Well-Separated Pair Decomposition

Subhash Suri

November 7, 2019

1 The n-Body Problem

- We have seen several efficient algorithms for geometric problems so far. However, for many problems, either NP-hardness or "curse of dimensionality" rules out an efficient algorithm if we insist on an *optimal* solution. In those cases, a natural approach is to design *approximation* algorithms, which are guaranteed to run fast and produce solutions provably close to optimal.

- In this lecture, we discuss WSPD, a data structure for organizing multi-dimensional points with a *tunable scaling parameter* s, which allows us to solve many higher dimensional problems in almost linear time with approximation controlled by parameter s.

- We first motivate the general idea behind WSPD using an application in Astrophysics, called *n-body simulation.* The n-body simulation involves performing a computer simulation at a galactic scale, in which the motion of a huge (astronomical!) number of bodies (stars, planets, galaxies) is tracked under mutual gravitation forces.

- No exact analytic solution is known, and even good numerical simulations are extremely costly. For instance, in order to compute the motion of any single object, we need to know gravitational effects of each of the remaining $n-1$ objects. This leads to $\Omega(n)$ computation *per point*, and therefore $\Omega(n^2)$ computational for the whole system *per time step of the simulation!*

- This begs the natural computational question: *is there is a faster way to carry out the simulation?* A geometric structure, called WSPD, turns out to be the key, and it has found a huge number of applications in geometric approximation, beyond just the n-body simulation.

- Reference Callahan and Kosaraju papers.

- Specifically, what we want is a geometric structure that can encode the $\Theta(n^2)$ pairwise distances much more compactly, say, using only $O(n)$ size?
- This may seem like an impossible task, but if we are willing to settle for *approximation* of distances, then this can be done, and this was the basis for a famous technique called Fast Multipole Method (FMM) by Greengard and Rokhlin in mid 1980s. WSPD is a geometric structure for encoding distances in a way that makes FMM such a popular method.

Well Separated Pairs

- A set of n points defines a set of $\binom{n}{2} = \Theta(n^2)$ distinct pairs and distances. To motivate how to encode the distances approximately, let be return to the n-body problem.
- Suppose we want to compute the gravitational effect of a large number of stars in one galaxy on the stars of a distant galaxy. If the two galaxies are far enough from each other *relative* to their respective sizes, then the individual influences of the bodies in each galaxy can be *aggregated* into a single physical force.
- In other words, if there are n_1 and n_2 points in the two galaxies, then the interactions due to these $n_1 \times n_2$ pairs can be *approximated* by a single interaction pair, say, at the *centers* of their galaxies.
- To make this more precise, suppose we have an n-element point set $P \in R^d$, and a separation factor $s > 0$. (See Figure.) We say that two *disjoint* subsets $A, B \subseteq P$ are *s-well separated* if
  - each of A and B can be enclosed in Euclidean balls of radius r, and
  - the minimum distance between the two balls is $> sr$.
- Trivially, any singleton point lies inside a ball of radius 0, and so a pair of two points $\{\{a\}, \{b\}\}$, for $a \neq b$, is always well-separated for any $s > 0$.

Well Separated Pair Decomposition

- We want a general structure, which works for any set of points and not just for galaxies, which tend to naturally have clusters with large distances between them. Given two arbitrary sets of points A, B in R^d, define $A \otimes B$ to be the set of all distinct (unordered) pairs between them:

$$A \otimes B \quad = \quad \{\{a, b\} \mid a \in A, b \in B, a \neq b\}$$

- We note that $A \otimes A$ consists of all the $\binom{n}{2}$ distinct pairs of A.
- Given a point set P and a separation factor $s > 0$, we define the WSPD of P to be a collection of *pairs of subsets of* P, denoted $\{\{A_1, B_1\}, \ldots, \{A_m, B_m\}\}$ such that

  1. $A_i, B_i \subseteq P$, for all i
  2. $A_i \cap B_i = \emptyset$, for all i,
  3. $\bigcup_{i=1}^{m} A_i \otimes B_i \;=\; P \otimes P$, and
  4. A_i and B_i are s-separated for all i.

- Conditions 1–3 ensure that we cover all the unordered pairs of P, and 4 ensures that each pair of subsets is well separated.
- Although these conditions do not require that each unordered pair from P occurs in a unique pair $A_i \otimes B_i$, our construction will have this property.
- Trivially, there exists a WSPD of size $O(n^2)$ by setting $\{A_i, B_i\}$ pairs to each distinct pair of points.
- Our goal however is to show that *for any constant s and any point set P, we can construct a WSPD of size $O(n)$*—the constant depends on s, d and has the form s^d.
- **Quadtrees.** A quadtree is a hierarchical subdivision of space into regions, called *cells,* which are hypercubes. Initially, we have a single large hypercube containing all of P, and for simplicity (by scaling) we assume that this is the unit hypercube $[0, 1]^d$, which corresponds to the *root* of the tree.
- The quadtree is then recursively build as follows: consider a cell and its associated node u.

  1. if this cell contains 0 or one point of P, then we declare it a *leaf node* and terminate the recursive call.
  2. Otherwise the cell is subdivided into 2^d hypercubes, whose side lengths are exactly half of the parent's side length. For each of these 2^d cells, we create a node and make it a child of u in the quadtree.
  3. An example in 2D is shown below. We label the 4 children in the SW, NW, SE, NE (left to right) order. Equivalently, first we make a east-west binary cut, and then cut each of them into top and bottom halves.

- In practice, quadtrees as described above tend to be quite efficient, *however*, there are a number of important technical issues if we want to ensure *worst-case* guarantees.

1. The first problem is that the quadtree we just described may have many more than $O(n)$ nodes! The reason is that a group of points that are extremely close to each other relative to their surroundings may need an unbounded number of subdivisions, leading to arbitrarily long trivial paths in the tree, where only one of the 2^d cells is an internal node.
2. This technical problem is easily remedied by a process called *path compression*. Any such trivial path can be replaced by a single edge, which is labeled with coordinates of the smallest quadtree box containing the cluster.
3. Each internal node of the resulting compressed quadtree separates at least two points into separate subtrees, and so there are at most $n - 1$ internal nodes and $O(n)$ nodes overall.
4. The second problem is that even the compressed quadtree can have height $h = \Theta(n)$, and so the straightforward algorithm for computing it may take $O(nh) = O(n^2)$ time. This problem is handled by using *fair splits*, and as a result the quadtree can be built in $O(n \log n)$ time for any constant dim d. (Read the paper for details.)

- We can, therefore, summarize the key facts about quadtrees, which will be used in our WSPD algorithm.

1. Given a set of n points in any fixed dimension d, a compressed quadtree can be computed in $O(n \log n)$ time.
2. Each internal mode has a constant number (2^d) of children.
3. The cell associated with each node of the quadtree is a d-dim hypercube. The size (side length) of a child (in the uncompressed tree) is half of its parent.
4. The cells associated with any level of the tree (in the uncompressed tree) are all of the same size, and have pairwise disjoint interiors.

- *While for efficient computation, we will use the compressed quadtree but for reasoning about its geometric properties and establishing complexity bounds, it will be more convenient to use the* uncompressed quadtree.

Packing Lemma and Constructing WSPD

- An important consequence of the properties 3–4 is the following Lemma.
- **Packing Lemma.** Let b be a ball of radius r in dim d, and let X be any collection of pairwise disjoint quadtree cells, each of side length $\geq x$, that overlap b. Then, we have

$$\|X\| \quad \leq \quad \left(1 + \lceil \frac{2r}{x} \rceil\right)^d \quad \leq \quad O\left(\max\left(2, \frac{r}{x}\right)^d\right)$$

- **Proof.** We may assume that all cells of X have the same side length, x, since making them larger only reduces the number of non-overlapping cells.

  1. The cells of size x form a grid G in the quadtree.
  2. If H is a hypercube of side length $2r$ enclosing b, then every cell of X overlaps H.
  3. Along each dimension, the number of cells of G that can overlap an interval of side length $2r$ is $t \leq 1 + \lceil 2r/x \rceil$.
  4. Thus, the number of cubes of G that overlap H is at most t^d. If $2r < x$, this quantity is $\leq 2^d$, and otherwise $O((r/x)^d)$.

- The main idea behind WSPD construction algorithm is the following: we will adaptively refine the quadtree, starting with the initial hypercube. At each step, we check for pairs of cells that satisfy the well-separated condition, and output any we find, and stop their refinement. For those not well-separated, we continue recursive refinement.

- The adaptive refinement however needs to be performed carefully to guarantee that we cover all pairs in just $O(n)$ cell-pairs.

- Let us first introduce a few technical definitions.

- **Representative.** First, for each node u of the quadtree (both leaves and internal nodes), we declare one of the points in its cell a *representative*, denoted $\text{rep}(u)$. We do this recursively, as follows:

  - If u is a leaf containing the point p, then $\text{rep}(u) = \{p\}$. If u is a leaf without a point, then $\text{rep}(u) = \emptyset$.
  - Otherwise, if u is an internal node, then it must have at least one child v that is not an empty leaf. (If there are multiple nonempty children, select any one.) Set $\text{rep}(u) = \text{rep}(v)$.

- Given a node u, let P_u denote the set of points that lie within the subtree rooted at u. We now define the levels for all the nodes of the tree.

- Assuming that original point set lies inside a unit hypercube, the side lengths of the cells are of the form $1/2^i$.

- **Levels.** We define the level of node u to be $level(u) = -\log_2 x$, where x is the side length of u's cell. *That is, level of u is just its depth in the uncompressed tree, where root has depth 0.*

- The key point is that $level(u) \leq level(v)$ if and only if side length of u's cell is $\geq$ side length of v's cell.

- We treat leaves differently from internal nodes. If a leaf contains no point, then we ignore it, since it cannot participate in any well-separated pair. If it does contain a point, then we think of the leaf node conceptually as an infinitesimally small cell containing this point. We do this by defining $level(u) = +\infty$. We will see later why this is useful.

- **Constructing WSPD.** We show that for any point set P in d-space, and any $s > 0$, there exists a WSPD of size $O(s^d n)$, which can be computed in time $O(n \log n + s^d n)$.

- Our construction will be recursive. We maintain a collection of sets that always satisfy properties (1) and (3) but may violate (2) and (4)—that is, the sets may not be disjoint and may not be well-separated. When the algorithm terminates, all pairs will be well-separated, which will also ensure disjointness.

- Each $\{A_i, B_i\}$ in our WSPD will be encoded as a pair of nodes $\{u, v\}$ in the quadtree. Implicitly, this pair represents the pair $P_u \otimes P_v$, namely, the set of cross-product of all descendants of u with all descendants of v. *This implies that the total storage is proportional to the number of pairs in the decomposition.*

- The algorithm $WSPD(u, v, s)$ is based on a recursive subdivision, and can be described as follows.

WSPD Algorithm

1. If u and v are leaves and $u = v$, return.
2. If rep(u) or rep(v) is empty, return $\emptyset$
   else if u and v are s-separated *return* $\{\{u, v\}\}$.
3. else
   - if $level(u) > level(v)$, swap u and v
   - Let $u_1, u_2, \ldots, u_m$ denote the children of u
   - return $\bigcup_{i=1}^{m} WSPD(u_i, v, s)$

- The initial call is made $WSPD(z, z, s)$, where z is the root node.

- **Remark.** Due to symmetry, the procedure will generally produce duplicate pairs $\{P_u, P_v\}$, and $\{P_v, P_u\}$. Use any disambiguation rule to eliminate duplicates.
- Explanation for the algorithm.
  1. If either node is an *empty* leaf, then we can ignore this pair–nothing to output.
  2. Otherwise, let u, v be the pair under consideration. Consider the two smallest Euclidean balls of equal radius that enclose u and v cells. If these balls are well separated, then we report $\{u, v\}$ as a WSPD pair.
  3. Otherwise, assume that u's cell is larger, i.e., $level(u) \leq level(v)$. We subdivide u by considering its children, and calling WSPD for each pair (u_i, v).

WSPD Analysis

- By construction, the algorithm only terminates when all pairs are well separated. So, the main problem is to analyze the number of pairs are generated by the algorithm.
- We will simplify our proof by assuming that the quadtree is *not compressed* yet has size $O(n)$. This allows us to assume that children of each node have size exactly half of the parent's. The proof for the general case follows the same steps with slightly more involved analysis.
- Our first observation is that, with the assumption of non-compressed quadtree, when a call $WSPD(u, v, s)$ is made, the cells of u and v differ in size by at most factor 2, *since the algorithm always splits the larger of the two cells.*
- We will also assume $s \geq 1$, to simplify our analysis.
- **Terminal and non-Terminal calls.** In order to analyze $WSPD$, we will count recursive calls to the procedure $WSPD$. A call is terminal if it does not reach the final *else* (Step 3), and otherwise is a non-terminal call.
- Each terminal call outputs at most one WSPD pair, so we just need to count the number of terminal calls to bound the output pairs.
- But terminal calls are generated by non-terminal calls, and each non-terminal call generates at most 2^d recursive calls, so the total number of *WSPD pairs is at most* 2^d *times the number of non-terminal calls.*
- In order to bound the number of non-terminal calls, we will apply a *charging scheme*, where each such call is *paid by* some quadtree node. Our proof will show that a node v of the quadtree pays for only $O(s^d)$ calls, and since there are $O(n)$ nodes, the total number of non-terminal calls is $O(s^d n)$.

- **Charging Scheme.** Whenever the final else statement is reached in the algorithm, and we split the cell u, the non-split node v *pays for it.* That is, we charge the *smaller* node for the non-terminal call which splits the larger node u. So, the question is how many times a node v can be charged?

- The node v pays for a call only if the call $WSPD(u, v, s)$ was non-terminal, which means that u and v are not (well) separated.

- Let x be side length of v's cell, and $r_v = x\sqrt{d}/2$ the radius of its enclosing ball.

- We know that u's cell is $\geq v$'s cell, but u's side length is $\leq$ $2\times$ v's side length. Thus, u's cell has side length either x or $2x$, and therefore u's enclosing ball has radius $r_u \leq 2r_v$.

- Since u and v are not separated, the distance between their balls is at most

$$s \times \max(r_u, r_v) \quad \leq \quad 2sr_v \quad = \quad sx\sqrt{d}$$

- Thus, the center of their enclosing balls are within distance

$$R_v \quad = \quad r_v + r_u + sx\sqrt{d} \quad \leq \quad \left(\frac{1}{2} + 1 + s\right) x\sqrt{d} \;\leq\; 3sx\sqrt{d}$$

  because $s \geq 1$.

- Let b_v be a ball of radius R_v centered at v's cell.

- The set of quadtree cells u that can make v pay for a non-terminal call (1) each must have size either x or $2x$, and (2) must overlap b_v. Finally, by construction of quadtree, all cells of side length x are disjoint, as are the cells of side length $2x$. Therefore, using the Packing Lemma, the number of nodes that can force v to pay us at most

$$\left(1 + \lceil \frac{2R_v}{x} \rceil\right)^d + \left(1 + \lceil \frac{2R_v}{2x} \rceil\right)^d \quad = \quad O(s^d)$$

- Putting it all together, we have $O(n)$ nodes in the quadtree, and each node pays for at most $O(s^d)$ non-terminal calls, and each such call can produce at most 2^d terminal calls and WSPD pairs. This proves the result.

- **WSPD Theorem.** Given a set of n points in d dimensions, and a fixed separation factor $s \geq 1$, we can build a WSPD of size $O(s^d n)$ in time $O(n \log n + s^d)$.

2 Applications of WSPD

- WSPD constructs a compact $O(n)$ size encoding of all pairwise distances (approximately) for any set of points in d dimensions, which is useful for solving many geometric optimization problems approximately in *nearly linear time in small dimensions*.

- We will discuss several such applications, including *diameter, closest pair, spanner graphs, MST* of d dimensional data sets. The following technical lemma will be useful.

- **WSPD Utility Lemma:** Suppose P is set of n points in d dimensions, and we have a well-separated pair decomposition of P with separation factor $s \geq 1$. If $\{P_u, P_v\}$ is a well-separated pair, and $x, x' \in P_u$ and $y, y' \in P_v$. Then, the following holds:

  1. $\|xx'\| \leq \frac{2}{s}\|xy\|$
  2. $\|x'y'\| \leq (1 + \frac{4}{s})\|xy\|$

- **Proof.** Intuitively, two points in the same set, P_u or P_v, are quite close compared to two points in different sets, and distances between points in the cross product $P_u \otimes P_v$ are very similar.

  1. The pair $\{P_u, P_v\}$ is s-separated, and so the sets can be enclosed in balls of radius r such that the min separation between balls is $\geq sr$. Therefore, $\max(\|xx'\|, \|yy'\|) \leq 2r$, while any pair in $\{x, x'\} \times \{y, y'\}$ is separated by distance at least sr. Therefore, we have

$$\|xx'\| \leq 2r = \frac{2r}{sr} sr \leq \frac{2r}{sr}\|xy\| = \frac{2}{s}\|xy\|$$

  2. By triangle inequality and the fact that $2r \leq (2/s)\|xy\|$, we have

$$\|x'y'\| \leq \|x'x\| + \|xy\| + \|yy'\| \leq 2r + \|xy\| + 2r \leq \left(1 + \frac{4}{s}\right)\|xy\|$$

2.1 Approximating the Diameter

- The *diameter* of a set of n points is the maximum distance between any two points. It can be computed by brute force in $O(n^2)$ time. In two dimensions, it is possible to compute the diameter in $O(n \log n)$ time, but in higher dimensions computing the diameter is not easy.

- We show how to estimate the diameter within a factor $(1 + \varepsilon)$ in linear time once the WSPD has been computed.

- Set $s = \frac{4}{\varepsilon}$, and compute the WSPD of P.
- For each pair $\{P_u, P_v\}$ of WSPD, let $p_u = \text{rep}(u)$ and $p_v = \text{rep}(v)$ denote the representative points.
- For each WSPD pair, of which there are $O(n)$, compute the distance $\|p_u p_v\|$ between their representatives, and output the pair with the largest such distance.
- To prove the correctness, suppose x, y are the points of P that realize the diameter, and let $\{P_u, P_v\}$ be the WSPD pair containing these points, with p_u, p_v being their representatives.
- By the WSPD Utility Lemma, we have

$$\|xy\| \;\leq\; \left(1 + \frac{4}{s}\right)\|p_u p_v\| \;=\; (1+\varepsilon)\|p_u p_v\|$$

- Since $\{x, y\}$ is the diameter-forming pair, we have

$$\frac{\|xy\|}{1+\varepsilon} \;\leq\; \|p_u p_v\| \;\leq\; \|xy\|$$

  which implies that $\|p_u p_v\|$ is an ε-approximation of the diameter.
- Once WSPD has been computed, the running time of the algorithm is $O(s^d n) = O(n/\varepsilon^d)$, which is $O(n)$ for any constant value of ε.

2.2 Approximating the Closest Pair

- The closest pair of points in P is the pair with the minimum distance. Using the same idea as used in the diameter algorithm, we can also estimate the closest pair distance of P: *simply report the WSPD pair with the minimum distance between its representatives.*
- Surprisingly, though, we will show that this algorithm actually returns the *exact closest pair*, and not just an approximation!!
- Suppose the pair $\{x, y\}$ is the closest pair of P, and let p_u, p_v be the representatives of the associated WSPD pair.
- If $x = p_u$ and $y = p_v$, then obviously we have the closest pair. Therefore, assume that either $x \neq p_u$ or $y \neq p_v$.

- But in that case, wouldn't it mean that either $\|xp_u\| \leq xy\|$ or $\|yp_v\| \leq \|xy\|$? (which would contradict x, y being closest pair)
- Formally, let us assume that $s > 2$. Since P_u, P_v lie within balls of radius r that are separated by distance at least $sr > 2r$. Thus, if $p_u \neq x$, we have our contradiction because
$$\|p_u x\| \leq 2r < sr \leq \|xy\|$$
- Therefore, using WSPD we can find the closest pair exactly in $O(n)$ time in any constant dimension!

2.3 Spanner Graphs in d Dimensions

- A set of n points in d dim defines a complete weighted graph, called Euclidean graph, in which each point is a vertex and each pair of vertices has an edge with weight equal to the Euclidean distance between those points.
- An Euclidean graph is dense with $\Theta(n^2)$ edges, and a spanner is a sparse approximation of this complete graph.
- In particular, given a parameter $t \geq 1$ (called the *stretch factor*), a t-spanner is a weighted graph G with vertices as points of P for which the follow holds for all pairs $x, y \in P$:
$$\|xy\| \leq d_G(x, y) \leq t \cdot \|xy\|$$
where $\|xy\|$ denotes the Euclidean distance, and d_G represents the graph distance.
- Do sparse spanner exist? We have seen that in two dimensions, DT is a sparse spanner with $t \leq 2.418$. But DT are unhelpful for sparse spanners in higher dimensions because DT in 3 or more dimensions can have $\Theta(n^2))$ edges.
- There are many approaches for constructing sparse spanner, and here we show one using WSPD.
- **WSPD-based Spanners.** Given a set of points P and stretch factor $t > 1$, we construct WSPD with parameter $s = 4(t + 1)/(t - 1)$.
- For each well-separated pair $\{P_u, P_v\}$ in WSPD, let $p_u = \text{rep}(u)$ and $p_v = \text{rep}(v)$. Add the edge $\{p_u, p_v\}$ to our graph.
- The number of edges in this graph is equal to the number of well-separated pairs, which is $O(s^d n)$, which is $O(n)$ for constant t.

Analysis of the WSPD Spanner

- Clearly, for any $x, y \in P$, $d_G(x, y) \geq \|xy\|$, so we just need to show that $d_G(x, y) \leq t \cdot \|xy\|$. We will prove this by induction on the number of edges in the shortest path from x to y.
- The base case is when x, y are joined by an edge in G, in which case clearly

$$d_G(x, y) = \|xy\| \leq t \cdot \|xy\|, \quad \text{for all} t \geq 1$$

- Otherwise, the x, y lies in some WSPD pair $\{P_u, P_v\}$ defined by two nodes u and v, with $p_u = \text{rep}(u)$ and $p_v = \text{rep}(v)$. (It may be that $x = p_u$ or $y = p_v$, but not both.)
- Let us consider the length of the shortest path x–p_u–p_v–y, and use the fact that edge $\{p_u, p_v\}$ is in G.

$$d_G(x, y) \leq d_G(x, p_u) + \|p_u p_v\| + d_G(p_v, y)$$

- By induction, this implies

$$d_G(x, y) \;\leq\; t\,(\|xp_u\| \;+\; \|p_v y\|) \;+\; \|p_u p_v\|$$

- By the WSPD Utility Lemma, we get

$$\max(\|xp_u\|, \|p_v y\|) \;\leq\; \frac{2}{s}\|xy\| \qquad \text{and} \qquad \|p_u p_v\| \;\leq\; \left(1 + \frac{4}{s}\right)\|xy\|$$

- Combining these bounds, we get

$$d_G(x, y) \;\leq\; t\left(2\frac{2}{s}\|xy\|\right) + \left(1 + \frac{4}{s}\right)\|xy\| \;=\; \left(1 + \frac{4(t+1)}{s}\right)\|xy\|$$

- We now just need to choose s so that $1 + 4(t+1)/s \leq t$, which can be done by choosing $s = 4\left(\frac{t+1}{t-1}\right)$. For this choice of s, we get

$$d_G(x, y) \;\leq\; t\|xy\|$$

- Since spanner are most useful for small stretch factors, let assume assume $t \leq 2$, and write it as $t = 1 + \varepsilon$, for some $\varepsilon \leq 1$. In that case, the size of the spanner graph is

$$O(s^d n) \;=\; O\left(\left(4\frac{(1+\varepsilon)+1}{(1+\varepsilon)-1}\right)^d n\right) \;\leq\; O(n/\varepsilon^d)$$

- **Spanner Theorem.** Given a set P of n points in d dimensions, and $\varepsilon > 0$, we can construct a $(1+\varepsilon)$ spanner for P with $O(n/\varepsilon^d)$ edges in time $O(n \log n + n/\varepsilon^d)$.
- **MST Theorem.** Given a set P of n points in d dimensions, and $\varepsilon > 0$, we can construct spanning tree of P whose weight is within a $(1+\varepsilon)$ of the MST of P in time $O(n \log n + n/\varepsilon^d)$.