Well-Separated Pair Decomposition in Linear Time?

Timothy M. Chan*

October 16, 2007

Abstract

Given a point set in a fixed dimension, we note that a well-separated pair decomposition can be found in linear time if we assume that the ratio of the farthest pair distance to the closest pair distance is polynomially bounded. Many consequences follow; for example, we can construct spanners or solve the all-nearest-neighbors problem in linear time (under the same assumption), and we compute an approximate Euclidean minimum spanning tree in linear time (without any assumption).

Keywords: Computational geometry; Approximation algorithms; Quadtrees

Techniques from computational geometry have led to efficient approximation algorithms for many proximity-related problems on n-point sets in low-dimensional Euclidean spaces [17]. For example, we can answer approximate nearest neighbor queries in logarithmic time after $O(n \log n)$-time preprocessing [2]; we can construct a spanner with $O(n)$ edges with $1+\varepsilon$ stretch factor in $O(n \log n)$ time [20]; we can construct a $(1+\varepsilon)$-factor approximate Euclidean minimum spanning tree (EMST) in $O(n \log n)$ time [19].

In this note, we observe that many of these $O(n \log n)$ algorithms can be sped up to run in *linear* time under a fairly reasonable assumption—namely, that the *spread*, defined as the ratio of the largest pairwise distance to the smallest pairwise distance, is bounded by a polynomial n^c for a fixed constant c.

Specifically, we show that a *well-separated pair decomposition* (WSPD) [5]—a well-known tool in the area (e.g., see various books and surveys [15, 17, 18] or below for the precise definition)—can be constructed in linear time under this assumption. Immediately, this implies linear-time algorithms for spanners, approximate EMST, and other problems (e.g., the exact all-nearest-neighbors problem). For the EMST case, we can in fact eliminate the bounded-spread assumption.

The model we use is one that computational geometers are most comfortable with—the real-RAM model. We only assume that the floor function is available and that the word size is at least $\log n$. (If instead we assume that coordinates are integers and adopt a transdictomous word RAM—a model that has gained attention in recent years [8]—we can eliminate the bounded-spread assumption and still obtain $o(n \log n)$ algorithms for WSPD, with running time matching the best integer-sorting results [1, 12, 13].)

The techniques we use merely involve combining known algorithms with the "shuffle-and-sort" idea from the author's previous papers [6, 7] (which addressed the static and dynamic closest pair

*School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@uwaterloo.ca).
This work was supported in part by NSERC.

problems and approximate nearest neighbor search). Although none of the individual steps are original, the end results have not been noticed before; for example, our linear-time algorithm for approximate EMST is a strict improvement over a previous $O(n \log\log n)$-time approximation algorithm by Bern et al. [3], which was applicable only for the 2-d case. Writing this brief note thus seems justified. The presentation below, though concise, will be largely self-contained.

Step 0: Rounding to a grid

Let P_0 be the given set of n points in $\mathbb{R}^d$, where d is treated as a constant. Let ε be any sufficiently small parameter exceeding $1/n^{\Omega(1)}$. Let D be the distance of an arbitrary point in P_0 to its farthest neighbor (computable in linear time). Note that the diameter of the point set (the farthest pair distance) is at most $2D$.

Consider a uniform grid with side length $2\varepsilon D/n^c$. Round each point to its nearest grid point, and let P be the resulting set of grid points (computable in linear time using the floor function). By scaling, we may assume that the coordinates of P are all integers in the range $[0, 2^w)$ where $w = \lceil \log(n^c/\varepsilon) \rceil = O(\log n)$.

Step 1: Sorting in shuffle order

Given a point p with coordinates $(p_{1w} \cdots p_{11}, p_{2w} \cdots p_{21}, \ldots, p_{dw} \cdots p_{d1})$ written in binary, the *shuffle* of p is defined to be the number $p_{1w}p_{2w} \cdots p_{dw} \cdots p_{11}p_{21} \cdots p_{d1}$ written in binary.

The shuffle of a point can be computed in constant time, since $w = O(\log n)$: We can first build a table storing the shuffles for all possible d-tuples of $(\log n)/b$-bit coordinates; the table can be initialized in $o(n)$ time if we choose a constant $b > d$. To compute the shuffle of a point with $O(\log n)$-bit coordinates, we break each coordinate into $O(1)$ subwords each of length $(\log n)/b$ (using shifts, implementable by the floor function), and then perform $O(1)$ shuffles on these subwords by table lookup and concatenate the results. Each shuffle can be stored in $O(1)$ words.

As preprocessing, we sort the points $p_1, \ldots, p_n \in P$ in increasing order of their shuffle values (the *shuffle order*). This step takes $O(n)$ time, since for a set of n $O(\log n)$-bit integers, radix sort with $O(1)$ rounds runs in linear time.

Remark: if w were superlogarithmic, we can still get $o(n \log n)$ running time by applying known integer-sorting algorithms, on a word RAM model where the shuffle operation can be done in constant time.

Step 2: Computing a compressed quadtree

Define a hierarchy of *quadtree boxes*[1] as follows: the hypercube $[0, 2^w)^d$ is a quadtree box at level 0; for each quadtree box B at level i, form two quadtree boxes at level $i+1$ by dividing B evenly via a hyperplane orthogonal to the $((i \bmod d)+1)$-th axis. Of the two subboxes of B, the one with smaller (resp. larger) $((i \bmod d)+1)$-th coordinate is the *left* (resp. *right*) subbox. Note that all quadtree boxes at the same level form a grid, with aspect ratio at most 2. We use $|B|$ to denote the diameter of a box B (which is solely a function of the level).

It is not difficult to see that quadtree boxes and shuffles are related: all points in the left subbox of a quadtree box B have smaller shuffle values than all points in the right subbox.

[1] The name arose from the special case $d = 2$. Several variants of the definition exist; we use a binary version here where the degree of the quadtree is 2 instead of 2^d.

The *compressed quadtree* T for a point set P is defined as follows: if P has only one point, then T is just a leaf holding this point; otherwise, T consists of a root holding the smallest quadtree box B enclosing P, and two subtrees recursively built for the subset of points in the left subbox of B and the subset of points in the right subbox of B. Note that T is a binary tree with $O(n)$ nodes (and $O(w)$ height).

The definition above does not immediately suggest a linear-time algorithm, but we can use the following equivalent reformulation, based on the observation that the left-to-right order of the leaves in T coincides precisely with the shuffle order $p_1, \ldots, p_n$. Let $\mathrm{BOX}(p,q)$ denote the smallest quadtree box containing p and q. Consider the index j such that $|\mathrm{BOX}(p_{j-1}, p_j)|$ is the largest. Then the compressed quadtree T for $\{p_1, \ldots, p_n\}$ simply consists of a root holding $\mathrm{BOX}(p_{j-1}, p_j)$ and two subtrees recursively built for $\{p_1, \ldots, p_{j-1}\}$ and for $\{p_j, \ldots, p_n\}$.

This re-definition actually reduces to a known construct (specifically the "Cartesian tree" of the sequence $|\mathrm{BOX}(p_1,p_2)|, |\mathrm{BOX}(p_2,p_3)|, \ldots, |\mathrm{BOX}(p_{n-1},p_n)|$), for which there is a standard incremental algorithm [10]. We quickly re-describe this algorithm for the sake of completeness (it bears some resemblance to Graham's scan [11]). We maintain the rightmost root-to-leaf path $q_1, \ldots, q_k$ of the tree as points are inserted in shuffle order. As the next point p_i arrives, we insert a new node for $\mathrm{BOX}(p_{i-1}, p_i)$ in an appropriate place along this path, then update the path by removing a suffix and appending the new node. In the pseudocode below, q.box, q.left, and q.right denote the box, left child, and right child of a node q.

0. $q_0.\text{box} = \mathbb{R}^d$, $q_0.\text{right} = p_1$, $k = 0$
1. for $i = 2, \ldots, n$ do
2.   while $|\mathrm{BOX}(p_{i-1}, p_i)| > |q_k.\text{box}|$ do $k = k - 1$
3.   create a node q_{k+1} with $q_{k+1}.\text{box} = \mathrm{BOX}(p_{i-1}, p_i)$,
        $q_{k+1}.\text{left} = q_k.\text{right}$, $q_{k+1}.\text{right} = p_i$
4. $q_k.\text{right} = q_{k+1}$, $k = k + 1$

The test in line 2 takes constant time: we can deduce the level of $\mathrm{BOX}(p,q)$ from the most significant bit position in which the shuffle of p and the shuffle of q differ. The most-significant-bit operation can be implemented in $O(1)$ time, since $w = O(\log n)$, by using table look-up as before if necessary.

The running time to compute the compressed quadtree is $O(n)$ by a simple amortization argument: the total cost of line 2 is proportional to the total number of decrements of k, which is bounded by the total number of increments of k, which is clearly at most n.

Step 3: Computing a WSPD

Two sets A and B are said to be *ε-well-separated* if the diameter of A and the diameter of B are both at most ε times the minimum distance between A and B. Notice that distances between pairs of points from $A \times B$ are all identical to within a factor of $1 + O(\varepsilon)$.

An *ε-well-separated pair decomposition (ε-WSPD) of size* m for a point set P is a collection of ε-well-separated pairs of subsets $\{(P_1, Q_1), \ldots, (P_m, Q_m)\}$, where $P_i, Q_i \subseteq P$, such that every pair of points $(p,q) \in P \times P$ $(p \neq q)$ lies in $P_i \times Q_i$ or $Q_i \times P_i$ for exactly one index i. The usefulness of the WSPD can be seen as it allows all pairwise distances to be compactly summarized by m distances. Note that the size of the WSPD is defined as the number of subset pairs m, not the total sizes of the subsets; in constructing WSPDs, the subsets P_i and Q_i may be represented implicitly.

Given a compressed quadtree T, we can compute a WSPD of linear size by the following simple recursive algorithm, which is essentially taken from Callahan and Kosaraju's original paper introducing WSPDs [5]. We quickly include both the pseudocode and analysis here for the sake of completeness. Below, $P[q]$ denotes the subset of points underneath the node q. We initially call $\textsc{wspd}(q)$ with q being the root.

$\textsc{wspd}(q)$:

0. if q is leaf then return $\emptyset$
1. return $\textsc{wspd}(q.\text{left}) \cup \textsc{wspd}(q.\text{right}) \cup \textsc{wspd}(q.\text{left}, q.\text{right})$

$\textsc{wspd}(q_1, q_2)$:

2. if q_1.box and q_2.box are ε-well-separated then return $\{(P[q_1], P[q_2])\}$
3. else if $|q_1.\text{box}| \geq |q_2.\text{box}|$ then return $\textsc{wspd}(q_1.\text{left}, q_2) \cup \textsc{wspd}(q_1.\text{right}, q_2)$
4. else return $\textsc{wspd}(q_1, q_2.\text{left}) \cup \textsc{wspd}(q_1, q_2.\text{right})$

The algorithm clearly outputs a WSPD. To analyze its size and the running time, observe that if $\textsc{wspd}(q_1, q_2)$ is called, then $|q_1.\text{par.box}| \geq |q_2.\text{box}|$ and $|q_2.\text{par.box}| \geq |q_1.\text{box}|$, where q.par denotes the parent of a node q: this follows because of the test made in line 3 (and induction).

The total running time is proportional to the total number of calls to $\textsc{wspd}(q_1, q_2)$ such that q_1.box and q_2.box are not ε-well-separated (as otherwise recursion is terminated by line 2). W.l.o.g., say $|q_1.\text{box}| \geq |q_2.\text{box}|$. Since in addition $|q_2.\text{par.box}| \geq |q_1.\text{box}|$, we can find a quadtree box B at the same level as q_1.box with $q_2.\text{par.box} \supseteq B \supseteq q_2.\text{box}$. Since q_1.box and B are not well-separated, B must be within distance $O(|B|/\varepsilon)$ from q_1. For each fixed node q_1, there are at most $O(1/\varepsilon^d)$ such quadtree boxes B (since these boxes have bounded aspect ratio and form a grid). For each fixed B, there are $O(1)$ candidates for q_2 in the tree. It follows that the total number of candidates for (q_1, q_2) is $O(n/\varepsilon^d)$. The running time, and hence the size of the WSPD, are $O(n/\varepsilon^d)$.

Finally, note that if P_0 indeed has spread at most n^c, then an ε-WSPD for P maps to an $O(\varepsilon)$-WSPD for P_0, since the rounding step changes the distance of any pair (p, q) by an additive amount of $O(\varepsilon D/n^c)$, which is at most $O(\varepsilon)$ times the actual distance. We conclude that an $O(\varepsilon)$-WSPD of $O(n/\varepsilon^d)$ size for any point set with polynomially bounded spread can be constructed in $O(n/\varepsilon^d)$ time.

Step 4: Computing a spanner

A *δ-spanner* for a point set P is a subgraph G of the complete undirected graph on the vertex set P such that every pair of points $(p, q) \in P \times P$ satisfies $d_G(p, q) \leq (1 + \delta)d(p, q)$ where $d_G(\cdot, \cdot)$ and $d(\cdot, \cdot)$ denote the shortest path metric for G and the Euclidean metric respectively.

As observed by Callahan and Kosaraju [4], we can easily construct an $O(\varepsilon)$-spanner with m edges given an ε-WSPD $\{(P_i, Q_i)\}_i$ of size $m = O(n/\varepsilon^d)$: just pick an arbitrary edge (p_i, q_i) from $P_i \times Q_i$ for each i. The total running time is $O(n)$ for any point set with polynomially bounded spread.

To see why this yields a spanner, take any pair of points (p, q), say, with $p \in P_i$ and $q \in Q_i$. Since (P_i, Q_i) is ε-well-separated, $d(p, p_i), d(q_i, q) \leq \varepsilon d(p, q)$ and $d(p_i, q_i) \leq (1 + 2\varepsilon)d(p, q)$. Assume that $d_G(p, p_i) \leq (1+\delta)d(p, p_i)$ and $d_G(q_i, q) \leq (1+\delta)d(q_i, q)$ by induction (in order of increasing distances). It follows that $d_G(p, q) \leq d_G(p, p_i) + d(p_i, q_i) + d_G(q, q_i) \leq (1 + 2\varepsilon + 2\varepsilon(1 + \delta))d(p, q) \leq (1 + \delta)d(p, q)$, by setting $\delta = 4\varepsilon/(1 - 2\varepsilon) = \Theta(\varepsilon)$.

Step 5: Computing an approximate EMST

We can compute a $(1 + O(\varepsilon))$-factor Euclidean minimum spanning tree (EMST) for the point set P, simply by constructing an $O(\varepsilon)$-spanner G with $m = O(n/\varepsilon^d)$ edges and returning the minimum spanning tree (MST) of G. The last step takes $O(n + m)$ time by Karger, Klein, and Tarjan's randomized MST algorithm [14], or by Fredman and Willard's deterministic MST algorithm [9]. Note that Fredman and Willard's "transdichotomous" algorithm is applicable here, since the coordinates in P are $O(\log n)$-bit integers, and so are the edge weights after squaring (squaring does not affect the ordering of the weights and hence does not affect the MST).

For the EMST problem, we can actually assume that the spread of P_0 is bounded by $O(n)$, by rounding to a uniform grid with side length $\Theta(\varepsilon D/n)$, since the weight of any spanning tree changes by an additive term of only $O(\varepsilon D)$, which is clearly at most $O(\varepsilon)$ times the EMST weight. We conclude that a $(1 + O(\varepsilon))$-factor approximate EMST can be constructed in $O(n/\varepsilon^d)$ time for *any* point set.

Remark: One implication is that a factor-$(2+\varepsilon)$ approximation for the *Euclidean traveling salesman* problem can be computed in linear time in any constant dimension. A linear-time PTAS might also be possible, but a close examination of Rao and Smith's algorithm [16] would be required.

Remark: For another application, Callahan and Kosaraju [5] have shown that given an ε-WSPD of size $O(n)$ whose subsets are given hierarchically (as in the preceding construction) for a sufficiently small constant $\varepsilon > 0$, we can solve the exact *all-k-nearest-neighbors* problem—finding the k nearest neighbors in P for every point in P—in $O(nk)$ time. In particular, we can thus solve the all-nearest-neighbors problem ($k = 1$) in linear time for any point set with polynomially bounded spread. This extends a previous observation from [6] that the closest pair problem can be solved in linear time deterministically for any point set with polynomially bounded integer coordinates.

References

[1] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *J. Comput. Sys. Sci.*, 57:74–93, 1998.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, 1998.

[3] M. W. Bern, H. J. Karloff, P. Raghavan, and B. Schieber. Fast geometric approximation techniques and geometric embedding problems. *Theoret. Comput. Sci.*, 106:265–281, 1992.

[4] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 291–300, 1993.

[5] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[6] T. M. Chan. Closest-point problems simplified on the RAM. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 472–473, 2002.

[7] T. M. Chan. A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions. Manuscript, 2006.

[8] T. M. Chan and M. Pătraşcu. Point location in sublogarithmic time and other transdichotomous results in computational geometry. Submitted to *SIAM J. Comput.*. Preliminary versions in *Proc. 47th IEEE Sympos. Found. Comput. Sci.*, pages 325–332 and 333–342, 2006.

[9] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Sys. Sci.*, 47:424–436, 1993.

[10] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th ACM Sympos. Theory Comput.*, pages 135–143, 1984.

[11] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[12] Y. Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms*, 50:96–105, 2004.

[13] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 135–144, 2002.

[14] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42:321–328, 1995.

[15] G. Narasimhan and M. Smid. *Geometric Spanner Networks.* Cambridge University Press, 2007.

[16] S. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th ACM Sympos. Theory Comput.*, pages 540–550, 1998.

[17] M. Smid. Closest-point problems in computational geometry. In *Handbook of Computational Geometry* (J. Urrutia and J. Sack, ed.), North-Holland, pages 877–935, 2000.

[18] M. Smid. The well-separated pair decomposition and its applications. In *Handbook of Approximation Algorithms and Metaheuristics* (T. Gonzalez ed.), Chapman & Hall/CRC, Boca Raton, pages 53-1–53-2, 2007.

[19] P. M. Vaidya. Minimum spanning trees in k-dimensional space. *SIAM J. Comput.*, 17:572–582, 1988.

[20] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in k dimensions. *Discrete Comput. Geom.*, 6:369–381, 1991.