>> Please stand by for realtime transcript.

>> Thank you for coming. Thank you for all of our MPS friends and other directorates that are coming to this lecture. He’s hard to get and took a semester off from Yale to travel, we really got on his schedule at the right time. He's part of our distinguished lecture series. If you have not seen it, he's the second out of six speakers. You want to look at a copy of this, let us know. Again, he's a professor of applied math and computer science with broad interest in algorithms, graph theory, and machine learning, and error correcting code in scientific computing. His title here today, actually, I really liked the first sentence of his abstract, that he will tell us the story of the development of shockingly fast algorithms for fundamental computational problems and that’s the more educated title. Go ahead, Dan. Thank you.

>> Okay. Thank you. I was told to give a talk for a general audience. So I'm going -- of course, I don't know who you all are. I am going to try to explain things at a fairly high level. If I say anything you don't understand, please stop me. That said, I will once in awhile give a side for people who I might know too much or know just an amount of. All of these terms I will define in the middle of the talk somewhere. The main one I want to get to is Laplacian matrices of graphs. But first I want to give you an overview of what is going on. The problem solving systems of linear equations, which I will explain in more depth, is a classic one studied in numerical linear algebra for a very long time. What this talk is about is about very fast algorithms for solving very special families of linear equations. They're called the -- the matrices that define them are the Laplacian matrices of graphs. And they come up in a lot of applications. I am going to emphasize the applications that come up in graph theory and I will mention few others as well. The interesting thing about this work is we're building on a long chain of results of other people. While we're solving a problem that seems like a problem in linear algebra, the tools that come to bear in this problem come from many, many disciplines of mathematics. You see the random matrix series, spectral graph theory and regular graph theory and lots of developments in algorithms, people studing random locks in graph. There are a lot of things that came into play in the solution of this problem. And then in turn in solving it, we developed a lot of other tools and graph theories areas that have gone on to have a life of their own. Other people improving them and using them for all sorts of things we never anticipated. So, my main goal of the lecture is to give you, I call it, an introduction to the characters involved in the story and to explain to you some of the relations that they have between each other and you see some interesting things that are going on. Also, you see some interesting things that have been used or put to new use. Far beyond what people anticipated. So -- oh, I should mention, just to explain my picture, this is a graph. It was one of the motivating examples. One of the things you will see in this talk is a notion of what it means to approximate a graph by another graph, one with fewer edges. This is an example of that graph, where we dropped out the edges and it turned out to be a beautiful approximation -- a beautiful approximation. And so formally my outline for my topic, I will talk about what graphs are to me in general, what we do with graphs. Talk about physical metafors for graphs, one of the main ways I understand gran graphs, anyway -- I understand graphs. After explaining the things, you get into explaining what Laplacian matrices is for. I can tell you what Laplacian matrices are. That comes in the middle of the talk. I hope you bear with me. I can tell you about linear effickians and Laplacian and other -- eq uations and maybe there is some more I forgot to stick in the outline. Okay, the first graphs are popularly called networks. People think of them in terms of social networks. That is the description I have given here. I drew a picture. They have a set of vertices or nodes and social network represent people. A graph is designed to represent the connections between them. So, particularly the connections we call edges. These are usually drawn here as lines connecting two of the nodes and that is usually how we draw a graph. Usually when we draw a graph, we have the nodes and edges connecting pairs of them. Often when we write about graphs rather than drawing pictures, I

describe each edge as a pair. Like here there is an edge between Dan and Donna. I will describe a pair put in pathen these ease with the names of the vert ease attached to them. How many of you are sort of familiar with graphs? Almost everyone. Good and that doesn't all happen that way when he talk to some of my colleagues. Yeah. [Laughter] One thing I want to emphasize is that this graph -- almost all of the graphs -- no, all of the graphs I will show you have been chosen so that I can draw them on the screen, but usually it's difficult to draw a graph. Graphs that are small you can draw but big social network, there is no nice picture of, usually. There is one way where I will be deceptiver. This is another example of graph, an air foil graph. You get this thing. It's a example of -- from a match that was drawn for studying airflow around an airplane wing. It's finer where there is more going on. This is a sort of graph that comes up in scientific computing-type applications. So, you know, usually you get 3-d meshes. Those are harder to draw. This is a graph of a mathematician I give you where I draw numbers withems going between number -- with edges going between numbers that divide them. If I get too big, this gets messy. The other thing I have deceptive here is all of the graphs are drawn on a planar. They can be drawn on the screen without crossing edges. Usually life is not that good. Almost never. Okay, one reason I show you three different graphs, I want to point out that graphs come from a huge number of applications. One danger when studying graphs is if you're working one domain, you get some ideas about graphs that you think are true and those ideas usually don't hold when you go to another domain. So graphs in different domains look very different from each other. Even worse things happen. If you take any two graphs and draw them on the screen and glue them together, put a couple of edges between them, you get this nasty chimeric graph and looks like both but not like either. One of the reasons I point pictures like this is to point out it's hard to understand what a graph looks like. Occasionally, people come by my office and say, dap, I have this -- Dan, I have this graph. Can you tell me what it is or means or looks like? I'll tell you a will about how we try to do that. Just a little. I sometimes get into lectures on this topic. The first question, can I come up with a good picture of your graph? If I can draw your graph, that tells me a lot. I understand a lot about it. The drawings I showed you before tell you about the graphs. Usually when someone is handing me a graph, they're not handing me a picture but a list of names of the vert ease and edges and that is -- vertices and the edges and that is the question, you can draw it? Unfortunately, most graphs you can't draw neatly. I will go for a river goal, sketching, or get an leave all idea of what the structure is like. You try to come up with a simple approximation of the graph that you can understand. One way of doing this is finding what we call communities or clusters. So, a community or cluster roughly should be a set of nodes that are better connected to themselves than they are the rest of the graph. And often you want to try to divide them to identify them. It helps you understand the structure, if your graph has communities and structure. Another problem I mentioned is the learning problems or inference problems that occur on graphs. That is one of the motivations for the work that I will talk about today. That is, of course -- graphs are probably the most popular thing used on computers. They show up all over the place and I am barely touching on what people do with graphs. Okay, so this is clusters, something I care about a lot. It will come up a lot. A notion of a cluster is going to be a set of vertices. Here, I'm drawing the graft. graph.The color is not great but I circle the set of vert ease. I think -- vertirk ces. I think of them as a cluster. There are edges inside and fourems leaving this one here. That is a qualitative idea. Usually I want to make that quantitive. You don't usually say this node is a cluster or not. You measure how good of a cluster it is and one way of measuring that is by measuring what I will call the cluster quality. That is the size of the boundary. By boundary, I mean the edges leaving it. Sometimes divides by the size of the set of nodes, a descent measure of cluster and quality. The lower number this is, the better the set is as a cluster. The fewerems leaving, at least relative to the size of the set, then we also think that is giving you a better cluster. Sometimes you want to divide by the number of 'ems touching nodes and not just the size itself. We'll look at something like that. This is a notion of a cluster of nodes that is going to play an important goal when we think about understanding graphs. Is there any questions? Yes. [Question Indiscernible]

>> Ah, it depends what you mean. When you say the same. We do sometimes talk about weighted graphs on top of that, I'll talk about that in a moment, where edges have different weights and strengths. Depends on more connections more important than others. There is one way of thinking of varying them. We will need to do it. But it's the first level that is easier for me to see if it has an edge in it or not rather than how strong the edge is. [Indiscernible Question]

>> Oh, if you cut off a single vertex, I don't think of that as being a great cluster unless -- [Question Indiscernible] -- oh, the remainder?

>> Yeah, the remainder.

>> Oh, I know. Oh, yes, it does. That is a good question. If you're looking at dividing the graphs into two pieces, you all want the smaller part to be your cluster. That is actually a very good question. If you told me to look at everything but one vertex, I would say that is not a cluster but the one graph and maybe measure the quality of that. That is a very important question. Thank you for missing that. -- mentioning. That learning on graphs, there is a lot of ways to do this. By Zhu-Ghahramani-Lafferty, maybe not the application they requested. Let's imagine you're trying to donate to some charity. What do you do? You call up the people and make the pitch. The people who accept the pitch and donate, that's great. You give them a one. Then other people who don't accept, probably most of them, that is a zero. You have this problem. If you keep calling people, they usually say no. You might ask yourself how you can try to find people who are more likely to be 1s than zeros? You go to Facebook and buy the social network and then you figure, huh, maybe the friends of people who are 1s are likely to buy than the people who are zeros. So now you have a problem. If I give you a draft and some nodes and I have values of a function on some nodes and this is zero, then you might try to estimate the value of the function assuming friends are similar. I don't know how good of an assumption that is, but you do it. The way I suggest is to try to assign numbers to every other node. So if they don't vary too much across the edges. I will make the notion of varying precise in a moment, but the notion of Zhu-Ghahramani-Lafferty is like this. Of course, it's a more interesting problem when you have a larger graph. This is a sort of problem from the matrices becoming useful. I want to say about the problems, which is, again, given to any fung, assigning numbers to vert ease in a -- verty cease in a graph. How do I get to the function that it looks like. Assuming that is not too much across the edges. Okay, those were two sort of applications. Let me tell you a little bit about how I try to think about these things. So, again, graphs are very abstract. For me to think about a graph, I like to pretend it's a physical object, and then I can at least deceive myself, maybe, into thinking my physical -- is helping me understand it. Here are some of the metaphors I like that are related to the applications I mentioned. One is thinking of edges in a graph as rubber bands or strings, perhaps. I like to think of the edges as pulling things together and maybe the nodes are little rings that are rubber bands and I think the behavior of the system is trying to tell me about the graph. Another metaphor we like to think of is an electrical one. Where I think of edges as being resistors and vertices are points that are welded together. And then you can think if you made that sort of metaphor, how does electricity flow in this structure? If you say it has nodes and another sort of popular metaphor is thinking about process ease that happen -- processes that happen on a graph. Sometimes -- some substance that lives on a node. Like a gas or liquid. And you puree bunk and spill -- and you puree bunch and it's going to defuse the edges and neighboring nodes. We like to look at processes like this and how they behave. That will help us understand the graph. This is closely related to what people talk about, random walks on graph. I won't explain the exact analogy, that is where you start at a node and move to a random neighbor at every step. Yes? Uh-huh. Here, I made a two-way connection for simplicity, let's

say and you can talk about connected graphs. Ridges are one way and then your life is more difficult but you can model much more.

>> You said two things were related. One thing, who myems were direct. For example, if you think about something defusing in a network, you can have a valve. If you can imagine the liquid, if you can have a valve on one of the edges, allowing liquid to flow one way like you do within your veins or you can imagine if a process is happening on a graph, you can imagine going one way opposed to another and you have directed graphs where you have directions on yourems. I'm not going to -- on your edges. I'm not going to talk about directed graphs for two reasons. One because we don't understand them nearly as well. A lot of the theory developed here. We haven't firmly developed the directed graph and, two, it's just a level of complication I want to avoid. The other thing you mentioned berk lengthical flow, different than this notion of diffusion. You think about attaching a positive terminal of a battery to one node in a graph and a negative terminal to another. You can understand that is defusion with seeing a constant flow coming in -- diffusion, a constant flow coming in, a positive charge in one way and a negative flow to the other. If you stuff died ons in here, then you -- diodes in here, you get non-linearities and continuities and graphs and don't stick inca pastors and you get manage that is intractable. At least not cleanly, anyway. We'll keep it -- we will keep it to the things you understand in a high school physics course. At least my high school didn't get behind the diodes. Okay, and let me give you a quick picture of diffusion. I don't get to talk about it much but I want to show you a simple example. I tried to put the nodes into two different clusters and I put all of my mass or stuff at one vertex here and this is one step zero. I imagine the processes where half stays put and half to the neighbors. I don't know if that is half but I tried to put dots exactly like that. You can see the stuff spread out. One thing is this cluster structure means initially not that much of the gap diffuses over here and it spreads pretty well over here first and this is actually related to one of the ways we freeway to identify clusters in a graph is examining how the process behaves with different starting conditions. By varying your initial conditions, you can find different clusters. Of course, eventually there is mass that will be diffused throughout the graph and at times it's not even. Should have mentioned understanding this is closely related to the Laplacian image. Understanding how it behaves. I haven't told you what they are but they will give us an upstanding of the behavior of diffusion. Okay, let me do this metaphor. First thing of spring. This is a simple graph. I have four nodes connected in a line. You can imagine doing the edges as I do on a linear spring, attaching the vertices. You can nail them down. Those this is what we're calling boundary constraints, hard constraints. You can see what happens, what position will the other vertices achieve when you let the system settle? They will bounce around for awhile. There is a grim position. So, physics tells us what that position will be? If you have a spring and it's stretched to length, the potential energy in the spring is the square. L squared over 2. We'll drop the one half. That won't matter much. Physics tells us the position you that get is the one that minimizes the total potential energy. Can you use the slide to introduce notation. A vertex will give us a name like a and we'll let x of a be its position and you can think of them being real numbers on the real line here and so the length of one of the edges you can write it something like x is a minus x is b. Ab is an edge and this means some leave all edges. The air of the length is x of a minus x of b squares and this is the formula for a potential energy. We'll see this formula a lot. I will explain it a few different ways. One of the main things to know is that physics tells us where the vertices will settle. The position that machine myselfs this is subject to the boundary of constraints. Let me show you my first example of this. Tutte's Theorem from 1963. Tutte proved if you had a planear graph and took one seekel in it, like this face that I joined the outside and youinally down all of the vertices in that cycle and let the rest of the vertices settle, you get a planar imbedding of the graph. Meaning no edges crossing it. Which this is a paper he wrote called how to draw a graph and applies to planar graph which I know are from a special family but it's one of the first real uses I know of this physical metaphor for graphs. This was, I think, it was the first interesting

theorem in the field of graph. It doesn't give you the best pictures. If you look at this example, if I kept putting things in here, they would get smaller and smaller and smaller together. The picture would be inconceivable and you can work on trying to fix that. Understanding the physical system gives you a way to draw the planar graphs. I want to mention one other reason I find it amazing, as a mathematician. This is a paper with theorems in it. If I were too to do this and try to prove this today, I would have run thousands of matlab experiments, with my copjections and examples and drawing them all before I would have the guts to group such a Theo Rem. -- theorem. Tutte didn't do that, I'm for sure. Okay, let me in the electrical metafor. The resistors. We have less intuition for this but a powerful metaphor. You can imagine the edges being resistors, attaching the node to one volt and the other to ground. You can look at the voltages induced when do you that and you can look at how much currents flow. There are a number of reasons to do it. One is the more -- when you have more current flow, it sort of measures the strength of the connection between the two nodes in the grasp. Not just in terms of the edges connecting them but more flow if you have many more paths between them, especially many more short paths. So the same way that edges represent connection, sort of the currents that will flow between two nodes, it can make the unit resistors measure through how well-connected they are when you take all of the edges in the graph into account. So, at least -- yes [Question Indiscernible]

>> When you talk about flowing from one to another, understand -- [Indiscernible]

>> O, if you do it electrically, there is a unique answer that electricity will take, at least. If you think about arbitrary circuit fleece, no, it's not. Physics gives us one unique answer of electricity. That is an answer going to make a lot of sense, too. So, this is the voltages you get here, by the way. Where do they come from? There is a notion of energy dissipation in the edge and a same sort of formula. V of a is what I'm assigning to node a. You will minimize the sum across edges of the square of the discrepencies of the voltages across the edges. This is the exact same formula that showed up with the rubber band and you can actually put these in one-1 correspondents, the electrical networks and rubber band networks, although in the rubber band, you don't see anything snowing. The positions of the node -- flowing. The positions of the nodes and the rubber band are equivalent to getting out here. We know what the formula of voltage is men myselfed. By the way, this is exactly what is going on in the paper of Zhu-Gharamani Lafferty. You don't vary too much across the edges, the measure that shows is exactly some same one the idea is some of the squares of the differences of the function across the edges andha is what they tried to -- and that is what they treed to minimize. They tried to minimize the same thing people with electric circuits have been men missing forever. And -- minimizing forever and this is what day looked at. This is a term coming up in a lot of places. This Tim is really what defines Laplacian matrix. I may never tell you what the Laplacian matrix is but I will tell you what we call the Laplacian quadratic form which just means if I give you a graph of vertices and v&e, what it does is tells you how nice is the function on the vertices. It tells you if I assign a real number x of every vertex, what we now do, our graph gives us a measure on that and the sum across allems of the square of the difference of the function across the edges. That is what we call the Laplacian quad ratec form. Let me give you an example. This is a very simple graph and this is a function x I have assigned to the vertices. Just map some arbitrary numbers on to the vertices and the way you evaluate this is you take a look at every edge and you take a look at the difference across the edges of the values. You sum up the differences and square them. So in this case, this graph with this function x would give us the value 15, the sum of the difference in square. So, really, this Laplacian quad ratec form, a measure of what we call how smooth the -- smooth the function is on the graph, that is how I think about graph and the Laplacian matrix. For those who know happen are algebra, you might recall this that any function like this, the quadratic form, you can write in terms of victors and matrices. And you can right it with lx, some matrix l and that is the Laplacian matrix, everything you want to know about it, its entries, the mouth base you should not

derive from looking at the actual entries of the matrix. You derive it from this, which is why I'm going to talk about the Laplacian quadratic form and we'll pretend we understand it and in reality, if you think about this, you do. Books tend to do it the reverse way. They tell you about the matrix and that is the wrong direction to go. Okay. I want to mention usually what I was talking about on the previous slides is minimizing the Laplacian. Imagining minimizing the quadratic form subject to boundary constraints. That problem always turns into a problem of solving a system of linear equations and that is a system in this Laplacian matrix. So most of you can trust me on that and for the rest of you, I will mention to see that formally, you know the minimum occurs when all of the partial dereceive televisions are zero and you set them to be zero here and you find an equation of this form. That is tomorrowally why it happens. This is who -- formally why it happens. This is one of my motivations of trying to solve e questionings in the Laplacian matrices. Ef time I want to solve one, there are a bunk of others in a moment -- bunch offing thes, the way we solve it is by solving a system of linear equations in that matrix. Let me give you another example. That is a cluster. Measuring the boundaries of sets. So let's say I have a graph and a set of vertices. I want to show you that the Laplacian is related to the number of everyones on the boundary. One way of doing that is we take our fung called the characteristic vector of S. That is one at every single vertex in the set and zero outside of the set. Now you look at what you're summing up in this quadratic form. Let me remember if I put it on the next slide. Yes. It sums over every single edge. I don't know if you can see the bottom here. The difference between the values of the function at the end points of the edge squared. Think about this, if I have the edge on the boundary, you're taking a difference of a 1 and 0 and squaring it. That is one. If you haveems that are outside of the set, you get the zeros, there is no difference, doesn't count. Edges between vertices inside the set, you both get one. There is no difference, it doesn't count. The Laplacian quadratic form enables us to count the sides of the boundary, the number of everyones going into a set leaving it. You plug in the x, the character is the vector of the set. If you wanted to measure cluster quality, which I call the size of the boundary divided by the size of the set, the same thing. You go to ratio. The quad rat wreck form up -- quadratic form up top and this is familiar to those of you who studied eigenvalue problems this is how you get them. They come up in ratios like this. How many of you studied eigenvalue problems? A good number. Great. Okay. And then you will see this formula come up by the name of a rally equation, usually and I did mention, though, that is the weighed graph coming up. Of course, you don't want to look at the graph but you want to measure what is the strength of the connection? We measure that by the weight of an edge. You can think of this as being the strength of the connection of the social networks, if it says spring network, you can think of it as a spring constant, how stiff is it or electrical networks. This should be one over the resistence. Better connected means less resistence and that is why it should be one over resistance or higher resistence means poorly connected. Like air is high resistence. I don't, if there is nothing between two vertices, we think that is in the resistance. Okay, linear equations, not many of you need reminding what they are. I put up a slide to be safe. Usually given a matrix a and a vector b, we usually write them as ax equals b. You're trying to solve for the x. Usually you can break out what the matrix looks like and it looks like a bunk of things like this, you're trying to solve it at once. X is a variable and this is one of the classic problems -- problems of lip are algebra. We learn how to do this. Some of us learn to do those problems in high school for the small ones and for the big ones, that waits until the linear algebra course sometime in college. I should mention before I go on to talking about Laplacian equations, I want to mention solving linear equations in general is one of the computational bottlenecks in a huge number of applications. There is a huge number of places where you're running codes from opization and a lot of codes and parts of specific computing where most of the computation time is going to solve the systems of lip are equations. And coming up with fast -- linear equations and faster ways of doing this will speed up a lot of applications. Also, the thing I want to point out is in most of the applications, people are not solving arbitrary to linear equations. Meaning they were -- usually they're solving them because they're somehow special. For example, usually they know there is a solution or a priority for some other reason. And it's my belief that that knowledge can often be built into the design of fast or algorithms for solving systems of linear equations. People in scientific computing do this all of the time. And people in physical sciences. It has -- this thought process is not made very far into the applications and optimization. Or in machine learning or other places where we saw them there. It's getting there and there is a lot more we can do. You know, other than for those who use math lab. You type back slash and that is in the equation. It's convenient and I love it. But, you can do much better. Okay. Here, at least from Laplacian, I want to mention a couple of classic applications. First is understanding the resistor networks and computing flows and the network. Solving eleptic partial equations. Following to a plied math matiques, if -- applied mathematics. If you see in the end of the chapter, you see he uses the right trianglations you get a system of lip are equations. If you want to compute eigenvalues and eigenvectors, the ones we care about, the low ones, not the high ones, you do it by solving systems of linear equations. If you solve the maximum flow problems a classic problem from maximum research, the linear program, one of the main ways we solve linear program is by the interior point method and, again, people just know they exist and do not know what they're doing. If you look at the interior method at every singlity rage, you're solving a system of help are equations in the Laplacian matrix. Once you know that, you can solve it faster than just using a black box and an equation solver. Also some more recent apidation -- applications. I told you at the beginning of the talk and I will tell you more, we have incredibly fast algorithms and led people to use them as comp attentional preliminaryitives. If you're designing an algorithm, let me sort my data. If you're designing an algorithm, you have a Laplacian lip are equation, let's do that. It's practically free. Now, a primitive use, you discover other things to do. Kelner, Madry and Propp came up with generating random spanning trees and others with faster algorithms and maximum flow. There are more papers coming out and a few are under submission and this is something we'll see a lot of, too. Okay. Let me go back to the basks of how to solve systems of linear equations. If you took a course in linear algebra, you learned Gaussian illegalination. That will work but it's slow usually. And you can take the time, the word case or upper bound and sometimes that happens. That, we think of, is being too slow for a large problem. That said, sometimes it's much faster and there is a lot of work people have done to try to make Gaussian Elimination as fast as possible when you can. There is code for doing that and we'll exploit some of that. For simple systems, sometimes you can do that fast. Another method you don't learn unless you take a course in numerical aljeb rat is a conjug at gradient. This is a linear algorithm. There is one shot. Gets you a solution of linear equations and can be faster. Especially if it -- [Indiscernible] Takes time to order nm where n is the number of unknowns and m is the number of non-zeros in the Patrick. The Patrick usually is few non-zeros and usually I will order a number of vertices and graphs andems. That is sort of proportional and squared. It's even faster than that. There are a lot of times that they're fast approximate solutions to systems of linear equations and I should mention bizarre examples. If we take the Laplacian over a random graph, this algorithm is blindingly fast inspite of the fact that the random graphs are complicated. Wonder important thing I want to mention is you can always check if you have the right answer. If you get an x, you can plug it in and see if ax equals b. This is important. This is a huge number of tricks that have been developed in solving systems of linear equations. Most of them people can't prove a darn thing about but use them anyway. When they're done, they can check if they got the right answer. As long as it runs quickly enough, they usually don't worry. I, unfortunately, can't begin to tell but all of the work that goes into it. Someone from one of the national labs, I think it was Bruce Hendrickson that solving linear problems is like making sausage. You don't want to know what goes into it. Okay. For those again who know linear algebra, I want to mention some of the things you can discover from the structure of Laplacian is the matrix ease and the matrix. You get one nan zero entry for the edge and the number of zeros depends on the number of 'ems you have. The null space you can understand and can ignore. If you take the characteristic vectors of the comPopens of the graph if it's connected to the vector and that is diagonally dominant. We will solve by a PCG. I don't have enough time to tell you about it. I will tell you the key point. Defind -- find what you do with the PEG. If you find an approximate problem, like an a Prock nation of the mate r or graph, it can be a crude approximate nation and you will solve repeatedly e questionings in the system. Get arbitrarily good answers to the original problems. As high accuracy as you want, you kit to the -- get to the original problem and depends on how many times you need to solve the crude approximate nation. I can tell you the approach we're telling. Vaidya's approach in 1990. He said we're going precondition, the word in numerical linear algebra for approximate. Laplacian by subgraph. If you have a graph, we'll take a subgraph and use that as an approximation and we then are going to use that to design our lip are system. One think this to know about Vaidya's work, it was never published. He gave a talk, distributed a manuscript to a few people, his paper on the topic was rejected from the conference he sent it to, his grant proposal was rejected and he said the heck with it and decided to start a company that did solve linear equations. I guess he was the ahead of his time. For someone to confidently review this work, they had to know numerical help are algebra and real graphs theory and algorithm. Maybe that person was not running around in 19 moment. -- in 1990. If you look at Bruce hendrickson's web page, he explained what Vaidya did. Let me give you the upshot. From Vaidya's work to know you we can solve to e accuracy in time the number of non-zeros times log n, there is an extra log on to our website we would like to get rid of, times log with the procedure. This is as fast as you can hope to get. If you ignored the log on to our website n here -- the log log n here, this would be the time to read the system. It was slower than sorting. So this is really fast. And I want to still you how it came up. This is the most recent result of Kirk outis-Miller And park eng. I played a role in this, a paper. It was called the nearly linear time algorithm it was some power of log-in here. I wrote 10 question marks and that should be interpreted as the last digit. The exPopent was probably $100. -- exponent is probably 100. To us, this is a great theoretical success and told us about the complexities of the algorithm and it was slower, the really, really large numbers, lower to n than any power greater than one. In reality, the hundredth power of log-in is huge. This is what I describe it as a moment that a theorist could love. But, it was not practically. This is evidence that there should be a fast algorithm. Once you know your order of complexity is not a Polly nomual n but n to log and you can think of how you get the logs down and make them lower, the interesting thing is there were many, many components that were improvable. And after each of them turned out, men interesting -- many interesting in their own light. A lot wrote papers and every year when I would give a talk about this, the constant kept getting lower and lower and lower. We finally got a one up here, thanks to them. Okay, let me tell you quickly some of the ideas that went into this. Oh, yes. A question back there. [Indiscernible Question] This is a preconditioned conjugate gradient. Yes, I will -- there is going to be a recurse of algorethems in the structure. This is -- okay. Yes. Some of you are familiar with multigrid, an algorithm developed in the specific community, a great example of an algorithm that, okay, there are few cases that you can prove it works quickly. This is the related structure. You can think of this size a proveibly correct multiple grade if you like. [Indiscernible]

>> Okay so -- so, let me explain, let me show you the beginning of the algorithm. It will look nothing like multigrid to you. You should see the algorithm. In the end, though, there is a close relation, but in the beginning, it looks nothing like it. I will show you what Vaidya did. So, to begin, let me tell you what it means to approximate one graph to another on the Laplacian. Before an approximation, should give you ine quailts. One Laplacian matrix lh is less or equal to another, less or equal to a meat rise ease. The standard one is if the value of the quad ratec form is always less than the value of the quad ratec form to the other and for all x, the quadratic form is less than the equal value of the quad ratec form in the -- this is one example. If h is the subgraph of t, you will recall the quadratic form is the sum over edges of squares. So if I throw away some of the handles, I throw away some of the squares and they're positive. I can only make it lower. If I take a subgraph, it's less than equal to the Laplacian of the original. I would

say that one graph is a t-approxmation of another. I shouldn't have put this up. Sorry. If the Laplacian of h is less than the g and that is -- that will go of me a t-approxmation. What does t times of Laplacian, you can have a number of t times in the matrix. It's look multiplying all the weights of the g graph by g and that is everything you wanted to be good, the same as multiplying the quadratic form of lt. And now I have a quadratic approximation. When you have this the preconditioned conjugate gradient -- the many thing, the bitter the approximation, the fewert rages you need. -- fewer iterations you need. The longer it takes to solve the equations. In their is a tradeoff there and to approximate a graph withstanding trees. Let me tell but trees. It's a connected graph with nodes like -- no loops like this one here. The great think this about the tree is a unique path between any vertices in a tree and makes them easy to analyze. That is another definition of a tree. Vaidya's first idea was to precondition using the tree. The other great thing, trees are matrices where you get equations in the Laplacian of a tree, you can solve it in lip are time amazingly fast. You can solve them in the same amount of time it takes to reason. And this is preconditioned with the tree. The victim of this, we said we needed to solve every linear equation and approximation. The tree with linear time and then we can measure how good an approximation we get. Since h is a subgraph, I know the Laplacian of h is less than the Laplacian of a tree. One of my inequalities to measure the approximation is easy. We need the other one. Let's go the other way. Vaidya for the looked at this and thinking about the sort of networks and measuring how well does your original graph imbed as a network into the subgraph, which is a tree. Two of the measures people traditional looked at, look I learned about in graduate school and a parallel competing class was dilation and congestion. Delation is, I look at anyem in my graph that I'm eliminating and trying to simulate in my subgraph and I look at how long the path is between the end point. In this case, it's length 6 and that is the longest. You imagine the longest of the dilation. Congestion said if I want to simulate my original graph in this one, how many times does any edge get used? Like this edge here, you can see getting it twice to support this one and this one. Also, this was an original edge in my graph. So congestion is three. And Vaidya said okay, we get an inequality here. The Laplacian of g is at most the congestion times the dilation times the Laplacian of h and that quay he took the network theory to get an upper bound on how good of a precondition he can get. He got improvement in time that you can't sea at the bottom of the board but it's faster than using conjugat, great opened. The first is by boman-Hendrickson in '01 who observed there is a better measure you can use which is rather multiply congestion by delesion. They said sum up the ranks of the path that are used. Sum up the lengths of all of the paths and that is called the stretch of a graph with respect to the tree. They proved that you get an inequality here using that. The Laplacian of the graph is at most the stretch time the Laplacian of h. Of course, can I always find a tree in which this is small? I need a better approximate make and the smaller the number s the faster our algorithm will be. It turns out in completely unrelated work, we were proved that for every graph there is a tree for which the stretch is small. I would say -- this is worked out in the competitive analysis of algorithms. They proved there is a graph. Ness is we cared a lot about this, I and others worked on improving this, we can get the stretch down to order m log m and there is a log air of log m and that will go away the next few years, I hope. And this is something, by the way, where we have algorithms we can prove things about them and there is room to make them faster. We can, I believe we can get fewer edges, and I really want faster algorithms for doing this. A lot of people have been bugging me for code for this. Seems a lot of people have graphs and don't understand them and can I at least approximate them of a tree of low average stretch and that is a pretty good approximation and man that will help them understand the graph and there is definitely more to be done here. There is one ingredient. I have to quickly skip to another ingredient called sparse e. This is -- sparsification. This is what yet means to sparse one graph to another and think of having a constant number ofs. What Teng and I asked is for every graph, we have one plus episilon approximate make. A very, very tight approximation in every graph. Here's an example. I drew the complete graph and an approximate make of it. Some of you will recognize the Peterson graph. You have to increase the weight from the edges and you get rid of edges and increase weights on others. Why did we think we could build things like this? Well, the reason was a few years before, Carter -- Benczur 47 karger did something similar. Every graph is a sparse graph and every set is a node of boundaries approximately today. That means the cluster qualities in the sparse graph are the same as in the original graph. Almost exactly. We have a one minus in faction. That means you drop edges and increase weights on the remaining ones and that is sort of our inspiration, a special case of hours. What S-teng showed in the first group is what only theorists could love. For every graph, we got a one plus episolon approximation, n to every log of n one square. Natural algorithm had a high in there in the '30s, but the existence was a cool use of random matrix theory, special graph theory, the algorithels were interesting and introduced something called local graph clustering. We use for graph decompensation and -- decomposition. We eventually came up with better algorithms. Let me zip that. Srivastava and I came up with something. The random. It's about as good as you can hope, or almost. The idea is we're going to tampel every edge with some probability and what is that? Proportionally axfective resistence between the end point. You look at the end of the graph and attach one node to zero volt and the other to one volt and see how much current flows. The one that flows less current is more important. The one with more current, the more ways to get around them, you don't need them and random with that probability. This turned out remarkably to fit exactly into the theory of mark Udelston and is what allows you to get the theory of the sparse approximation. -- we needed to solve systems of help are equations. This doesn't necessarily help us design a faster algorring theem yet. One mother thing, we know that amazing sparsefiers new exist. So Srivastava and I came up with algorithms and Polly nomual time where the number of 'ems are in and there are no more logs in there divided by the epsilon squared and we know the constant writing can be approved by factors 2. We know about the amazing desparsifiers and this is a area of work. A lot of people are asking me for those things and I don't have anything but an extreme of undergrads that are -- sparsification. The answer is we don't yet have a truly faster one that gets you a fast sparsifier. We do. I don't know about me. Sorry. Someone will get that and at least we have a good enough theory here to solve systems of equations and I can tell you briefly then of the story, ultra-sparsifiers, we compeened the low-stretch trees with sparsifiers to get something sparse. We describe it as a tree plus a few edges. I drew you a tree in black and green to draw the edges back in and this solves the equation of linear equations. Let me tell you what Kirk oh utis, Miller, and Peng do, which is simple, the way they built ultrasparsifiers, they took a low-stretch spanning tree and throwems from the original graph back in. They throw them back in with probability proportional to the path stretch that is the length of the path between the end point and the tree. The longer the edges, the longer you want to throw them back in and this amazingly simple algorithm gave a good sparsifier and ultrasparsifiers. The amazing thing is the analysis goes to Srivastava and my analysis based on effective resistence and they proved what we were doing was just way too fine in details and we did something that optimized and you could get really great results for something much cruder in this work and there is code by Yiannes Koutes we now have fast algorithms and codes for this thing. If I can take two minutes to mention one last problem that offered, the local graph clustering. This is something that came out of the work that we don't need anymore. But for this, so it still gets used in a lot of places. The question is e man know a massive graph, look the graph of everyone or facebook, the node of interest, you want to find a cluster near that node and you don't want to look at the whole graph. If you want to do this in time for proportionals to the size of the cluster, you want to grab a node and the neighbors and do this in an intelligent way that you output something that is a good cluster and you haven't read too many more nodes. So we intended to use this for graph decomposition and turns out people use yet more in social network analysis and analysis of biological network. It seems to be a useful way of finding clusters near a vertex. This has a large computational resource. So, I won't tell you how we did it. Let me tell you briefly some of the techniques that went into it. Well, S-Teng and I did some infusion, Anderson-chung and Lang did -- and is this is some good work. This is what people often use and this is better by Andersen and Perez and this is those are the duo for random walk on the graph. Something mathematicians were playing with and seem to be exactly the right way to solve this problem, which is amazing. And that is our best algorithm for this so far. There is more to come. Let me give you some conclusions. One is a lot left to do. There are a lot of other families of linear equations that occur in the applications and we want to solve them and most of the components I described to you here, we need to improve and people want to -- and people are trying to improve. I want to point out, this work would not have been possible, one, without very powerful tools developed in a lot of mathematical and algorithmic fields developed for entirely different reasons and, you know, we were fortunate to read a lot and know about them. But it would never have happened otherwise. I also want to point out this would have never happened without the framo work of asymptotic complexity. How will algorithm behave when the input is really issue really huge, impossibly large. Google is not a big number if we think about it, but because we're thinking that way, it sometimes leaves us to design faster algorithms. When I design something with an exPopent of 100 a log-in, it was a completely useless algorithm and led to new ways with the problem. People can improve and improve and refind that and get manage that is useful. -- and get something that is useful. To learn more, all of these papers and more, I have a web people of all of this that heads -- leads to all of them. If you want to see details, a lot of them are in my lecture notes. Thank you. [Applause]

>> Anyone have questions?

>> Yes, from the expert in the back. [Indiscernible Question]

>> There are no assumptions whatsoever of weight or graph structure. Graph, no assumptions about weights either, but that you do need enough presession to keep them around. But, and when you get very large weights, the problem can become ill-conditioned.

>> Right. You do need them to be positive. Right, no, there are no assumptions about weights or a graph structure no. Yes. The condition number. The copVernence of these algorithms doesn't. The convergence of these algorithms does not did pend at all on the numbers you have this this nature. -- you have in this nature. [Indiscernible] Okay, so the question, how you measure what it means to be an epsilon approximate solution, and if you measure it in the matrix norm, then, which is actually the norm you care about from most applications, then it does not become an issue. So, if you try to measure, you have to worry about the condition number. In the matrix norm, you do not, I would assert. Though you have enough presession. But. Yes. -- precision. Yes. [Indiscernible Question]

>> This Laplacian equation had nothing to do with that. I put it up because it's a pretty picture and I am not actually a competitional test is, so I don't think about airflow. That is the physics of that are well-beyond me. That gets to more complicated surface system. I just put that up as a pretty picture for graphics.

>> Well, thank you, again then.

>> Thank you. Thank you. [Applause][event concluded]