a rectangular partition here. So you have this thing.

Okay, now if the lower bounds on these things were, for example, you know, 5 – 5.5, you know, 4 and 8 what can you give? What is the lower bound for the entire thing? What's the lower bound on the whole function? I just drew the lower bounds on those nodes.

Student:[Inaudible].

Instructor (Stephen Boyd):What is it? What?

Student: Eight.

Instructor (Stephen Boyd):That's the lower – I want the lower bound on the optimal value of the parent – the top rectangle.

Student:[Inaudible].

Instructor (Stephen Boyd):What's that?

Student: Four.

Instructor (Stephen Boyd):It's four. Do you know where it is? Well, not until I draw some upper bound there. If I draw upper bound you'll know where it is. You might, you might not. Every upper bound there could be 20 and that just means it's weird but, I mean, it's perfectly valid. Your lower bound is four, okay. And now we can actually describe – I can tell you exactly what a certificate of the lower bound looks like in a non-convex problem.

If I send a halt message to this process and say, "Stop, I'm done." Let's see, you just did one, two, three, four, five, six, seven, you just did – did an effort of solving seven. Let's say each lower bound is a convex problem you've solved. So you just solved seven convex problems, say, "That's it. I'm out of time. I quit."

It returns a lower bound of four here. And then you say – then you go back to the method and you say, "Prove it." Right? So how do you prove it? So in a convex problem you prove it by producing a dual certificate. How do you prove the lower – what data – what object do you hand back that proves it?

Student:[Inaudible].

Instructor (Stephen Boyd):No, there's no – this is abstract, there's no Lipschitz constant, you don't know anything. I mean, it's an abstract lower bound method.

Student:[Inaudible].

Instructor (Stephen Boyd):There's no point – if there a point that achieves it you're done. If you have a lower bound – if you know the global optimum is bigger than or equal to four and you have a point whose value is four you're done.

You don't have a point. The – you might not even have a feasible point yet. All the upper bounds everywhere could be labeled plus infinity. So how would you prove it? If someone said – if you said, "I happen to know that the function value in this big rectangle is bigger than four." Someone would say, "Prove it." How would you do it?

Student:[Inaudible].

Instructor (Stephen Boyd):Okay, but so? You have to return a – you can say, "Here my proof. My proof is this that in this region every point in that region has a function value bigger than five. Every single one in this one has a function value bigger than 5.5. Everything here has a function value bigger than four and everything here has a function value bigger than eight."

So what you return is you actually return the partition with a lower bound on each one. Everybody – so that's your certificate of proof. So, in fact, when you finish a branch and bound method what's – instead of giving the actual certificate proving your lower bound is actually gonna be a partition of the original space with each element in the partition will have its own lower bound.

And if you want to certify that lower bound you have to go to whoever provided that lower bound and it could be Lipschitz, could be duality, Lagrange. I mean, who knows what it is, right? It could be anything you like, any lower bound method you know.

But that would be the matter of the lower bound method. That particular lower bound method; it could be different lower bounds by the way for each one, right? Yeah?

Student: Based on the lower bound with the non-leaf nodes?

Instructor (Stephen Boyd):Oh, no, I have a lower bound here, too. Yeah, definitely have a lower bound there. Here, I'll draw it. It's 4.5 here, something like that. There. Yeah, let's put some lower – and when I did the whole thing I started with three and then this could be also 4.5.

Student: You have the lower bound to be 4.5.

Instructor (Stephen Boyd):In this case? Uh-oh, you're right, sorry. I'm doing it the wrong way. Thank you. Sorry. Yeah, I shouldn't make it four. I meant to the other way around. I was working on it. How about 3.8 and you're gonna have to help me on this – three – how about that? That look reasonable?

Yeah, but the way I drew it, it would have been 4.5 and that's because we're – despite whatever this slide says in the top line we're actually assuming that the LI's are increasing because that's always a possibility, right? So if you call a lower bound method on a subrectangle and you get a worse lower bound than the parent you just take the parents lower bound. So, okay? Everybody got this? I mean, these are – it's very, you know, it's simple and all that but you actually – it can easily get confused. Make sense?

Student:[Inaudible].

Instructor (Stephen Boyd):Where? Over here? It's quite likely. Well, what is it? You don't like it? Oh, you mean this?

Student: Okay, about [inaudible].

Instructor (Stephen Boyd):No, no, that's okay. That's cool.

Student:[Inaudible].

Instructor (Stephen Boyd):Yeah. That's fine. It means basically that the function value – so what's the problem with it?

Student:[Inaudible].

Instructor (Stephen Boyd):There probably is one but – tell me what you don't like about it. You should all be checking this, by the way, this reflects on you. In fact, it reflects on you more than me because I don't really care.

But imagine what other people will think if they watch this video and they think, "You gotta watch this." It's like, "There's this whole class from Stanford they – this guy said something so stupid it's amazing and they just sat there happily." So it'll reflect on you more than me. I think it's cool. What do you think? Everybody cool with it? All right. Yeah?

Student: What is the lower bound [inaudible] is, like, like solving the same kind of problem but there's no [inaudible]?

Instructor (Stephen Boyd):No, no. It's totally abstract. This actually – heterogeneous methods for lower bounds could be being run for each node and totally different. One could be based on a Lipschitz constant; one could be based on Lagrange duality. You could fire up four lower bounding methods on one, you know, for one region and return

to the caller the best of them. One could be like an SDP bound; one could be a SDP relaxation, one a Lagrange dual. I mean, it makes no difference.

So, no. And then you have the option if you get a bunch of lower bounds and you'd pick the best one. You can also pick the best one of the calling parent because that's also a stupid lower bound, right? Because if you look at a subregion it can't have a lower bound – well, it could be worse, but it's just silly than the [inaudible]. So, okay?

So did we reach stationarity? I think we're cool. Okay, so – okay. All right. So let's see – so this is the basic idea. In fact, all we did was we walked through – whoops. This, this business here. And then this would go – this actually now would go recursively through the tree. And so in fact it's the same as saying it's the minimum of all of the leaf nodes. Okay?

So to – by the way to make this algorithm – to full specify it you need a couple of rules. You need to know things like this, which rectangle do you split? So you have this binary tree – well, you have a partition and you can split a rectangle so that's – you have to choose which one do you – so for example, on the next step someone says, "Okay, I give you one more step. You have to split like this guy, you know, one of the leaves here you're gonna split, so one of the four.

Then when you decide to split a rectangle you'd have to decide to split it like, you know, along which variable? In the 2D case you want to split it either vertically or horizontally but in the M dimensional case which I guess is this, you choose which of the M variables you're gonna split on. Okay?

And, you know, it's immediately obvious you can do things like if you had eight processors split could mean you could not divide it in half but divide it into eight sections and send – and then send the 8 subrectangles or whatever to your eight processors or whatever. So, you know, these are all kind of obvious things.

Okay, so here are some rules – this is where the personal expression – this is one of the areas where personal expression comes in is in choosing these things. One is you split rectangle with the smallest lower bound along the longest edge and in half. I mean, that's kind of just a default method.

But in any particular application maybe you can think of a better way and you can experiment with some and find that one will work really, really well and stuff like that.

And you can mumble all sorts of idea about justifying this but they are absolutely nothing, you're just mumbling. I mean, there's no reason to prefer one of these – I mean, you can talk about that there's probably stupid methods but this – there's lots of other methods, other splitting rules and things like that that can be justified just as well as this one.

Student: So in every example we split one of the leaves?

Instructor (Stephen Boyd):Yeah.

Student: And we'd get new upper and lower bounds on that leaf and probably get them all the way up to the root or no?

Instructor (Stephen Boyd):Yeah. So let's see what happens when you split. Let's choose one and split it. I mean, assuming, you know, that we're not in – I'm already in some big trouble here or whatever.

So in this case if we follow this rule we would take this four here and I would split this four. And so that's gonna be this guy, I think, right? And so I split it like that and then it would – I would develop my tree this way and now my two diagrams have hit each other. So I would get this – this was 4 and now you're gonna have to help me make some consistent things here.

So let's see what would happen. Let's make it interesting and let's imagine progress is made in lower bound and I would do that by saying this is 4.5 and I have to be very careful, is that right?

Am I doing it right? Yeah. And then this one could be 4.2. Okay? And if that were my two new lower bounds what's the new global lower bound? 4.2. So we just made progress in the lower bound. Okay?

So that's how that works. So that's – so the idea – the rough justification, I mean, this is like a depth-first search or something like that. So the rough justification of this is you're going after – pulling the lower bound up as soon as possible. Okay, so that's it. Okay, so here's an example but we've already done an example so – yeah?

Student: What happens if they return the same lower bound?

Instructor (Stephen Boyd):No problem, it's fine. You want me to do it? Here, let's go over here and do that. Yeah, so the lower bound of what? Like, you want 4.0 and 4.0? Like that? You tell me, what just happened? Tell me, what happened? That can happen.

Student:[Inaudible] the lower bound [inaudible].

Instructor (Stephen Boyd):Well, it's not worse.

Student: No, it's not.

Instructor (Stephen Boyd):It means you just called those two methods – how much did you learn?

Student: Nothing.

Instructor (Stephen Boyd):Nothing. At least looking at the lower bounds. In terms of lower bound you made no progress. It's fine, no problem. That's perfectly fine. It's nothing wrong with that. So that's – yeah, that's fine. That's just not a problem. Okay, let's see. Oh, what if the lower bound had been five on one of those? What would that mean?

Student: Just [inaudible]?

Instructor (Stephen Boyd):What's that?

Student: If one of the smaller [inaudible] of five just go back and replace it with 4 again.

Instructor (Stephen Boyd):Well, you could but you could also just label that leaf. I mean, you can now cross off that node – that rectangle, and say – can you? No, you can't.

Student:[Inaudible].

Instructor (Stephen Boyd):See?

Student: It depends on –

Instructor (Stephen Boyd):I fooled myself. Okay, all right. All right, scratch that. This would be a good time to turn on some feedback in here, by the way. I could blame it on that. Okay, no don't, please.

Okay, so this is the picture. Okay, so we've already talked about this – pruning goes like this. It says that if there's a rectangle whose lower bound exceeds the upper bound – the known upper bound across the whole thing then it can be marked – it can be pruned.

And that means you can sort of put an X – well, it means lots of things. It means actually everything you can now prove you know for sure that the solution is not in that point. And that – sorry, in that rectangle. Okay?

And that can actually happen while you're running and that would allow you to free memory, for example. Actually it would have – if you're using – if you're splitting on the lowest lower bound it – you'll never split that rectangle anyway because it's got a high lower bound. So you'll never split it. You'll be splitting others.

So, in fact, it doesn't change the algorithm in the slightest. What it does do is allows you to free memory. So what would happen is if all of a sudden a point were discovered with value – uh-oh, now I have to be super careful here.

If I – suppose the – these are all lower bounds, I haven't talked about upper bound. Let's suppose that in this set I found a point with upper bound 4.2. Now let's be super careful. By the way what's our global lower and upper bound at that point?

I mean, it's at least as good as 4, 4.2. Right? Because I have a point with value 4.2, it's in – left, right, that's top – it's in this set, it's in this rectangle but – and I know that across the whole thing the bound is 4.0, okay. Now what can I – now let's work out what I can conclude. You do this, too. I did this as the examples, not prepared and these things are tricky. So help me. Tell me, what can I conclude?

Student: You [inaudible].

Instructor (Stephen Boyd):I can get rid of what?

Student: The rectangle.

Instructor (Stephen Boyd):I can get rid of which one?

Student: The left one.

Instructor (Stephen Boyd):This one here?

Student: Yeah.

Instructor (Stephen Boyd):Absolutely. So it is now known absolutely certainly that the global solution does not lie in here. So this one is pruned officially.

Student:[Inaudible].

Instructor (Stephen Boyd):This one is pruned. That's gone. This one is now officially gone. And so that means, let's see, you're not here, that's top, you're definitely not here, that's - is it this one?

Student: Right.

Instructor (Stephen Boyd):Like that. And that says that at this point still in the running are these two, okay. Everybody got this? Now actually, by the way, you can even go up a tree farther and prune something below that. So, by the way, this is not gonna change the iterations, especially if you're going after the lowest lower bound first.

It's not - it won't - because you don't choose things with high lower bounds. What it will allow to do is call free on a bunch of these and just free the memory. It also, by the way, if somebody stops and says, "What's the progress on the algorithm?" You could say, "Well, I had," let's see, "One, two, three, four, five - I had five active rectangles two of which," or something like that. I don't know. "Before the algorithm I had five or something and now I have only two, so it went down."

And you could also talk about the total volume. In this case the total area. So you could actually say – well, actually you can say something now. You can say the global solution is in this little – it's in this rectangle here, period. Everybody see?

Student: And we just like to destroy your certificate or is there another way?

Instructor (Stephen Boyd):Have I what? Destroyed what certificate?

Student: Of –

Instructor (Stephen Boyd):Oh, if I free those?

Student: Yeah.

Instructor (Stephen Boyd):No, I store them. If this is – if it's like lawyers around and I'm gonna need it in the end to make my certificate I store a copy of them, yeah. But, by the way, had I developed this node I can safely destroy those records, right? So I can safely destroy the records that went below this but I have to save that one in case a lawyer asks me to prove when I terminate that the lower bound is bigger than 4. Right?

So I keep those records but I can destroy everything below them. So - okay. So this is pruning. And, like I said, it doesn't affect the algorithm but it can reduce the storage requirements or whatever. And then you can track the progress had been total pruned volume or unpruned, and the number of pruned leaves in the partition. So – and that gives you a rough idea of how complex the problem is, right?

Because when I terminate and say, you know, "Okay, I'm done. The lower bound is 4.11, the upper bound is 4.18," and you stop and someone says, "Really? Prove the lower bound." If I say, "Oh, yeah, no problem. By the way, it's a convex problem. I give you one dual point and I'm done." Right? Or something like that.

If it's a non-convex problem we go, "Oh, yeah, sure. Let me tell you how I know the lower bound is bigger than 4.18." You say, "Here is a partition of the original rectangle into 40,000, you know, 40,346 rectangles. And each one, by the way, I've attached a list – a lower bound for each one and the minimum of all those numbers is 4.18."

And so that's how you – that would be the certificate. Now that actually tells you sort of how complex your function is essentially and you can actually work that out. I mean, if it's something with a bunch of lower bounds, I mean, sort of minimum or something like that it would have to look something like that.

Okay, so let's do the convergence analysis quickly. So if you don't prune every time you split a rectangle you take a child – you take a leaf, sorry, you take a leaf, and then you append two children to it. So it's no longer a leaf but what had been one leaf is now two so the number of rectangles after K steps is K - the number leaves which is the number of rectangles in a partition is K.

And the total volume of all the rectangles and certainly the volume of the initial rectangles and that says that the minimum volume of all your rectangles is less than or equal to the initial volume divided by K.

It's actually probably a lot smaller but that's a lower bound. And that would only occur if you split these, you know, I mean, you could get a much, much better lower bound if you're actually splitting things in half so the volume's going down by a factor of two you could probably – it'd be a log in there or something like that. But it doesn't matter. I mean that's the – this is good enough.

And what that says – because this is gonna be a really, I mean, crappy proof. It's just gonna basically say it works and not much more because actually there's no point investing and having a fancy proof that gets a better complexity of result but they're all gonna be terrible. They're all gonna be exponential in the end. So it doesn't – it really hardly matters then who has the best exponential bound.

So this says the following. After you've done a large number of iterations there's at least one rectangle with small volume. Now we're gonna show that small volume implies small size.

Now, of course, in general that's completely false, right? If you keep splitting a rectangle – just in R2, if you keep splitting a rectangle, you know, vertically then it'll have small volume eventually and its diameter will still be the same.

So you have to – we'll see that you're gonna have to – it has to be something in your selection rule which controls for that. Now, if you have a rectangle that has small diameter then by definition it – sorry, by our assumption it says that our – the bounds there are small, okay.

Now that's interesting because when you picked that rectangle L was the leading candidate, was the lower bound. And if that was the global lower bound when you picked it and you got a U that's close to it, that means that your global upper bound is very close to your global upper bound and that means that UK-LK is small. So this is gonna – that's how that proof is gonna go. And these all require little bits and pieces of the assumptions there. We'll see.

Okay, so to prove that the – to insure that the – if you want insure that small volume implies small size then you need to control essentially the condition number. And the condition number is – for a rectangle it's easy, it's just the longest edge divided by the shortest edge.

And what you need is the following, if your splitting rule is you split along the longest edge then you're absolutely guaranteed that the condition number of the new rectangle is less than the condition number of the previous one in two.

Now there's many other rules that would satisfy this and it really hardly matters because these are not – in some ways these algorithm are in the worst case they're extremely slow. So it's not interesting that they converge, of course they converge, I mean, unless you do something stupid, they converge.

And you generally can't prove that they converge faster than exponentially so it's – I'm not exactly sure, I mean, this is just enumerating some of the intensely stupid things you could do to make this not work.

But, okay. So – and let me just explain this because it's not totally obvious but it's actually true. It's this – if I have a rectangle, let's take a rectangle like that, what's the condition number on that? Just roughly? What's the condition number?

Student: Four.

Instructor (Stephen Boyd):Four, fine. And if I split along the longest edge like this what's the condition – the maximum condition number of the children?

Student: Two.

Instructor (Stephen Boyd):Two. So I made progress. My condition number went down. Everybody – okay. Now the question is, can you split a rectangle and have the condition number go up?

Student: Yes.

Instructor (Stephen Boyd):Yeah, like what?

Student:I could split this.

Instructor (Stephen Boyd):Split what?

Student: What you had horizontally.

Instructor (Stephen Boyd):No, no, I'm sorry, yes, sorry, I meant to say this. Can you split a rectangle along its longest edge and have the condition number go up? Could you?

Student:A square?

Instructor (Stephen Boyd):Yeah, let's take a square. What's the condition number on that? One. Let's split it only – and let's say that this edge is 1.0001. Okay? So that's the longest. I split. What's the condition number on the children?

Student: Two.

Instructor (Stephen Boyd):Two. Okay. So the condition number can go up when you split along the longest edge. However, it can't go more than two. So you'd have to argue that this is the worst thing that can happen and that's not too hard to do. That is true. Okay?

So it says when you split a rectangle along its longest edge the following is true. The condition number is less than or equal to the maximum of the condition number of the rectangle you split, two. And the two would be achieved if the condition number of the parent had been one, which is to say it was a cube.

Everything cool on that? I mean, I didn't show it but, you know, it's like arithmetic. So that's what that is, okay. So that says if you use the longest edge splitting rule the condition number will never be worse than the initial condition number and the max of that in two, so period.

There's lots of others, it doesn't – the only thing you can't do is you can't keep splitting in the same direction or something like that so that the volume goes to zero and the longest – so every now and then you have to split against the longest edge or something like that. I mean, it doesn't matter.

If you bound the condition number you can bound – then you're done because you can bound the diameter. So you would have this, I mean, the volume is this thing and that's bigger than or equal to I can assume the worst thing would be – the smallest it could possibly be would be to have one of these things have the max and all the others be at the min.

That's this thing and so you can bound this by condition number is hardly surprising. And so this tells you that if the condition number is bounded, which it is if we use the longest edge splitting rule, then it says that the size of Q is small.

And so that actually finishes it. That's done. So that's the whole proof now and there's a few things I didn't show. Oh, I know it's not – I didn't formally show that when you split a rectangle its condition number is no worse than the parent or two, whichever is the worst of those two. So that's all. So that's it. Okay, so that is – that's branch and bound.

We're gonna look at one more example of it, just a different flavor, exact same ideas. This will be more specific about it. So it's gonna be mixed Boolean convex problems. These come up all the time. So it's a problem that looks like this. You minimize a function and we're actually gonna assume that these functions are all convex in both X and Z.

Now a little bit weird here because these functions don't actually have to be defined for the Z's except Boolean Z's, right? So and in fact they – if they're not defined for those you have to extend the functions FI to make sense when the Z's are in between, okay.

And there's actually many ways to do that but that has to be done. So okay, the X is called the continuous variable and the Z is called the Boolean variable. By the way, it could be a problem with just Boolean variables.

By the way, if there's no Boolean variables that's called a convex problem and we can solve it very effectively. And what happens is this, if you commit to the Boolean

variables here, so in other words if I've got ten of them and I commit to them, I take one of the 1024 patterns of Booleans this problem becomes convex in the X's and therefore it's easy to solve.

So, by the way, if N here is ten then you can always solve this problem by just taking all 1024 variations on pattern – bit patterns on Z, solving those convex problems. Right?

Okay, so that's the brute force method is you simply evaluate – you simply solve this problem for all two to the N values of Z. And if Z is less than ten or these are small problems and you have a lot of friends with idle machines and, you know, you know how to distributes jobs on different machines or something, you know, that can work fine, I mean, provided N is, like, small, like less than 20 or something like that. That'll work fine.

But – well, we'll see what that corresponds to. By the way, the partition of the feasible set is actually gonna be a discrete partition. It depends on the values of Z. Okay, branch and bound – if you run branch and bound on this what's gonna happen is in the worst case you're gonna end enumerating stuff anyway.

So it's – that's – there's no – and it's not hard to create problems where that's exactly what'll happen except with more overhead than just writing a bunch of for loops that goes through and tries every value. Because you maintain some big tree and the tree just fills up. You make a full binary tree and then you are exactly the same as if you had – you just filled it initially or something like that. But the idea is that it'll work much better and it generally does.

So in this case we can talk very specifically about how to find a convex relaxation. The simplest one by far is the linear relaxation. So you take Z, which had been in the set squiggly, you know, left bracket zero, one and now you make it in the interval, square bracket zero, one. That's the interval here.

Now, you know, if anyone – and people ask you what you're doing you can make up all sorts of cool stuff. You could say, "This is the quantum mechanical interpretation. You see the Z's before had been true or false now they have some – they're some number between zero and one." And if they buy it, you know, great.

But, anyway – or in communications you would call this a soft decision, right? And you'd say, "No, no, pardon me but I'd like to know whether that bit is a zero or one." And you could say, "I'm working on it. At the moment it's a .8." That's a soft decision or – actually, do they – they have these in these [inaudible], right? People have seen that, no, somewhere?

So that's – so it's got all sorts of names in different application areas where they make it sound like it's more sophisticated and more useful than actually the zero, one's. But anyway that's a good excuse.

Those are good techniques to learn. I mean how to relax a problem and then convince the people you're working with that although you're no longer solving the original problem you're solving something that's more sophisticated and actually has more information in it, you know?

Like in fault detection, they could say, "I just want to know if the fault has occurred or not." And you go, "Ha, that's a very unsophisticated way to look at it." I'm saying it's occurred, you know, .8 and they go, "Well, what does that mean?" And you go, "Well, it's more sophisticated. I mean, I – it means it's probably occurred." Or anyway – so we won't go into – I won't go into that. Let's go on.

Okay, now obviously when you solve this problem which is convex you get a lower bound on P*, that's obvious. And you can get plus infinity, that's actually a very, very nice lower bound because that means that the original problem is infeasible. Now, by the way, how would you get an upper bound? How would you get an upper bound for a problem like this?

Student:[Inaudible] the combinations, right?

Instructor (Stephen Boyd):Yeah, yeah. You can test any, right? By the way, how would you get it though? Tell me some practical methods for this thing. Well, here we can look at this several ways but we can – I'll go over some of them but here's one. The simplest method by far would be to round each relaxed Boolean variable to zero or one, okay, which, by the way, can destroy feasibility at which point your upper bound is plus infinity, okay.

Another option is you could round them and once you've rounded them go back and solve for X's. That can only make things better. You could, for example, restore feasibility, reduce the objective value, okay. That's your other option.

You could use a randomized method, I mean, this is – you would generate a random ZI and 01 with probability equal to this relaxed value. So now the .8, the soft decision, makes sense. So you generate random ones and try this and so on. Let's see, I could say – let me say one more thing probably more useful in practice than a randomized method which, however, will impress people. So I'd recommend it on that end for that reason.

But there's actually a much better way to do this. What you can do is you can do a local optimization. So once – assuming you have a feasible Boolean point you simple cycle through the Z's, you flip each bit. So if Z17 is one you flip it to zero and then – and see if actually things got better. If it got better you keep it.

So, I mean, you're just talking about you do some really greedy algorithm for this, can only improve your upper bound, can't make it worse. So, by the way, methods like that often work – I mean, just basically in general convex relaxations followed by the stupidest local optimization methods you can think of, the stupidest, greediest methods work shocking, just shocking, well. So that's actually way to get a good upper bound.

Okay, how do you branch? That's easy. You pick an index K to branch on and you do the following, you pick index K and you split your problems into two, one where ZK is zero and one where ZK is one, okay.

And you relax – how do you get the lower bound and upper bound's on the thing? The lower bound you relax these, you get a lower bound here; you relax these you get a lower bound here. Upper bound you can do whatever – however you like. You can round the relaxation followed by a local optimization, that's gonna work very well.

Okay, so these are just Boolean convex problem within -1 Boolean variables because you can eliminate the chosen variable if you like and you get exactly the same thing as before. So you can just run branch and bound. How do you get the new lower bound? I mean, this is the same as before so I'm not even gonna go into it because it's kind of obvious. You get these new lower bounds.

So here's the branch and bound algorithm for a mixed convex Boolean problem. You do the following, you just form this binary tree, you're splitting, relaxing and calculating bounds on the subproblems. So each – now each node is labeled – when you split the binary tree, each – the edges are labeled by a certain Boolean variable either equal to zero or one. That doesn't say the left and the right sides in the tree.

And so actually a node is very – a node in the tree is very interesting in that case. So in that case this is the wrong picture and in this case it would look something like this. There, so there's a tree and it basically says that you first split on variable 17 and so this would be Z17 equals zero, this is Z17 equals one. This is Z13 equals zero. Z13 equals one and so on, okay.

So now the meaning of a node is quite – very interesting. So it basically means – it says here if I go to this node here it means that two out of M of my Boolean variables have been fixed. Two of them, and they've been fixed to the values 1 and 1 because I took this thing.

Here they're fixed to the values one and zero here and then below this. But the others are still floating. I haven't committed to them yet so that's the picture. And if you develop the full Boolean tree then congratulations you just did brute force on it. So, okay. So there is no convergence proof because it's stupid at most when you've filled out two to the N leaves you just did brute force. So it's silly.

Obviously you can prune. Same thing, you can pick a node with a smallest L and to pick a variable split that's actually if you read about things – again, this is where, you know, personal expression comes in, so one of the places. The other is in the – when you actually implement the methods that produce the lower the upper bound. That's where the – honestly that's probably where the more important personal expression comes in.

But this is also important and there's two, actually, you'd find defended in the literature. One would be the least ambivalent. So you'd take the ones where you – or 01, the idea is

to prune that quickly, to just prove that the solution is not there. And so that's one method.

Another one is maybe the most ambivalent and here's what you do is you choose the value of K for which ZK* minus half is minimum. If, by the way, you get to – whoa, what does it mean if you get to a node and the lower and upper bound are equal? Actually, what if you get to a node and at that node when you do the – solve the relaxation it's a 01 solution in the relaxation?

It means the lower and upper bound are equal and it means you now know the global solution on that node. In other words, if those binary variables leading from the top down to your node have those values you now know the optimal value below it, okay. So but the most ambivalent would be to pick the one for which that's minimum, I mean, that's one method.

Okay, here's a small example just to see how this works. I think you're gonna do exactly this by hand on the homework. So it's the only way to really clarify your mind about these things and that way you can say you've done branch and bound. You can even have a nice pause then so the person gets it and then you can lean forward and say, "By hand."

So which very few people can say that actually. So it's a cool thing to be able to say and then if they looked shocked you can say, "I mean, I didn't solve the QP's by hand but I did do branch and bound by hand." But anyway, okay.

So here's an example, we start off – and let's just even back out what on earth this means. It's a three variable Boolean LP, this is stupid, I could have solved 8 LP's and be done with it. Okay? So I could have. But let's see what this means. Let's just see what these numbers mean to see if they make any sense. I solved the original relaxation and I found out that the – I relax all three Boolean variables I find out the lower bound is -.143. What's the infinity mean?

Student: Infeasibility.

Instructor (Stephen Boyd):What was infeasible?

Student:[Inaudible] that one was one the problem.

Instructor (Stephen Boyd):No, no, this infinity. What does that mean? It has a meaning. Oh, it means – yeah, it means infeasible but what's the precise meaning of that infinity? It has a very precise meaning. I mean, it sounds informal but it's completely precise.

Student:[Inaudible] infinity means nothing.

Instructor (Stephen Boyd):I means that whatever upper bound method I used – rounding, whatever, couldn't have been that great actually but whatever upper bound

method I used it did not produce a feasible point. In which case the upper bound remains plus infinity.

By the way, it couldn't have been a very good upper bound because there's only 8 – but it doesn't matter. So the point is it could have been just I rounded the relaxed solution, evaluated, violated constraints which are N plus infinity. I split on Z1 and I got here's 0 and 1. I got – and by the way, what does this mean? That means something, too.

Student:[Inaudible].

Instructor (Stephen Boyd):It means this says that when I solve the relaxation with Z1 equals one, by the way, the relaxation there is I'm relaxing Z2 and Z3. There's just three variables here, it's kind of a stupid problem, right?

So I relaxed those two and that LP was infeasible and you know what that tells you? That tells you that if Z1 is zero there's no feasible point. And so we will never – basically you know what that says? That tells us at that point we – at this moment we know exactly what Z1* is. Well, it's still – we still don't know, by the way, the problems even feasible. So I should say Z1* is either nan or 1 and that's gonna – we're only gonna find that out later.

But at the moment it cannot be zero. It's either nan or one, okay. So here's z1 equals one and I solved this LP, the relaxed one, and I find the lower bound is .2 which means that I've actually gained information on the lower bound but I still have yet to produce a feasible point. You know, who knows why? Because I have a very bad upper bounding mechanism or something like that.

Then I split on Z2 is zero and one, this can't be the answer and then here I get 11 and now I'm done. And let's figure out what I – how much work did I save myself? Actually, how much work did I save myself by running branch and bound?

Well, let's see, I could have just solved eight LP's and been done with it but instead I solved one, two, three, four, five, okay. So, I don't know, I cut it by a factor of two. I mean, it's a stupid example, right, but that's not a significant one, but okay.

We'll see significant ones. Now we're gonna see a significant one. So we're gonna go back, we're gonna solve the same problem we were solving in the L1 lectures. There we were doing heuristically. We want to minimize the cardinality of X, subject in a – we want to find the sparsest vector in a polyhedron.

And so you write that as a mixed Boolean LP, you have lower and upper bound on the Z's and I'll get to those in a minute. Those you get simply by bounding the box and I mean, putting a box – the smallest box around the polyhedron, okay.

So the relaxed problems are simple enough. If you exactly relax this it comes out to this. This, by the way, we saw. It's the sophisticated version of L1 heuristic for minimum

cardinality when the box is asymmetric with respect to the origin or something. By the way, if any of the ranges – if L1 is positive or U1 is negative the sign – sorry, whether or not that variable is zero or not is done, okay. So that's fixed.

Okay, so this is – we're gonna run – we're gonna split the node with the lowerest bound. Oh, there is one thing you can do. When you're solving this minimum cardinality problem the outcome of value is clearly an integer. So if I tell you that the – if I solve that LP or whatever it was and I get a lower bound that is 19.05. What is the actual? I can actual immediately say that the lower bound on the cardinality is what?

Student:[Inaudible].

Instructor (Stephen Boyd):Twenty, it's 20 because the cardinality, which is obviously an integer, it's bigger than 19.05 and the next integer is 20. So this is obvious.

Okay, so we'll do a small problem. This one has got 30 variables and 100 constraints so – but the number is, you know, these are not small numbers, right? Two to the 30 is about a billion or something. And it turns out it takes eight iterations to find a point with global minimum cardinality. That's the good news.

Now the bad news – and this is extremely typical of branch and bound. It took a lot more iterations to prove that the minimum cardinality was 19. And by the way, these numbers get even more crazy when you put in a sophisticated local search because then basically it's very typical, you find it in six iterations or something like that, you know, very fast and then 80,000 iterations later you've now proved that the point you found on the sixth iteration is actually like 5 percent suboptimal or whatever your thing is.

Okay, so this took 309 LP solves, that's including the 60 to calculate the lower and upper bounds. And so these were actually just done by Jacob and actually all the Python code is on – you do not want to write this in matlab. That would be a joke. Well, not joke it wouldn't be funny at all for you. It would be funny for other people but not for you.

So here's what sort of the tree looks like at various points. This is kind of what it looks like. And by the way, when it finally returns and says it's 19 and someone says, you know, "How –" or whatever it is, it wasn't 19, whatever the minimum cardinality is 30 something. Is it 19? The minimum cardinality is 19.

It basically says, "Look, you know, how can you prove this?" You would say something like this – you'd have to return this thing. Actually, you'll see what you would have to return the certificate proving it.

This just shows the global upper and lower bound. The dash one shows the true upper bound. You can see it starts at 12 and ends up here at 18 in 60 steps or something. And then you can't see it but it terminates right there when this thing goes up and you're done.

So that's sort of the picture and you might, by the way, this kind of puts it all in perspective. It basically says that these L1 heuristics are awfully good, right? Because you've produced a point with a cardinality of 21 and the global minimum was 19. By the way, had you done an iterated L1 or a local search here you almost certainly would have got a point with cardinality 20 and you would have got it just in a handful of LP solves. So – but it took a whole lot more effort to prove it, so this is very common.

This shows sort of the portion of non-pruned values. Out of the 2 to the 30, this shows you the fraction. And, I mean, it's not – it's just something to kind of look at and see how progress goes. And the number of active leaves in the tree; I guess it terminates at 50 something or other. So what that says if that if someone says, "Prove that the optimum – the minimum cardinality is 19." Then the way you'd do it is you'd say, "Not a problem. Here is an X with 19 non-0's that satisfies the N equality."

And they go, "Fine. That means it's no more than 19. How do you know there's not a point with 18 non-zeros?" And you'd say, "No problem," and you would return this data structure which is a tree with 50 some odd, 53 different – a tree with 53 nodes which is a partition of this space and on each one of those you would have solved this convex problem, you'd have left the evidence or the certificate for that there and you'd say, "There you go. There's your proof."

So your certificate is a complicated object in this case, okay. This is a larger example, I'm just gonna zoom through it because we're – well, we're out of time. But – and this is just in the interest of intellectual honesty just to show that if you try to solve larger problems – by the way, these are problems with just not many variables, right? Convex problems with 50 variables are joke. As you know, you can solve a convex problem with 50 variables and a hundred constraints, what are the units?

Student:[Inaudible].

Instructor (Stephen Boyd):Okay, milliseconds. I'll accept that, at the way small end of milliseconds. I would – I'm going for microseconds on that one, but no problem. By the way, thank you for saying that on the last day of the year. That makes me very happy.

So but the point is here, you know, the minute you go to the non-convex problems even like 50 variables, 100 variables, these are now very – these are large numbers. And you can be lucky, of course, and solve gigantic problems again if the gods of global optimization branch and bound are smiling on you, you can actually solve very large problems.

But you better understand it's only by luck or something like that, that that happens. So I think I won't go into this. I mean, there's a huge problem and this problem is great because it kind of – at least it sets the record honestly.

So this was run 5000 iterations and at that point here's what you know about the global solution. It is between, you know, something like 135 and 180 or something like that.

And this is really the way these – this is kind of how it – when it fails, this is what it fails like. This is the way it looks.

Notice how much better your point was with all that effort. So and this picture would be a very good endorsement of these L1 methods as heuristics. This is why it's better off saying, "No, I'm vague about scarcity. I don't really need the absolute sparsest." In which case you're way better off right down here.

So we are out of time. You're gonna hear another communication from us about the exact – we'll set the time and we'll probably give you the poster format for Monday and well, maybe we'll see you tomorrow and then maybe – definitely Monday.

So oh, and I have one more thing I have to say. I have to thank the TA's very much for creating all the new lectures. One very important thing. Someone not here but who has, I mean, helped enormously in the last two classes is Michael Grant. So some of you know this by saying something doesn't work and having the build number increment because of you, literally within an hour.

So we all owe a huge debt of gratitude to Michael Grant who – I'm not even gonna tell – he'll never even find out. Somebody will point him to this part of the lecture then he'll hear about it.

Okay, well, thanks and we'll see you Monday.

[End of Audio]

Duration: 81 minutes