Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module # 12

Clustering Point Sets Using Quadtrees and Applications

Lecture No. # 01

Well Separated Partitioning

So, we begin this lecture with you know shifting our attention from what we look into the last couple of lectures of range searching. So, we have only done part of range searching, which is Orthogonal Range Searching. The part of range searching is more general. There is more general kind of ranges, other than the rectangular ranges and that will be covered by Professor Aggarwal at a later stage. These two topics, although they all of both are range searching, they are kind of disconnected. I mean the techniques that we used for the general range searching; it is not going to be very similar to what we have done before.

So, what we have done in the in the context to orthogonal range searching, whether it was 3 sided or you know the 4 sided, the the solutions were primarily data structure related right. So, the performance that we got out of this space, pre-processing time etcetera, they mainly dependent on the where we defined the data structures, whether it was kd-tree or it was range search trees.

Now, when we look at the general range searching, well they will have some data structures, but you know will have to be we will have to use some more you know deeper, let say partitioning techniques which use you know random sampling and geometry ok. So, there will be somewhat different, overall scheme will be same. You know you will have to again resolve to some kind of canonical partitioning, but that canonical partitioning has to be of a kind, where when you are dealing with a general kind of a range, again it should not intersect two many of those canonical partitions. Eventually, the query time depends on the number of nodes that data structure, which is essentially the number of canonical partitioning that this range intersects. That is overall thing, but for the more general case, you know that partitioning uses some very sophisticated techniques I think.

So, today you know I will start on a topic which is somewhat different in spirit from what we were doing so far and although, I never explicitly discussed this aspect. The the algorithm set we designed you know had essentially the the running time or space, whatever the performance bounds were expressed in terms of the input size, may be the input size and output size ok.

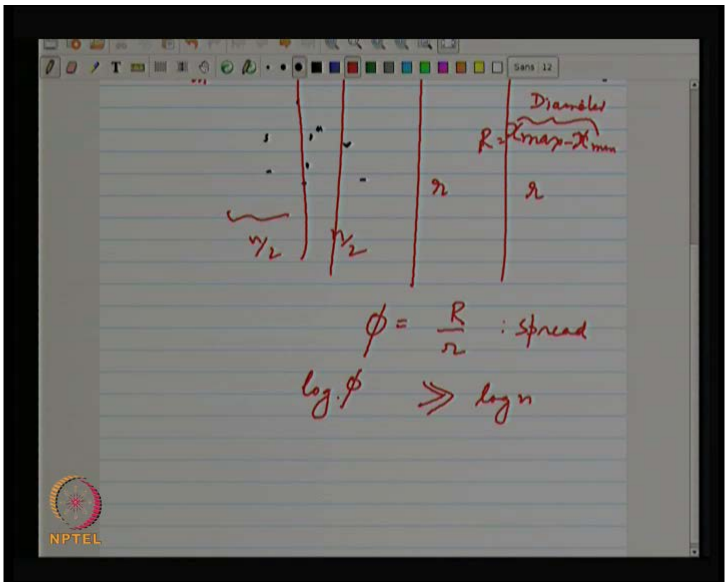
Some kind of you know commentorial expression on just, I mean you know just a commentorial expression right. The input size is a commentorial quantity, you know the output size again a commentorial quantity, but it did not have anything that is depended let us say on the slopes of the lines or you know some something that is geometric in nature, which may be a property of the input objects, but we did not want to set or make our algorithm dependent on that. A simple example would be that you know if we make our algorithms, we we could let our algorithms depend on the size of the numbers. That would be one departure or deviation from this module.

So, so for instance, when we did let us say you know something like your segmentary tree you know data structure and we were always trying to partition these on the basis of the size of the input right. You know we want about half the points in this side, half the points that side, but you know someone could also think about you know why do not we partition on basically the coordinates of the points by look at the largest x coordinates, look at the smallest x-coordinate and sort of divide you know on the basis of you know that interval.

So, we are going to then divide the interval, but in dividing the interval, we may not actually divide the input right because you know things could be. So, what I am saying is, suppose you know we had we have you know the spread of the points is like this. Let me take a bad example and this point is kind of we out here ok.

((Audio not available: 4:29-4:35))

(Refer Slide Time: 04:22)



So, we were always trying to divide using some kind of median partitioning line, which you know divides the input set of points, roughly equal parts about n over 2 and n over 2. The other option could be, you know I take the x min, x max and whatever the values are, I look at the median of that and use that to partition the set of points. Now, in this situation as this example shows, you know it may not actually have the effect of dividing the input set, but nevertheless there is some some partitioning that is happening and the partitioning is happening essentially on the coordinates or the size of the numbers that are being used in the algorithm.

One could you know again define, let us say a segment tree or range tree just based on this kind of partitioning. Then, when you do the analysis, what we show of eventually is some number that will depend on, so say r equal to x max minus, sorry x max minus x min. So, it could depend on this x max minus x min and perhaps another, so we keep on partitioning. We will keep on partitioning till you know we have, we we eventually end up dividing the points, so that the points appear in only one vertical slab. So, we have to keep dividing because eventually, you know whatever query algorithm we run, we will be run on that basis, but then the number of divisions that we require here will depend on some kind of a ratio between the maximum separations to the minimum separation. Do you agree with that? Ok.

So, this is the maximum separation, I am calling big R and the minimum separation could be some small r, which is let us say defined with the closest pair and I claim essentially, if you divide this way, then the algorithm will have some kind of dependence on this ratio. So, this this is also sometimes called as a spread. It is a, I mean I should not say x max minus x min, but something like the diameter is a better measure of that. So, this is only a very simple case, but usually it will be the diameter. The 2 points those are furthest apart. The ratio of between the 2 points furthest apart divided by the 2 points that are closest is what defines this thing called spread. If you do your partitioning based on this, on the coordinates, then it will have some kind of dependence, like roughly about log of. So, you keep dividing till you know the the point that are closest must also fall into some different intervals ok.

So, this will be again some kind of geometric algorithms, where you have an explicit dependence on the on the values of the coordinates extra input and this kind of algorithm is actually very common in practice. I would actually go as far as to say that you know people who do not study you know geometric computing and computation geometry formally, they often resort to this kind of algorithms and often, they also work reasonable well in practice. Otherwise, they will be concerned and actually they may consult someone who understands this in better.

The algorithms turn out to be simpler and often work well in practice because this, you know this point set that I just drew, this is a bad point set because its lots of clustered somewhere and you know the 2 points and a few points are very far apart but, they are kind of a uniformly distributed. Then, this this methodology where you actually dividing on the basis of coordinates would would work reasonably well, provably well actually in some sets ok.

So, in many cases because the input set is not that bad or not that skewed, even when people use this kind of thing, you know it is like this. If you are not done this course, you would probably thought about designing algorithms based on this kind of things. That is what it is. So, intuitively you are doing fine, but then when it comes to sort of you know regress analysis and and looking at the worst case, then it may not work well because you know R may be much much larger than small r, much bigger than the ratio n that we are talking about right.

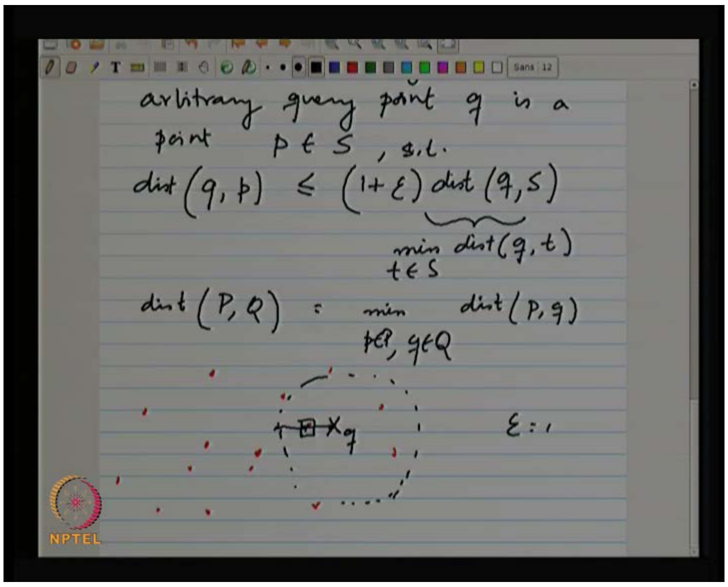
We are looking at log of n kind of things and here, you are looking for log of 5 kind of thing. So in in cases, where this quantity is much much larger than log of n or the basically 5 is much much larger than which can very easily happen right. It is basically the two values. The ratio, the two values can have ratio much larger than n and exponentially somewhere could very easily happen right.

I could have the largest separation as 2 the power n, the smallest separation could be some few some constant and then, we are talking about the ratio of n larger than log of n right. So, things could go out of hand in some pathological cases ok, but actually for the next couple of lectures, I have decided to pursue this method a little bit more and in the context note of exact algorithms, but in the context of approximation algorithms. These approximations are not really approximations of hard problems, these are problems for which you know one can design polynomial time algorithms, but primarily you know we have great difficulty dealing with higher dimensions. The moment you you take this convex hulls, you know range searching and voronoi diagram set into high dimensions, you know that the output size, everything sort of you know behaves, you know increases exponentially with dimensions.

So, when you go to high dimensions, we may not actually have the computing power to compute the exact convex hulls, exact voronoi diagram and and whatever structure is you know we have developed in the lower dimension. Although, theoretically you know they are very clean structures and they give you you know optimal performances so on and so forth, when you go to higher dimensions because of this exponential depends on d, you very quickly may be after 5, 6, 10 dimensions, you know you give up on those methods. Then, think in terms of doing something, you know sacrificing something in precision that is basically resort into some kind of approximate answers.

So, let me give you one quick example, you know what what kind of approximation we are talking about here. So, you know think about this problem. It is very very fundamental problem you know that is, build a data structure for nearest point query. Now, this is the motivation why we came up with a structure like voronoi diagram right.

(Refer Slide Time: 11:33)



arbitrary query point q is a point $p \in S$, s.t.

$$dist(q,p) \le (1+\varepsilon)\underbrace{dist(q,S)}_{\min_{t \in S} dist(q,t)}$$

$$dist(P,Q) := \min_{p \in P,\, q \in Q} dist(p,q)$$

q
$\varepsilon : 1$

We build a voronoi diagram, build a planer point location structure on that, we can give any arbitrary query point and we can report the nearest neighbor very quickly. Now, because I said the voronoi diagram, you know the complexity of voronoi diagram increase exponential high dimensions, so building or maintaining and often these are done in a dynamic setting in the building or maintaining the data structures you know becomes a nightmare. No one would ever try that you know, the space, the moment space could be exponential ok because the voronoi diagram itself is you know is going to grow exponential in space. Sorry, the space will grow exponential dimensions. So, you cannot afford to even think about storing that kind of data structures ok.

So, then one compromise we can make is that instead of looking for the exact nearest point to a query, we can think in terms of what are call approximate nearest neighbors. What is approximate nearest neighbor? So, approximate nearest neighbor will be actually parameterized by something called epsilon. Let us call it epsilon approximate nearest neighbor.

So, we so given a set S of n points, the, sorry not the, I should say a. An approximate, epsilon approximate and approximate neighbor of some arbitrary query point q is a point p in s, such that the distance between q and p, so let me write this distance is less than or equal to 1 plus epsilon times distance between q and if I can use this notation to understand this notation, it will be basically the minimum distance between q and s, the set s. The distance between a point and a set S, that makes, I may be using this later on. So, this is basically min overall points, let say t in S would become lighter ok.

(Audio not available: 15:03-15:15)

So, analogously let me, since I have brought out this using this notation, so distance between let say 2 point sets P and Q is also defined as min overall pairs p belongs to P, q belongs to Q distance. So, this is a matrix basically for the given 2 point sets, the minimum distance between the 2 point sets is basically taken the closest pair of points, where one point is in this side, the other point is on the other side ok.

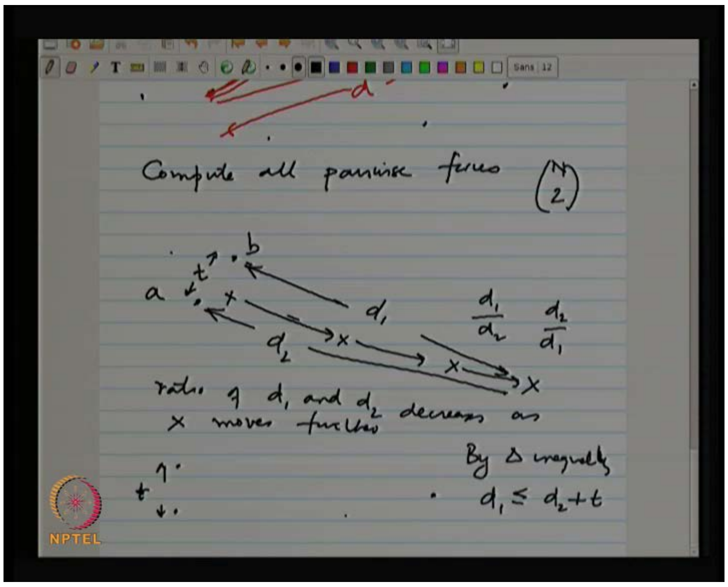
So, the epsilon approximate, sorry the epsilon approximate nearest neighbor is any point which is within that 1 plus epsilon. So, epsilon you can define, it is a user defined thing. So, if I want something within 2 times of distance epsilon equal to 1, if I want something 0.5, 50 percent of the closest neighbor, then it should be 1.5 times. So, you know just to give you an example I have the set of points and I have a query point. Let us say, this is my query point q. Clearly, this is the nearest neighbor, but then if I use say epsilon is equal to 1, I am allowed to go as far as twice a distance. So, if you take this distance, increase the radius by factor of 2, any of these points could satisfy the definition ok.

So, I think the reason why this may work out better in terms of algorithm may not be immediately clear, especially when you are talking about high dimensions. Now, this is something that may be, it will come out in the next few lectures. I may or may not be able to get as far as approximate nearest neighbor, but I will you know define certain basic data structures, which will which is used to you know compute these kinds of quantities ok. So, the approximate nearest neighbor is the very active area of research, I think still continues to be to the large extent. This is just a motivating example, you know why one looks at you know approximations for.

No, I am not reporting all of them. I am reporting any of them, any arbitrary. So, any of these points in this disc can be reported as an epsilon nearest neighbor. I am not going to report all of them.

So, another example is very practical, very famous problem. Just one, should I. Let me just mention it. I may not be able to give you the full details of that. It is called the, you know it is in the literature. It is known as N-body simulation. What is N-body simulation?

(Refer Slide Time: 18:51)



Compute all pairwise forces $\binom{N}{2}$
a
b
t
d_1
d_2
$\frac{d_1}{d_2}$ $\frac{d_2}{d_1}$
ratio of d_1 and d_2 decreases as x moves further
By Δ inequality $d_1 \leq d_2 + t$

An N-body simulation is you want to have a physical system. May be think about the solar system or some part of the universe, some basically it could come up in astronomy, it could come up in you know some kind of let us say, you know modeling of some kind of you know molecular process. So, at macro or micro level whatever, you have essentially some kind of bodies which interact using some forces, could be gravitational forces and could be electrostatic forces, magnetic forces whatever you want ok. So, you have these bodies of particles you know which you want to call it and when you simulate the evolution of the system, where there are some essential some forces acting on these bodies, they will move in a certain direction. They revolve in certain direction ok.

So, to do that accurately, you would like to, probably you know the most accurate modeling will be to compute all pair wise forces right. So, if there are N-bodies, basically capital N choose 2 pair wise forces. Well, I mean N could be very large, fine n choose 2 is still a polynomial. So, if you only bothered about you know polynomial verses non polynomial, this is not interesting case, but when you do simulations, you want to simulate over many many steps. So, if you can bring down this n choose 2 kind of step, I need to compute this pair wise n choose 2 pair wise forces. So, each step will take me about politic time and if I want to simulate it for you know the next billion years or some such thing you know this substantial right.

So, you want to do these updates. So what happen? These are these are you know these are simulations over discrete time steps. So, we simulate you know for the next nano second and then, find out the new positions and then simulate. Of course, there are a lot of errors involved. Let us not worry about the errors.

So, we are going to we are going to simulate over many many steps and for each step, you know if you require this major competition, you are clearly being slowed down. So, let say n is about a million and if you doing some like n choose 2 competitions for each steps, so it is million times million competition and if you can, let us say cut it down whatever hundred or thousand, perhaps a million whatever, then you can do that much more simulation ok. Either, you can do that much more time steps or you can do that much accurately. So, you can why nano second, why not a fraction of a nano second because the more finer your simulation times, time steps are, the more accurate the results. So, either way, so either you do it more precisely, do it for a longer time, you know it makes a lot of sense to try to somehow cut down on this computing all pair wise forces ok.

So, one technique that you know people have used very successfully is, we look at all these particles. You may not instead of computing a pair wise forces, you know this particle for instance is kind of far from all these particles and I could compute the you know the pair wise interactions between this and this, all these particles ok. When because they are very large distance away, so let us say you know this distance, minimum distance from this to this set, some distance d and the all this cluster let us say is within, let us say d over 100 something. So, what I am saying the diameter of this cluster is much smaller than the distance from this point ok.

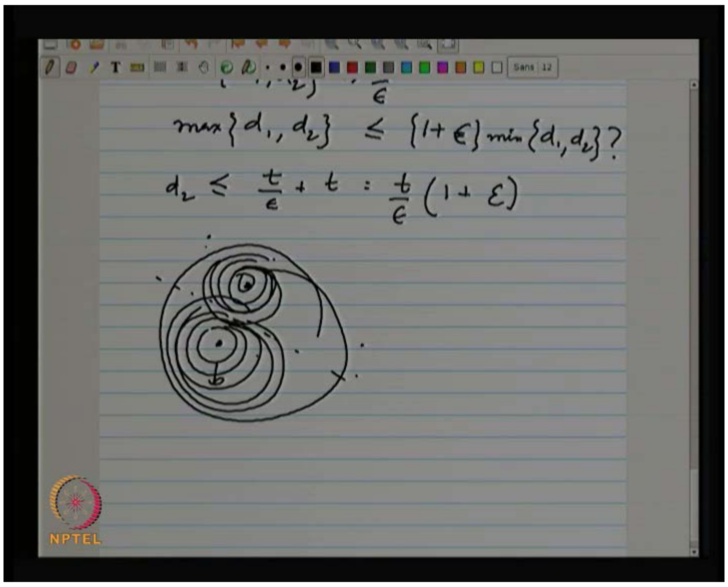
So, what would you do in such a situation? Yeah. So, whatever, whether it is gravity or you know some like electrostatic charges, you will probably replace the whole thing by some center of masses, center of gravity or center of electrostatic forces and actually the technical part of it is, it can be worked out in some kind of Taylor series. We will not get into all that.

The observation that we are making is that if you know a whole set of particles is kind of far away, then we can approximate the interaction of all these particles by a single particle essentially, you know because the difference between interaction of all these will be very very small. It depends on again, what kind of error the simulation can tolerate and if you can do that, then there is a chance that you can see these lot of pair wise interactions ok.

To give you one concrete example in terms of geometry, not not I do not calculate forces. So, suppose I have a, I have 2 points, a and b. I want to compute the nearest neighbor. So, for a query point, it is just a very trivial case of course that I want to compute given any query point which is closer. Is a closer or b closer? I can do it you know the perpendicular bisector, whatever we do with the voronoi diagram ok.

Now, as this point moves away further and further, what happens? The distances from this point to a and b becomes what? Yeah, they become the distance between the they become not very different right. So, if you move very far away, this distance, let us say d 1 and d 2, I claim that the ratio of d 1 and d 2 decrease. Well, I mean decrease or increase either way, they become very similar. So, d 1 is greater than d 2. I am saying d 1, either d 1 is greater than d 2 or d 2 is greater than d 1, either of them. So, that decreases as x moves further away. In fact, we can even quantify it ok. If the let us say, the separation between a and b is, let us say t. So, if that is t, then at a certain distance this is t right. So, we can claim that by triangle inequality right, by let say d 1 is less than or equal to d 2 plus t right.

(Refer Slide Time: 27:13)



$$\max\{d_1, d_2\} \leq \{1+\epsilon\} \min\{d_1, d_2\} ?$$

$$d_2 \leq \frac{t}{\epsilon} + t : \frac{t}{\epsilon}(1+\varepsilon)$$

(Audio not available: 26:58-27:08)

So, the question is that if you are looking at some kind of approximate nearest neighbor actually, so if we are interested (Audio not available: 27:18-27:44) how far do we go before, a and b become indistinguishable, that is I can report either a or b as nearest neighbor without and staying within this approximation factor.

(Audio not available: 28:12-28:21)

So, can you do any quick calculation on that? So, what I am saying is that d 2 is less than d 1 plus t.

(Audio not available: 28:30-28:40)

So, what you are saying is d 1 is greater than t by epsilon. So, d 1 is greater than t by epsilon, then d 2 is you are saying d 2 is further apart. So, either d 1 is further apart or d 2 is further apart right. So, if the minimum distance, I think what you are saying is the minimum distance is at least t over epsilon right. So, min of d 1, d 2 is that basically what you meant? So, min of this is this, what can we say about the max? That is what you are interested in right. I want to report either of them, can we claim this?

(Audio not available: 29:38-29:52)

So, d 2 is, suppose you know just taken example. Suppose d 1 is exactly the minimum of that exactly t over epsilon right. Let us say that is d 1. So, then this is less than t over epsilon plus t right. It is not working.

(Audio not available: 30:21-30:37)

Yeah, the better way to write it right. So, we should try in different way t by epsilon 1 plus ok. So, you know you can verify. If it is larger, it will become better ratio. So, you know if you look at this figure and I have this, let me draw the figure. We have this 2 points, a and b. So, one way to think about, so this is a perpendicular bisector right. So, as you so you can you can think about basically you know the points either on this side or that side. So, you are actually growing these balls right. You get some point to meet, but basically the perpendicular bisector right and as these balls grow larger and larger. So, outside of a certain region, basically both these points kind of have similar distance depending on the epsilon. You choose a smaller epsilon, you have to go further, but after a certain distance, basically after a certain distance, the two, the minimum and the maximum distances are very similar.

So, it got something do with diameter point set here at 2 points. You can extend the same thing with that with diameter of points. I could have a cluster of whole cluster of points. Now, I could add another point and then, again this will also add something and then, if you take a larger radius outside of that, all these 3 things will kind of look alike. So, basically something do with diameter of this point set and actually, the closest actual closest neighbor. So, you can the same kind of calculation will work out. So, what is happening is that as you move further apart, I mean from some set of points, I do not need to maintain the exact voronoi diagram or whatever the partition define the voronoi diagram.

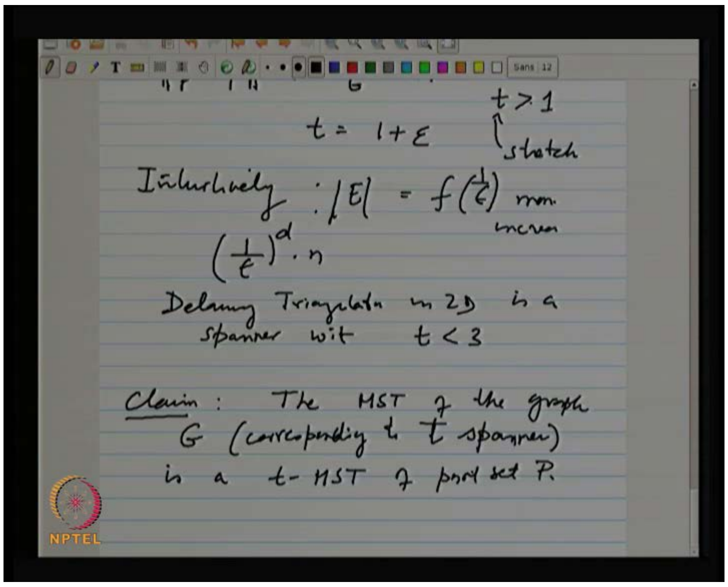
Now, after you know, after a certain distance I do not need that voronoi diagram kind of, you know it is like saying if I am really very far away from all the points, I do not care about the partitioning there. Although in the actual voronoi diagram, they will some partitioning of the space, but I do not care about it because now, I am only looking for an epsilon nearest neighbor ok. So, this is another example of this kind of (()).

Let me review my last example before we start looking at this problem in more detail and that is the notion what is called a spanner.

(No audio available: 33:20-33:35)

So, given again n points, even a set p, if the Euclidian d dimension we define a graph g where (( )) with set of what is set of points what I write p and some weight function. E is some set of edges is a weighted graph, such that for any p, q in the set capital P, the distance between the the Euclidian distance between p and q can be approximated by this graph basically. So, let me write d of g p q. So, the distance in the graph must be at least that much. We do not have any contractions and it is guaranteed to be (( )) using t. So, t times strictly greater than 1. Ok.

(Refer Slide Time: 33:19)



$t > 1$ stretch

$t = 1 + \varepsilon$

Intuitively : $|E| = f(\frac{1}{\varepsilon})$ mon. increa

$(\frac{1}{\varepsilon})^d \cdot n$

Delaunay Triangulation in 2D is a spanner with $t < 3$

Claim : The MST of the graph G (corresponding to t spanner) is a t- MST of point set P.

So, the distance could get dilated in this graph, but it will never get expanded. So, you look at the path. So, the the graph has the points of p as a vertices and there are some edges because if we do not have any edges, then the distance becomes infinity and that is not consistent with definition. We will have some weight and in fact, it must be connected with graph, otherwise there will be some point, some pair of points which is not connected in the graph and therefore, the distance becomes infinity. It cannot satisfy this bound t, t is not infinity. It is some bounded normal. So, it must be connected graph.

Moreover, if you look at the shortest path in that graph between p and q, it is no more than t times, the Euclidian distance. That is the notion of a geometric spanner the equivalent definitions for just general graph spanners. So, we are not talking about general graph spanners, we will talk about only Euclidian spanners ok.

So, now this quantity, so let me if I use t equal to suppose 1 plus epsilon. I am just writing t is equal to 1 plus epsilon. So, how does one go about constructing, so what is that advantage of this? There will be an advantage of this, only if we have, what? Why we are doing all this? This graph, trivially what kind this graph be-the most trivial graph. No, the most trivial graph is the the complete graph on this p points ok. Just put an edge with the Euclidian distance, then the graph has about n choose to edges.

What do we hope to accomplish by relaxing this is requirement that you know the distance can be dilated by epsilon. Sorry, dilated by 1 plus epsilon. We are hoping to take away some edges from this graph and still maintain some kind of bound on the distance between 2 points within the graph ok. So, that is the notion of a spanner. It spans all the points and someone said spanning tree, yes spanning trees is also going to be a spanner, but then the we do not know that what is the t corresponding to the the spanning tree. The spanning tree, in fact you can have a you know very bad spanning tree where you know.

So, just to give you an example you know you could have something like this, where you take out one edge and then, you will gone right. So, you know the stretch, the t is called often the stretch. The stretch could be very bad, but we want to have a controlled stretch. You know I will actually parameterize this, so I will supply the epsilon and will have to construct a graph that satisfies this and also, somehow not have too many edges. The whole point is not only we should have this bound, but we should also have we should not have too many edges in this graph, you know preferably may be linear number of edges. From n choose to, I want to let us say close to linear number of edges.

Yeah, saying something. That will depend on the epsilon. Yes, that will certainly depend. It is already a function of epsilon. So, intuitively the size of E, sorry not not, the number of edges will be some kind of monotonic function of 1 over epsilon in the smaller you want the epsilon is is monotonically increase in epsilon. There should the smaller you want the epsilon, clearly the the more accurate distance is you need, more and more edges you need. Some point we cannot do anything but, actually have the all edges, but epsilon is a real number, let say between 0 and 1. That is what we are interested in mostly.

If it is between 0 and 1, then you know it is going to increase, but somehow it will be parameterize with the epsilon, so that even something like this, let us say you know 1 over epsilon square times n is also constant linear ok. So, it could be something like whatever, you know so could be square, could be cube we do not know really something and most likely it is going to also depend on the dimension d. So, perhaps it could be like this exponentially more over epsilon and dimension. So, as we increase the dimension, this will become harder.

It has been pointed out in some of the previous lectures that we already know of one such spanner. Can you. It is a passing remark, but it was noted without proof of course. So, what kind of sparse graphs do you know about the point set that we have constructed? Yeah right. So, Delaunay triangulation in 2d is a spanner with t less than 3. That is what is known. Some number between 2 and 3 and even this bound, less than 3 you know has been improving over the years. Some people have, say may be prove to 2, 2.77, another person comes and say no it is 2, 2.72 strictly more than 2 and between 2 and 3, so you will have to try to improve it all that time. So, there is already something known, so in the Delaunay triangulation which is linear number of edges right. So, we have a linear number of edges, we have a constant factor in the stretch. So, it is already known.

So, in higher dimensions, we want to somehow match that and not only match that, we want to have a kind of continuous trade off. You know I could specify in epsilon, so this is for a fixed thing. I do not have any choice. I have the Delaunay triangulation, but tomorrow if someone said ok give me something that has stretch 1.5, Delaunay triangulation does not work. So, you want something, something more general, so that we can actually execute this trade-off between the number of edges and epsilon. Also, this is an exercise for you what falls out of this, out of the spanner. So, the claim is, this is I leave it as an exercise that the MST of the graph G corresponding to the t spanner.

Can someone complete this? Well, I mean no something in the context of this. So, there is a t spanner. No. Is what is called as t MST of the point set p. So, there is an exact MST which corresponds to a delaunay which is the sub-graph of delaunay is that holds in actually all the dimensions, but then I am not interested in the exact MST because constructing you know delaunay triangulations in higher dimensions is kind of nasty you know because things grow exponentially. So, I may want to have somewhat faster algorithm for MST in higher dimensions. Then, what do I do? I could somehow, if I could I do not even use, need to use the delaunay triangulation. If I could construct d t spanner, if I could construct a t spanner, not d t spanner and I just work on that graph, construct the MST of that graph, that MST will be an MST that is no more than t times of weight of the original MST.

Why we are doing all this? You know after all a set of points, define a complete graph of n square edges or n choose two edges and I can run any kind of spanning tree algorithm on that graph and construct the exact spanning tree. The real interest is that I want I do not want this n square kind of construction time. I want something much faster ok because these points in the Euclidian space, I know the geometry, so I want to do something faster. In two dimensions, we can do it in n log in time because of the connection with the delaunay triangulation. As we go higher up, also I want to see at what dimension and how far can I go till I you know I can get something reasonably sub-coordinating and this is the one method that people are adopted for that ok.

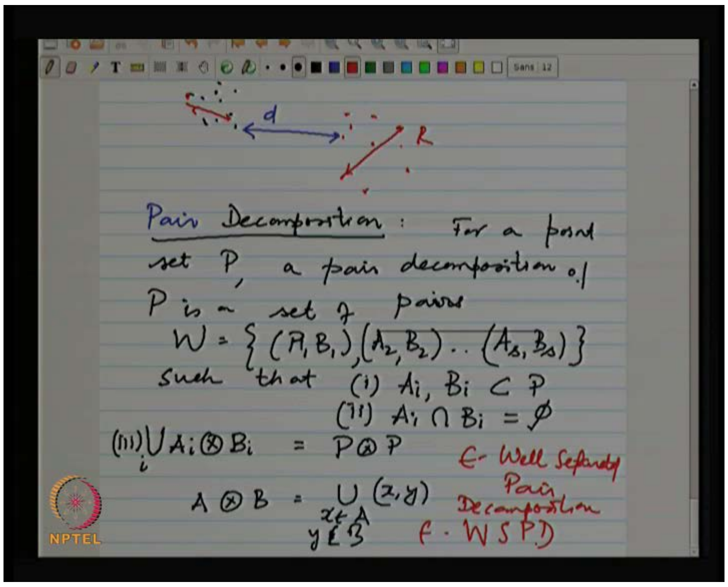
So, with this let me move over to how do we start off with. So, we started out with this notion of, well I mean can we do something, can we design algorithm that somehow I use the values of the coordinates and still not be too far away from our notion of optimality, that is in terms of running time and as well as we are allowed, we are willing to sort of live with sort of approximation.

So, I will just introduce a definition and then, we will take this up later on in the next class

(Audio not available: 46:10-46:29)

So, a pair of point sets Q and R are 1 by epsilon separated, if maximum of the diameter of Q and the diameter of R is strictly less than epsilon times the distance between Q and R. You remember I have defined the distance between 2 point sets Q and R ok. So, geometrically what this means is that I have some point set P, sorry Q. I have some point set R. You know both have both defines some kind of diameter. You know may be, these are the 2 point set further apart and these are the point that are furthest apart ok.

(Refer Slide Time: 46:34)



Pair Decomposition : For a point set P, a pair decomposition of P is a set of pairs

$W = \{(A_1, B_1), (A_2, B_2) \ldots (A_s, B_s)\}$

such that (i) $A_i, B_i \subset P$

(ii) $A_i \cap B_i = \emptyset$

(iii) $\bigcup_i A_i \otimes B_i = P \otimes P$

$A \otimes B = \bigcup_{\substack{x \in A \\ y \in B}} (x, y)$

ϵ- Well Separated Pair Decomposition

ϵ - WSPD

Now, I am trying to relate the diameter of the maximum of the two diameters. So, between Q and R has the larger diameter and the distance between the 2 point sets is the, 2 is the pair of point. One belongs to Q and one belongs to R. So, may be this is the distance. These 2 points may define the distance, perhaps the distance between Q and R ok. So, this distance must be what am I saying? This distance must be much larger, that is 1 by epsilon times this. So, this distance must be much larger compared to the diameter of Q and R which is something that we said in the context of when we said, that we now do these n bodies simulation. If these 2 point sets are very well separated, that is what basically the separation is.

The separation is being defined with respect to diameter of the point sets. If they kind of far apart, then I said you can actually approximate either the distance or some kind of forces or some any kind of interaction, basically more easily by you know some kind of doing a, depends on the context you know by the center of mass or the center of the key median or I think the geometric center I think so ok.

So, this is the definition of 1 by epsilon separated pair. So, of course the given set of points may not have this kind of property. Of course, I am given one set of points. I am not talking two sets of points. So, what is that we are interested in? What we are interested in is given any set of point, I want to somehow partition this set of points, such that I am going to produce some pairs at the end of the algorithm and each of the pairs should be well separated. So, this last notation

(Audio not available: 49:35-50:57)

Here is a notation. I will just explain this. Union of this particular operator, think about it the direct product operator. Let me first explain the third part. So, this operator P cross P, you know P operation P basically is a set of all pairs of this produced by the set of points. Here is a P direct product with itself basically is, you take anyway, we should form a definition. So, AB is equal to union of all xy pairs, x belongs to A and y belongs to B. Think that is a simple definition ok.

So, P cross P or P operator P defines all sets of pairs, point pair with itself and I have actually decomposed this set P into this A 1, B 1, A 2, B 2, up to A. So, S pairs I have decomposed, such that all these point sets are subsets of P. That is a natural thing. You know any pairs of points are disjoint and more over, if you look at the union of all the pairs produced by the individual A i, B i's, they contain all the pairs of the original point set. In addition, if I can guarantee that A i, B i are 1 by epsilon separated, this decomposition is then called the epsilon Well Separated Pair Decomposition or epsilon WSPD. This is what basically will take up next time. I will show that this way of decomposing points has applications to many things, including one that I mentioned today.