Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 12

Clustering Point Sets Using

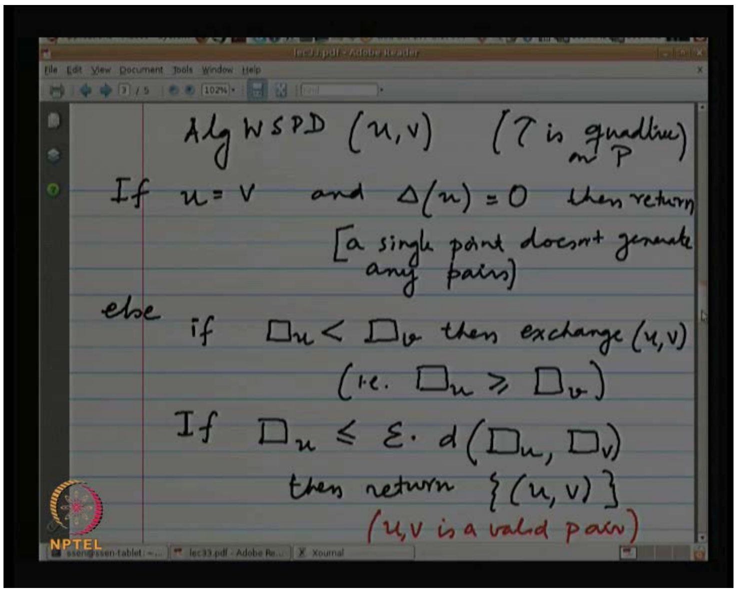
Quadtress and Applications

Lecture No. # 04

E-WSPD Construction

So, let us continue with the proof of the… a correctness and analysis of the construction of epsilon WSPD, this to give you a small example. So, I outline the algorithm yesterday, and sketched you know a kind of a not a proof, but you know some justification why it should terminate and what is the kind of running time, we will try to prove. So, just to recap the algorithm. (No audio from 01:16 to 01:36) So, this was the algorithm.

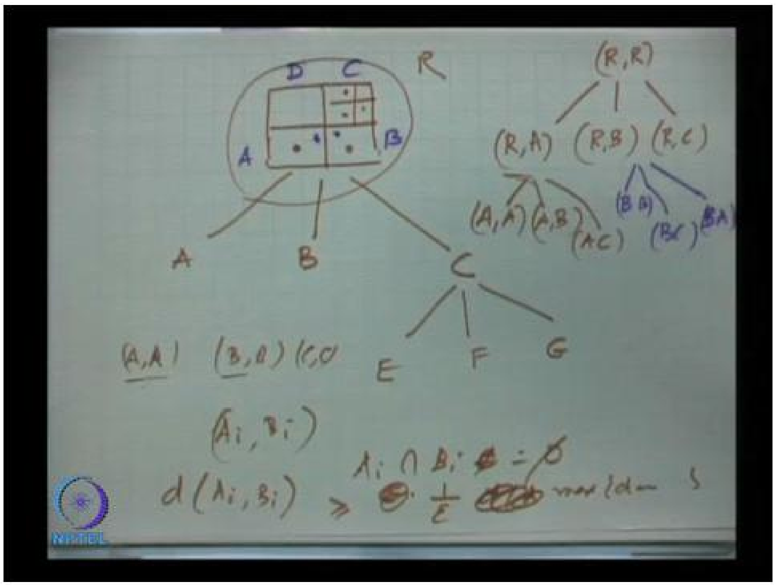
(Refer Slide Time: 00:40)



```
Alg WSPD $(u,v)$    ($T$ is quadtree on $P$)
If $u = v$ and $\Delta(u) = 0$ then return
                [a single point doesn't generate any pairs)
else   if $\Box_u < \Box_v$ then exchange $(u,v)$
                (i.e. $\Box_u \geq \Box_v$)
       If $\Box_u \leq \varepsilon \cdot d(\Box_u, \Box_v)$
            then return $\{(u,v)\}$
                (u,v is a valid pair)
```

So, if you look closely steps, the first the first test is that whether or not we are dealing with a single return point, with a single return point we do not generate any pairs, that is takes care of that. Otherwise, we compare the diameter of the two point sets, and we the kind of make sure that the first point set has the larger diameter. So, that when we call it recursively we call it recursively on the children in the corresponding quadtree, the children of the node u, and then call it recursively on the children of u, and pair them up with v(s). So, that is the entire algorithm. Let me run it on a small example. So, that becomes clear.

(Refer Slide Time: 03:03)



So, we have a reference quadtree. So, let us say that my quadtree is … (No audio from 03:04 to 03:20) this is my …Is a root of the quadtree and this leads to 3 children of the quadtree corresponding to the regions. Let us, call them A B C. So, D is an empty region. So, in the quadtree there is no child corresponding to D, what there will be children corresponding to A B and C and therefore, C itself with we will probably do another sub level of sub division. So, C may have let us say some E F G. So, it is a very simple small example, other way it becomes completely clotted.

So, suppose this is the quadtree corresponding to the point set of 5 points. So, how is this algorithm behaving? So, this algorithm starts with this recursive call (R, R), that is the initial call come the root of the quadtree and then subsequently you call… So, you tell I mean they are of the same it is a same point set. So, it going to gets split, according to the children of the larger one in this case, you know both of them are equal size. So, from here, we are going to make this calls (R, A), (R, B), (R, C) and then subsequently again, if you look at this one you will again gets split into what (A, A), (A, B). So, I am not careful about the ordering of the pairs and (A, C).

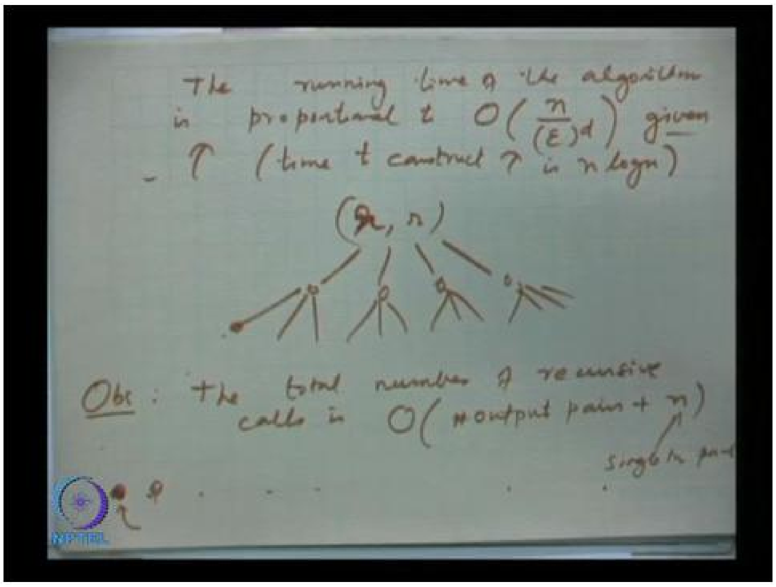
So, this is the way this is going to proceed. So, these are the recursive calls, the pairs basically correspond to the recursive calls and of course, there is a termination condition of the recursive calls, if it actually the pairs of points and achieve the desirable separation then it is output. So, if you go further and expand this one, then this again will be something like (B, B) (B, C) and (B, A). So, what you notice is that, we are generating something like you know, all the (A, A) (B, B) pairs now A B C D of course, D is an empty region, we do not need to bother about that, but in the next level or the level after that we are generating these pairs (A, A) (B, B) and (C,C) the recursive calls, we are going to call these the algorithm recursively on these pairs of points. So, what is the significance of this, one of the significance of this is that you know, if you want to actually regressively prove that the algorithm works correctly you can easily apply induction.

So, these are smaller point sets. So, if you assume that (A, A) produces all the pairs of points that are contained in the point set A, and similarly B and C then you can easily, we now just argue that you know all the pairs of points will be produced. So for, all the points in the point set A index by induction you know, we are going to produce generate the well separated pairs, corresponding to the points in A, and if the 2 points that are considering are in different point sets, suppose one is here and one is here again, because of these calls you know A B, the call A B will produce some pair one of it will compute contain this point, the other will contain this point. So, every pair of point will appear in somein one of the pairs of the well separated pair decomposition. So, this is if and a file is straight forward prove that it is going to produce all the pairs how do you prove that you know, it is going to the pairs that we are going to produce will be disjoined.

You can argue on the basis of the diameter. So, you can argue that, if the points are the point sets are not trivial since, the separation condition says that the distance must be larger than epsilon times of diameter it means that the point sets cannot have any common points, if they not disjoined you cannot have this separation. So, that argument tells you that, we will that the pairs produced eventually the (A i, B i) pairs that are produced by the well separated pair decomposition A i intersection B i will be basically empty. So, that is a argument for that, because the separation condition is such that you cannot have a point common in the 2 sets otherwise the… see the distance between the A i B i that must exceed one over epsilon times the diameter of distance between maximum of the diameter.(( ))

So, max of the diameter. So, if unless… So, if the diameter is not zero, the distance must be greater than equal to 0 therefore, there must be a separation, and if it is a trivial case you can again argue that, that will be separated, if it is one of them is you know, both the points are trivial points then by definition a single pair of point is always well separated. A single pair of point is always separated by definition. So, this is what I kind of last over yesterday, but you know, if you want to prove it rigorously this is a proof that it produces the correct result.

(Refer Slide Time: 10:21)



The running time of the algorithm is proportional to $O(\frac{n}{(\epsilon)^d})$ given

τ (time to construct τ is $n \log n$)

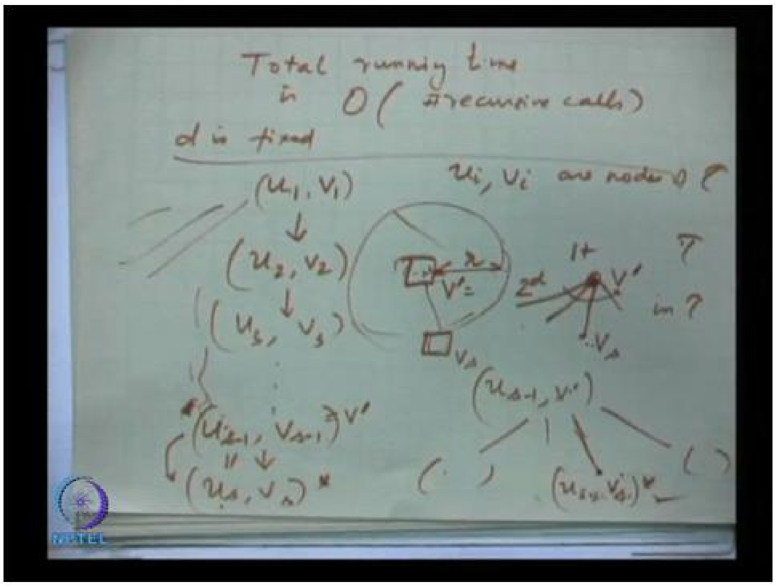
<u>Obs</u>: the total number of recursive calls is $O(\#\text{output pairs} + n)$

single pair

Now, let us get back to you know what we started with regarding analysis. So, you wanted to show that the running time of the algorithm is proportional to O n epsilon to the power d given tau. So, I am not considering the time to construct tau, time to construct tau is n log n. So, the total running time then becomes order n log n plus n over epsilon to the power d, but we will only focus on this, because this we already (( )). So, how do you do that now, this algorithm basically makes lots of recursive calls. So, if you think about the recursion tree, we are starting with this call to the root, root and that will make certain calls and this will again make certain calls, this is my recursion tree essentially.

At some point the recursion tree basically you know, you have some leaf nodes when do you stop, when does a chain of recursion stop, when you have actually produced an output pair except for the case, where there is a terminating condition with a single point, but that those single point pairs they are only n of them otherwise, a leaf nodes all correspond to output pairs. So, in the recursion tree, if all leaf nodes correspond to output pairs then the total running time or let us say the total number of recursive calls is bounded by the total number of leaf notes asymptotically, and the total number of leaf nodes correspond to the total numbers of pairs that are output. So, that is why they claim that. So, observation essentially that the total number of recursive calls is O of number of output pairs plus sum n, because of this single term points, some of them did not terminate without producing anything and those are exactly the single term points. So, this total number of output pairs is, what we are going to calculate that will give you the total number of recursive calls, and we are assuming that each recursive call takes constant time why? So, the total running time, so, I am saying it is somewhat, if it is simplification, but it justified.

(Refer Slide Time: 13:18)



Total running time is $O($# recursive calls$)$

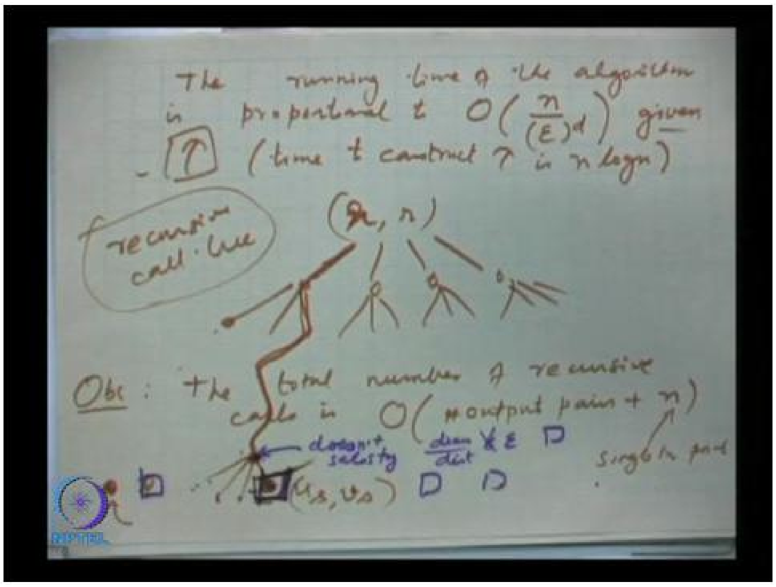
d is fixed

u_i, v_i are nodes of T

The total running time is essentially order of number of recursive calls, why can we assume that, it is only end with the falling assumption that d is fixed. So, when you call it recursively on the children, the number of children is 2 to the power d and since d is fixed, we are saying that the recursive the time to make the recursive calls is constant, and more over what we testing within each when we call it recursively, we are basically testing the ratio between the distance of the cells divided by the diameter of the cells. Now, the cells are basically our canonical cubes or cuboids in the dimensions and the distance between 2 such cuboids can also be compute in constant time, because they have size proportional to the dimension, it is exponential in dimension, but if dimension is fixed everything is fixed. So, under those assumption, the total running time is proportional number recursive calls otherwise, if you do not construct d to be fixed you just have to multiply by that, but it certainly not proportional to the number of points that is precisely the simplification we sort by enclosing the point sets by this canonical cubes.

So, we do not have to compute the explicit diameter neither, we have to compute the explicit distance between a pair of sub sets. So, we only need to compute the distance between 2 cubes and the diameter of the cubes, which are in any case you can compute by a formula. So, now, let us focus on when a how do we bound the number of pairs. So, a pair is produced essentially by a nested sequence of calls. So, let say we call them some (u 1, v 1). So, I am just looking at one such path.

(Refer Slide Time: 15:22)


The running time of the algorithm is proportional to $O(\frac{n}{(\epsilon)^d})$ given $\boxed{T}$ (time to construct T is n log n)

Obs: the total number of recursive calls is O(# output pairs + n)

So, we started with this, loop basically look at some path in this tree and it eventually stops, because the pair is being produced and prior to that, the pair being produced suppose a pair being produced here is some (u s, v s). So, prior to that clearly the termination condition is not satisfied, only then it is called recursively. So, the parent of this node, where a pair is actually produced. So, this is where a pair is actually produced, let me mark it by red. So, here are basically the leaf nodes some of them are leaf nodes, the pairs are being introduced it can be different levels of the tree of course. So, this one does not satisfy the termination condition, this condition that diameter over distance is less than this epsilon, it does not satisfy that condition here otherwise, it would have stopped an output the pair here itself.

So, the reason it went here, because it did not produce the pair here and then of course, it has more than one child it could have many other children. Now, one of the pairs is output here may be there are some other pairs also being output here. So, what I said yesterday was that, we will charge the output to its parent. (( )). From first quality of contain that (( )) that I am saying (( )). So, I will not (( )).Can you repeat the question. ((

)) first the top one that denies one recursive (( )).So, this is some chain of recursive call, that is generating this output. (( )), but this a tree. So, you can always bound the size of tree by the number of leaf nodes that is what I am saying that is all.

Asymptotically you can always bound the size of the tree. So, that is why I am only counting the number of output pairs. So, I want to somehow count the number of output pairs, and how do we count the number of output pairs, I would actually charge this output to the parent. Now, the parent is not in this, this is the recursive call tree kind of… So, this is very distinct from tau, tau is my quadtree, this is basically my the way the recursion un folds that also has been captured by the tree. So, to count the number of recursive calls, I am looking at the tree of the recursive calls, but when I do this charging I will actually refer to this tau.

Every pair of node here in this is a node in the quadtree. So, whenever I make a call. So, let me write this tau. So, whenever (u 1, v 1) from (u 1, v 1) I make a call to (u 2, v 2) and then (u 3, v 3) and finally, I come to some (u s, v s), each of these (u i, v i) are nodes of tau and moreover, when we make a recursive call actually one of the nodes is different, one of the nodes remains same, because the larger node gets split the other nodes remain same. So, it is possible that v 1 equal to v 2 and only an u 1 has been split to u 2 you know they are many children. So, this is not the one children. So, u 2 is one of the children of u 1 in tau.

So, if this is an output pair, I want to charge this to some node in tau, because I know the tau have size a linear size order n size you know that, the quadtree has order n size. So, I am going to charge this output node to one of… So, both u s and v s are nodes in tau, the parents let us say u s minus 1 and v s minus 1, they are also nodes of tau and of course, one of these two are same either, v s minus 1 is equal to v s are u s minus 1 is equal to u s. So, let us assume without of loss of generality that u s minus 1 is equal to u s and all these are nodes of tau. So, there is some node let us, call these v prime. So, just a minute.

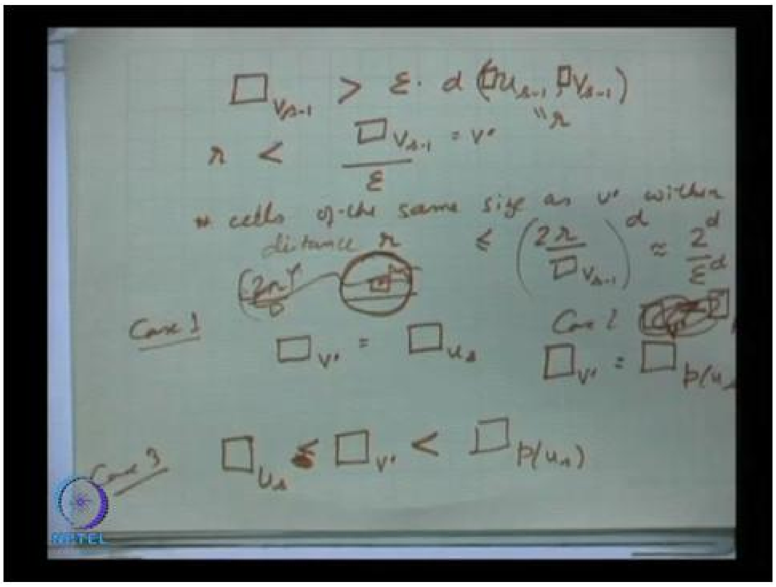
So, which one should be consider to be same. So, if... So, suppose this is same. So, v prime has a child v s in tau, it has other children also. So, this is I am referring to tau now, v prime must be the parent of v s in tau, that is why we could go from this recursive call to this recursive call given that u s minus 1 is equal to u s. So, this must have been the larger this set must have be the larger sub set I mean in terms of diameter. So, we have split that. So, we split that and the number of children basically is about maximum of 2 to the power d in d dimensions.

So, we had some kind of… So, u s minus 1 v prime, makes lots of calls, one of the calls is to u s is equal to u s minus 1. So, I could write u s minus 1 or u s comma this child, which is u s and v s is actually child of v prime. Now, I want to… And this is an output node now, in these things also there may be some output nodes, I want to charge this to v prime this output node is being charged to the node in tau whose child is v s. So, one of these nodes change. So, I am assuming that v s basically is the child of v prime and this has been actually output by the algorithm.

So, I am going to put this charge to v prime. So, v prime will be charged 1 plus 1, because of this output 1. So, that is a charging scheme exactly. So, now, we have to count, how many charges accumulate in a certain node of the quadtree. What is the maximum number of charges in the quadtree that will give us, the bound on the total number of output pairs? So, how do we get a bound and how many charges will accumulate on a given node of quadtree and that well I mean…

So, there are some case analysis I will do one or two I would not do the entire thing, but the basic argument is the following that. So, let us… So, this v prime of course, corresponds to some cell a d-dimensional cell and v s is a d-dimensional cell, we are defining to the quadtree. So, this is some d-dimensional cell and this is some d-dimensional cell let us, call them delta v prime and it is a square, the cell correspond to that v s and this must be a subtle of this in the quadtree. Now, at this point or this node no pair was generated; that means, the termination condition is not satisfied, in other words I can claim from here that, we have to see here can you see both now (( )). So, I know at this point the termination condition was not satisfied.

(Refer Slide Time: 24:49)



That is delta of v s minus 1 this diameter was larger than epsilon times, the distance between u s minus 1 and (No audio from 25:08 to 25:31). So, the distance between these two cells. So, actually I should use cell and cell, the distance between the cells or suppose this is equal to some distance r, which means that r is less than delta v s minus 1 over epsilon. So, the distance between the two cells, in the previous level was less than some quantity and that is why you know, it did not produce an output it had to make a recursive call now, if you recall yesterday is discussion, what is the number of cells at distance r from this v s minus 1. So, v s minus 1 is equal to v prime is also equal to v prime. So, we have this v prime this is a subtle of v prime and this subtle a v prime of course, you know a pairs of with u s and produce an output. Now, how many such cells can be there is what we need to get a hand alone.

So, v prime look at the cells at the same level. So, at a distance r, you look at all the cells of the same size in the grade. So, when r is less than this distance, the number of cells of the same size as v prime within distance r is less than or equal to r over v s minus 1 that we correct it that 2r to the power of d and so, this is about 2 to the d divide by epsilon to the power d. So, this is the observation by which we are going actually bound the number of charges that will be accumulated in v prime and there is some case analysis I will do just one case, and the rest you know well we will we have to again argue, but that will basically sort of give you the handle about how do you get a bound on this. So, one case that I will argue on is (( ))

This one is very simple. So, I take a cell I look at the …I will this distance r, and I have the size of a single cell. So, 2r divided by whatever done, 2r divided by this the size of cell to the power d, it is a hyper cube in the dimension that is all. So, I am trying to get a size of this sphere by I am approximating this with this, this is a bad approximation, but you know I am leaving with it. (( )). It is a same thing I am just using the notation v prime. So, that I can avoid using the substitute that is all.

So, I will deal with one case, and that is when the diameter of v prime is equal to the what the cell of the v prime is equal to the cell size of u s. So, this is one possibility, other possibilities are when v prime is equal to the cell of the parent of u s and, the third case is when it is in between. So, case 1 is this case 2 is something like v prime equal to below it is called this is the parent of u s, this is one possibility that these are in the same level. So, we are these two cells at the same level of the grade here, we are saying the v prime in u s are in the same level of the grade and the other cases that when case 3.(( )).

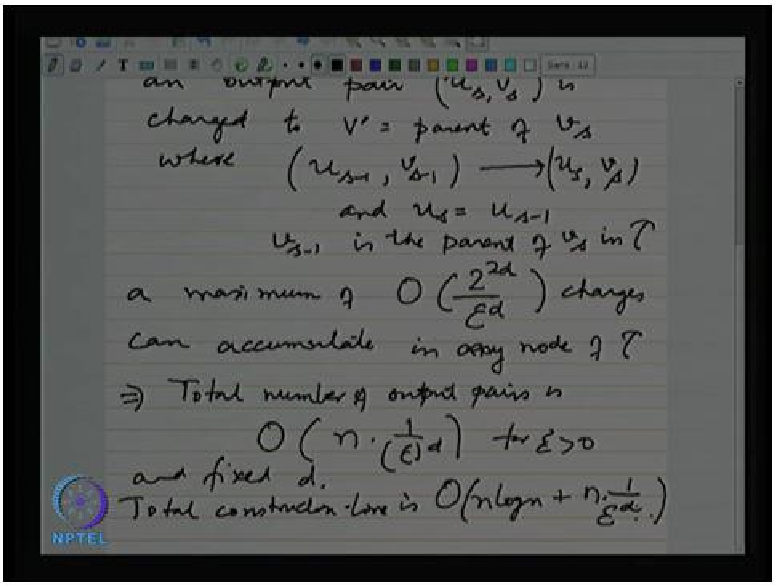
So, the cell v prime is at the same level, the cell of v prime has the same size as the cell of the parent of u s. So, let me write cell of v prime has the same diameter as the cell of the parent of u s, and third case will be in between that cell of v prime is strictly greater than u s and strictly less than p of u s. So, these are the cases this that basically, we deal with and again let me refer to this figure that v prime is at the same, but this cell is at the same level acts as u s is one of the cases, this cell is at the same level as the parent of …at some point we will have the parent of u s minus 1 you know, if you go up the tree at some point this must be replaced by the parent. So, either this cell has the same size of the cell of the parent or this cell has the same size as the u s itself or it is somewhere between these are the 3 cases. So, I am looking at the case when this cell has a same size as a cell of u s. So, I have to focus then on all those u s that can result from this kind of configuration I know that then the u s has a same size as this.

So, it must occur within this radius. So, the total number of cells, for which such an output can result is bounded by this quantity, because there are so many cells that are within distance r. So, this is how you can bound for this case. So, if these are the same sizes then you bound the number of cells that are within this distance from v prime, and only those can be paired with v prime as an output and that can be charged to v prime. Similarly, here also you can argue, when there is a parent actually the parent will have, if this is the number of parents will be bounded by this, but each such parent will could have 2 to the power d child. So, in the case 2 whatever, we bound we got for they have to be multiplied by 2 to the power of d and then in the case 3 again this and this case will be very simple.

So, again it is a packing argument that you have a cell v prime, and v prime is being paired with some output u s. So, for that to be output, we have to stay within this radius, because otherwise, if you go beyond that then it would have been output in the previous level itself. So, because it did not terminate the recursion did not terminate it must be that they are very close. So, how many cells can be very close you know, it is bounded by some quantity like this and those are the number of output pairs that can be charged to this node of the free tau. So, v prime is a node of the tau.

So, maximum number of charges that v prime is going to accumulate is no more than roughly this quantity and that is the proof, because there are only order n nodes each node can accumulate at most so many charges. So, the total number of output pairs is no more than n times well 2 to the power d is being node, because I have said, if d is fixed I can ignore node to power d. So, this is the proof of the running time of the proof of the number of pairs that are produced.

(Refer Slide Time: 34:47)



an output pair (u_s, v_s) is charged to $v' =$ parent of v_s

where $(u_{s-1}, v_{s-1}) \longrightarrow (u_s, v_s)$

and $u_s = u_{s-1}$

v_{s-1} is the parent of v_s in τ

a maximum of $O(\frac{2^{2d}}{\varepsilon^d})$ charges can accumulate in any node of τ

$\Rightarrow$ Total number of output pairs is

$O(n \cdot \frac{1}{(\varepsilon)^d})$ for $\varepsilon > 0$

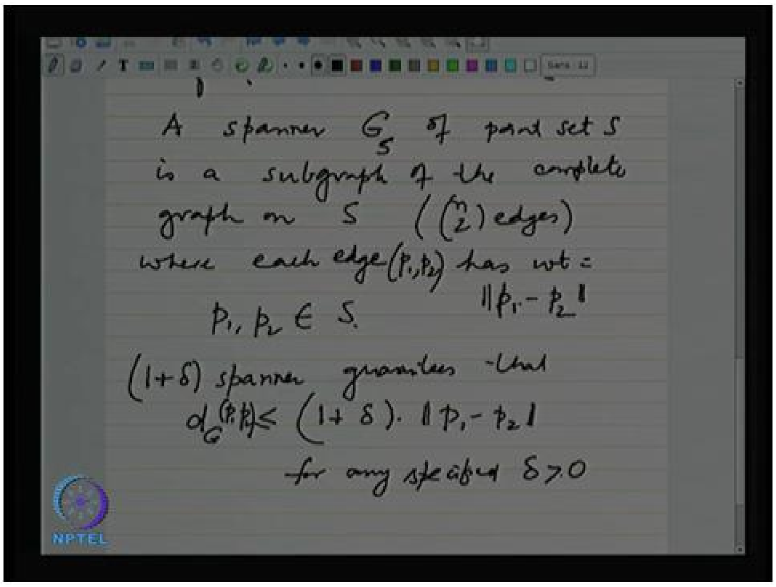
and fixed d.

Total construction time is $O(n \log n + n \cdot \frac{1}{\varepsilon^d})$

Any questions. So by the charging argument, where an output pair is charged to v prime equal to parent of v s, where u s minus 1, v s minus 1 calls (u s, v s) and u s is equal to u s minus 1 and v s minus 1 is the parent of v s in tau. So, let me do this charging argument a maximum of order 2 to the power… So, I will just to be safe you know, I will write some slightly larger quantity let us say, some 2 to the power two d over epsilon to the power d. (No audio from 36:20 to 36:37) In prime total number of output pairs is O of n times fixed. So, d is fixed then this is the number of output pairs, and this is also the time to construct the WSPD given tau, but then if you are not given tau.

So, total construction time is order n log n this part is for the tau plus n times one over epsilon to the power d for... So, for fixed d and reasonably you know not too small, and epsilon essentially you get n log n. So, for small dimensions like 2, 3, 4 now, you can certainly argue that this is basically linear, which is you know which is quite remarkable actually. So, you are getting a linear sized output out of this well separated paired decomposition that is what is lies the power of this composition of course, you know it does not it grows explanation with d. So, that is not good. So, it is not going to particularly useful for high dimensions, but with relatively low dimensions you can achieve this.

(Refer Slide Time: 38:51)



A spanner G_S of point set S is a subgraph of the complete graph on S ($\binom{n}{2}$ edges) where each edge (p_1,p_2) has wt = $\|p_1 - p_2\|$

$p_1, p_2 \in S$.

$(1+\delta)$ spanner guarantees that

$d_G(p_1,p_2) \leq (1+\delta) . \|p_1 - p_2\|$

for any specified $\delta > 0$

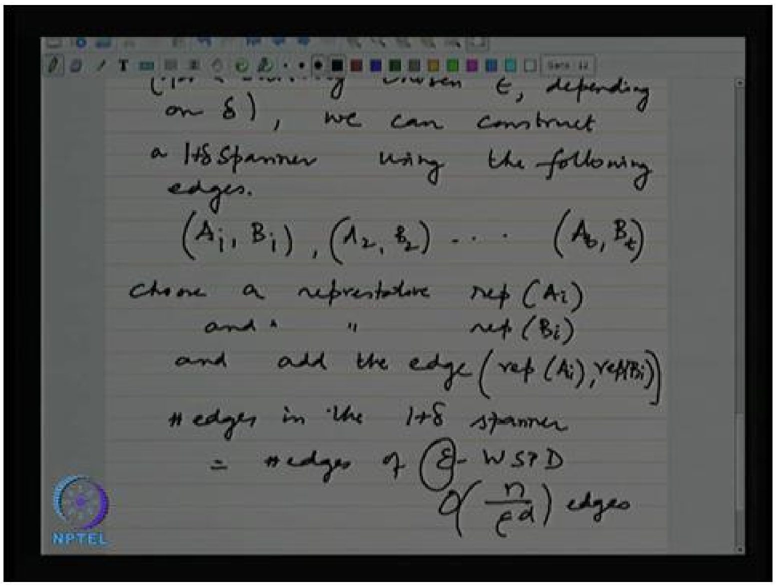
So, let me now show you one application of this WSPD. In fact, I did mention it as the motivating example, and that is how do you construct? So, given an epsilon well separated paired decomposition of point set S construct a 1 plus delta spanner of S, to recall the definition of this spanner, if you do not here is in. So, a spanner let us call it G sub S of point set S is a sub graph of the complete graph on S, which has basically n choose 2 edges, where each edge has weight equal to the distance between… let me write equivalent distance each edge p 1 p 2 each edge p 1 p 2, where p 1 p 2 belongs to S. So, this is a spanner and a 1 plus delta spanner guarantees that the distance in the graph is no more than one plus delta times the euclidian distance.

So, no big deal, the big deal is that we also want to control the number of edges in this graph is to make a truly a sub graph. So, the complete graph on this, the implicit complete graph actually it is an implicit complete graph, because I do not really have to draw the graph I have the set of points, and for any pair of points I know that the edge has weight exactly equal to the euclidian distance between the pairs of points. So, if I retain all the edges of course, you know I satisfied this property with delta equal to 0, but then what we want is to be able to actually throw away some edges to have a strict sub graph of this complete graph, and still we able to guarantee this kind of a distance bound for any pair of vertices.

So, I am going to throw away this edges and I should be able to guarantee that the distance even after throwing the edges does not get dilated by more than one plus delta factor for any delta greater than 0, for any specified delta is strictly greater than 0. Now, the same definition or the same instrument can be defined for a genetic graph, this is a implicit complete euclidian graph. Now, I can give you any arbitrary weighted graph, and I can demand the same thing can you construct a spanner know that guarantees that the distances will not be greater than let us, I throw some edges, but let us the distance will not increase the more than a factor of 2 at the same time you have to guarantee that the number of edges in the graph is no more than such and such.

So, those actually also have been you know, tackled fairly well in the literature, but we are going to only focus on this a special case of this, which is a geometric graph. So, of the geometric complete graph and therefore, you can actually exploit the geometry of the euclidian space. So, this construction is very different from the construction of genetic graph how do you construct spanners for graph that is a very very different kind of construction in fact, we cannot even guarantee those such spanners exist for small delta usually delta is greater than one for genetic graphs. So, we can only achieve when delta is about 2 or so. So, we cannot get anything, we cannot always get a spanner that is, which is strictly a sub graph is the expansion is not at least 3. So, that is with certain number of edges of course you know, if you have to throw some edges. So, here we will only look at this point set, the implicit graph defined by a point set and how do you do this construction, it is absolutely trivial given the WSPD. So, given the WSPD. So, how do you construct that.

(Refer Slide Time: 44:34)



[illegible] ϵ, depending on δ), we can construct a $1+\delta$ Spanner using the following edges.

$(A_1, B_1), (A_2, B_2) \ldots (A_t, B_t)$

Choose a representative rep(A_i)

and " " rep(B_i)

and add the edge (rep(A_i), rep(B_i))

edges in the $1+\delta$ spanner

= # edges of $\boxed{\epsilon}$- WSPD

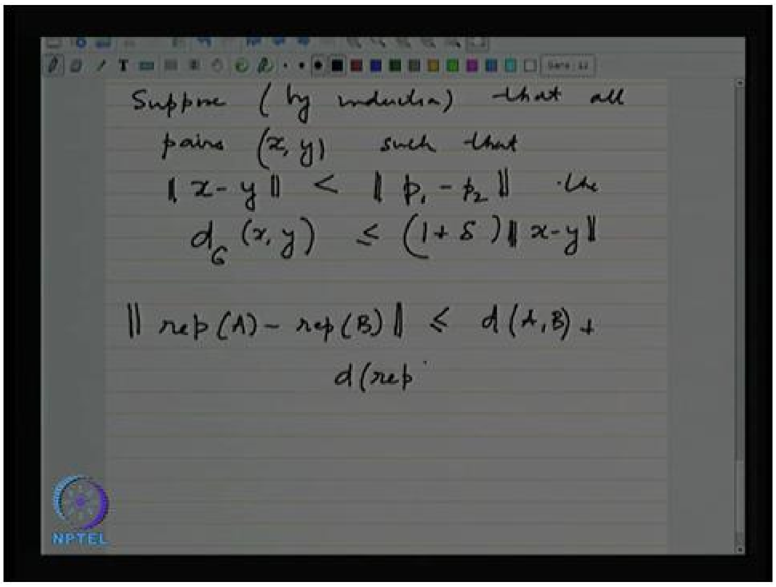
$O\left(\frac{n}{\epsilon^d}\right)$ edges

So, given a WSPD or let us say given that we can construct a WSPD (( )).Let us say an epsilon WSPD of S, the point set S for a suitably chosen epsilon. So, epsilon we are going to choose depending on what on delta, and that is the calculation that we will end up doing depending on delta, we can construct a spanner one plus delta spanner using the following edges. So, I have now basically define the edges of this graph. The point set is the set of vertices and I must define the edges and what are this edges you look at. So, look at these pairs produced by the WSPD (A 1, B 1), (A 2, B 2) etcetera (( )) and for each such pair what we do is, we choose one representative of A I, and one representative of B i and draw an edge between this they actualize. Choose a representative rep of A i, and representative rep of B i and add the edge.

So, this is basically my edge thus a set of edges, the set of edges is defined, why should it be a spanner, how many edges are there first of all .(( )) points in side of particular (( )).This is my construction now, we have to argue that this see unless they are connective it cannot even guarantee one plus delta this thing, if the graph is disconnected then; obviously, the spanner property is not guaranteed for some of the proof will basically lead to everything, but this is my set of edges.(( )). Well I mean the …if you think about the intuition is that. So, the proof will actually give you the… why it is working on the proof is not the difficult. So, I actually go through proof. So, how many edges do we add by this policy, the number of pairs.

So, the number of edges in the spanner equal to the number of edges of the epsilon WSPD, but we do not really know the value of epsilon this point, it some function of delta we do not know, but if it where constant if delta is a constant and also epsilon is a constant assuming that you know, epsilon does remain a constant. It some function of delta a v delta square a v 2 to the power delta something that we can claim from earlier observation that this is n over epsilon to the power d edges, and again for d an epsilon not too large, what we are guaranteeing is linear number of edges. So, from your n choose two edges this is practically linear number of edges. So, why does it work why does these thing work, and we just try to write up the proof. So, you have essentially this A i B i's.

(Refer Slide Time: 49:43)



Suppose (by induction) that all pairs (x, y) such that $\| x - y \| < \| p_1 - p_2 \|$ the $d_G(x, y) \leq (1 + \delta) \| x - y \|$

$\| rep(A) - rep(B) \| \leq d(A, B) +$ $d(rep$

A i, B i here is some representative here is the representative of B i, it does not matter which one which use as a representative and here are some arbitrary points let us say p 1 and p 2, we have added the edge (( )). I am not doing this, I am only added this edge according to my construction I have only added this edge, this is the edge that exist for this pair of this pair A i, B i of course, other edges may exists, because of other pairs. So, now, I only have this edge for this pair and the proof is by again by some kind of induction. So, suppose again by induction that all distances strictly or let us say all pairs (x, y), such that distance between x and y is strictly less than distance between p 1 p 2, the graph distance. So, d G of this spanner (x, y) less than equal to 1 plus delta x minus y.

(No audio from 52:01 to 52:22). So, representative let me call it just A and B. So, that I can avoid this subscripts rep, I am not sure I can actually complete the proof, let me try. (No audio from 52:45 to 53:33). So, let me not even start with, because I do not think I will be able to finish you know we are running out of time. So, I will complete the proof next time, but essentially you know it is just writing out some of these equations and use triangle in equality nothing beyond that and this induction and it comes through. So, the construction of this spanner is basically trivial once we have given the WSPD, and the WSPD is also kind of trivially constructed from the quadtree, because algorithm is very simple after that.

The drawback of all these is, we are paying a price exponential in dimension, but if you are restricted when you restrict yourself to low dimensions and not very small epsilon, this gives you something that is quite remarkable, which you know you would not have probably thought of you know it is not clear how to even start this process, we have told you that here is a set of points, build a spanner with guarantee 1 plus delta and that has only linear number of edges, this is very very non- intuitive, how whether or not such a thing exists first of all. So, this is also a proof that it exists and not only that, it can be constructed fairly efficiently in about log n time, at some point we did not mention that the delaunay triangulation itself, if you look at the edges of the delaunay triangulation, the graph defined the edges the delaunay triangulation or the set of points that itself is a spanner, but the delta for that spanner is not has small here, I can choose delta I can make delta as small as I want.

You know points 1.001 except of course, I will be paying some price in terms of the epsilon. So, it is like n over epsilon to the power d. So, the epsilon will presumably becomes smaller as delta becomes smaller. So, you will be the number of edges will scalar, but for the delaunay triangulation, which has just about some three times n edges these the dilation that you obtain you know for the spanner is roughly about 2. Something it between 2 and 3. So, you cannot. So, does not give you arbitrarily small dilations here, you can basically get as close to one as you want, but that is of course, you know a more simple geometric construction.

For this one you know now, we went through a series of basically steps before we write with this. So, we did have to work through your quadtrees, then from quadtrees we went to well separated decomposition with itself, I would say all the construction is very simple given the quadtree, but the motion of WSPD is not single you know, the definition that way it was described and the way we bounded the points into boxes etcetera you know they are far from preview and then form there it leaves to this and also from here this one you can also get something very easily, which is a an approximate minimum spanning tree. So, when you go to higher dimensions you know, it is a problem that people have worked on very hard that how do we construct the minimum spanning tree of a given set of points in high dimensions.

In 2 dimensions, we know that you know what this one also exercise problem that the delaunay triangulation is a super set of the minimum spanning tree. So, if you run your minimum spanning tree algorithm just on the delaunay triangle edges of a delaunay triangulation you get a minimum spanning tree, but you know how about higher dimensions. Now, constructing delaunay triangulations of a points, it is a d-dimension is prohibitively expensive, because you it everything scales up exponentially with d. The number of edges etcetera will be some n to the power d and so, that.

So, we are not going to try to construct the delaunay triangulation of points in higher dimensions. Why do we (( )). Let me say what is the real goal, the goal is that we do not want to spend n square time in constructing the n as d, n square is do something close to n square, because you are your implicit graph has n square edges, but we do not want to spend n square time in constructing, the m as tree of a set of points in high dimensions, because we want to somehow exploit the geometry otherwise, it becomes the same thing as any graph.

In a graph, if I have n square edges you know, we will take let say about n square log n time to construct it, but for point sets in d-dimensions, we want to exploit the geometry. So, that we can do it much faster than n squares o of n square. So, that is a goal and on that people have not really quite succeeded, yet you know there are some algorithms that give you marginal improvements over n square you know with some absurd dependencies on dimensions, again it only works when dimension is not that large.

In general we still do not have a construction; however, if you are willing to relax that minimum spanning tree to an let us, some make it 1 plus epsilon minimum spanning tree. So, my minimum spanning tree is not going to have weight more than 1 plus epsilon times, the actual minimum spanning tree. Like we are defined to the spanners, see in that case actually you can argue very easily that once you have a spanner the minimum spanning tree of the spanner is also a 1 plus epsilon minimum spanning tree. This sort of again falls directly from this and that has the construction time is kind of linear it is not n square. So, you n log n plus some sometimes. So, we stop it.