Dynamic Well-Separated Pair Decomposition Made Easy

John Fischer* Sariel Har-Peled†

Abstract

We focus on making compressed quadtrees, particularly those used in the implementation of well-separated pair decomposition, into effective dynamic data structures. We use random shifting to achieve logarithmic insertion and deletion time per pair with high probability.

1 Introduction

Well-separated pair decomposition (WSPD), a means of concisely encapsulating distances among members of a finite point set in $\mathbf{R}^d$, has proved to be a useful tool in computational geometry; using it, one can readily obtain approximation algorithms for closest pair, nearest neighbor, and spanner problems in low dimension. Callahan and Kosaraju [CK95] defined and implemented WSPD using a *fair split tree*, arguably because this data structure lent itself to the design of effective parallel algorithms. However, the fair split tree is an unecessarily complex data structure to realize the WSPD of a point set. As an alternative, one can simply use a *compressed quadtree*, a data structure representing a dyadic partition of a hypercube; the compressed quadtree yields simpler WSPD algorithms.

Callahan [Cal95] developed methods for mantaining the fair split tree and the WSPD dynamically. For the fair split tree, Callahan appealed to Frederickson's [Fre97a] *topology tree* technique for maintaining binary trees dynamically. For the WSPD, the dummy points are introduced to guarantee that when one inserts or deletes points, the amount of change to the WSPD is small; unforutunately, the maintenance of the dummy points introduces a substantial level of complication into the dynamic maintenance of WSPD. In this paper, we show how this can be avoided.

Although a WSPD for a point set can be computed using a simple algorithm run over the point set's compressed quadtree, this particular WSPD for the point set may change significantly due the insertion of a new point — an artifact of the rigid high-level boundaries of quadtree cells. However, by randomly shifting our quadtree's boundaries, we can ensure that with high probability, insertion of a new point actually affects relatively few pairs of the WSPD. In particular, we can ensure that point insertion and deletion can be performed in $O([\log n + \log \varepsilon^{-1}]\varepsilon^{-d} \log n)$ time with high probability. Random shifting is an old idea, used by Arora [Aro98] for approximate Euclidean TSP, and in the work of Bartal and others on embedding of finite metric spaces [Bar96, Ind01]; however, this is the first work we know of that applies random shifting to WSPD.

2 Preliminaries

Definition 1 *Let H be a hypercube in $\mathbf{R}^d$. A* quadtree *over H is a partition tree of H, defined recursively: the root node r represents H, and the 2^d children of r are roots of quadtrees for each of the 2^d identical hypercubes obtained by making d axis-parallel cuts of H.*

When a node v represents a region H of $\mathbf{R}^d$, we say that H is the "cell" of v (since such regions lie within grids). Given a quadtree T, a *canonical cell* relative to T is a region of space that could serve as a cell of T (regardless of whether the cell explicitly appears in T).

In a quadtree defined over a finite point set P, a node v whose region contains at most one point of P has no children.

Definition 2 *Given a quadtree Q over a finite point set P, the* compressed quadtree *over P is obtained by (i) removing all leaf nodes whose regions contain no points of P; and then (ii) replacing any chain (maximal sequence of nodes with one child) with a single edge.*

Henceforth, unless otherwise stated, quadtrees and compressed quadtrees will be defined for d-dimensional n-point sets P of diameter $\Omega(1)$ lying within $[0,1]^d$; hence the cell corresponding to the root level of the quadtree will be simply $[0,1]^d$. This can be achieved by maintaining global translation and scaling factors, which in addition allows for a random shift to be applied to the quadtree with ease.

Definition 3 *The level of a quadtree node v, denoted $\ell(v)$, is the base two logarithm of the sidelength of its corresponding cell.*

The root has level zero, and all other nodes have negative level. Notice that in a compressed quadtree, the level of a node does not generally correspond to the node's distance from the root.

*Department of Computer Science, University of Illinois, jrfische@uiuc.edu

†Department of Computer Science, University of Illinois, sariel@uiuc.edu

```
WSPD(u, v, Q)
  Assume ℓ(u) > ℓ(v) or (ℓ(u) = ℓ(v) and u ⪯ v)
      (otherwise exchange u ↔ v).
  If 8√d · 2^ℓ(u) ≤ ε · ‖rep_u rep_v‖ then
    Return { {u, v} }
  Else
    Denote by u_1, ..., u_r the children of u
    Return ⋃_{i=1}^{r} WSPD(u_i, v, Q).
End WSPD
```

Figure 1: Algorithm for computing well-separated pair decomposition.

Definition 4 *Let P be a set of n points in $\mathbf{R}^d$, and $1/4 > \varepsilon > 0$ a parameter. A well-separated pair decomposition (WSPD) with parameter ε^{-1} of P is a set $\{\{A_1, B_1\}, \ldots, \{A_s, B_s\}\}$ of "WS pairs" such that*

1. *$A_i, B_i \subseteq P$ for every i;*
2. *$A_i \cap B_i = \emptyset$ for every i;*
3. *$\bigcup_{i=1}^{s} [A_i \otimes B_i] = P \otimes P$;*
4. *$\mathbf{d}(A_i, B_i) \geq \varepsilon^{-1} \cdot max\{diam(A_i), diam(B_i)\}$, where $\mathbf{d}(A_i, B_i) = min_{(p,q)\in(A_i,B_i)} \|pq\|$.*

Some WSPD involve fewer pairs than others; we thus consider the *size* of a WSPD to be the number of its WS pairs. Techniques for generating WSPD of small size given a point set P typically involve the creation of a partition tree of P; we use the compressed quadtree due to its simplicity relative to the fair split tree of [CK95].

Theorem 5 *Let P be an n-point set in $\mathbf{R}^d$. For $0 < \varepsilon \leq 1$, one can construct, in $O(n \log n + n\varepsilon^{-d})$ time, an ε^{-1}-WSPD of size $O(n\varepsilon^{-d})$.*

Proof. First, one constructs a compressed quadtree Q for P. One then traverses the tree to instill within each node v a *represenatative* $p = \text{rep}_v$; here p may be any point $p \in P$ lying in v's region. Given the root node of Q is r, one then uses the algorithm given in Figure 1 by making the call $\mathsf{WSPD}(r, r, Q)$. We omit the proof of correctness, which resembles that of [CK95]. □

To efficiently maintain the WSPD dynamically, we must modify not only the WS pair list itself, but also the compressed quadtree. In the case of insertion, given a compressed quadtree Q built over the n-point set P, we wish to add a new point p to P, updating the compressed quadtree quickly. The addition of p can only change Q in one of two minor ways: (i) a leaf node x representing p is simply hung from an existing node v; or (ii) between some node w and its parent v, a new node u is inserted, and x is hung from u (along with w).

In either case, were we to know the lowest node of the tree whose cell contains p, these operations would take constant time. One could simply descend from the root of Q to find this node; however, the compressed quadtree of an n-point set can have depth linear in n. Compressed quadtree point location can be improved to $O(\log n)$ time by building a *finger tree* (as in [AMN+98], for instance); however, it takes $O(n)$ time to build such a structure. We next address this problem.

3 Dynamic Compressed Quadtrees

3.1 Topology Trees

Frederickson ([Fre97a],[Fre97b])'s *topology tree* is defined over any unrooted tree of maximum node degree 3 or less. We first consider topology trees over binary (hence rooted) trees.

Binary Trees. To build a topology tree $\mathcal{T}$ over a binary tree T, one creates a hierarchical set of 'clusters' of nodes within the tree. We consider a *cluster* within T to be any contiguous set of nodes within T. Notice that a cluster has a "degree" – the number of edges with exactly one endpoint within the cluster.

We cluster T is as follows: first, set $T_0 = T$, and consider each node of T_0 to be a singleton cluster. Next, maximally cluster adjacent pairs of nodes within T_0, maintaining the invariant that no multinode cluster can be of degree 3; then, replace each cluster with a single node. This yields a new tree T_1. Repeat this procedure until for some k, T_k consists of a single node. Given T is a binary tree over n leaf nodes, [Fre97a] shows that $k = O(\log n)$.

The *topology tree* $\mathcal{T}$ for T is defined relative to this sequence: the node r of T_k is the root of $\mathcal{T}$, the two nodes of T_{k-1} that were combined to form r are the children of r in $\mathcal{T}$; and so on recursively. $\mathcal{T}$, a partition tree for the node set of T, has at most $k = O(\log n)$ levels; it is binary, and thus of linear size. Every cluster of $\mathcal{T}$ is of degree 1 or 2; each such cluster corresponds either to (i) a subtree S of T, or (ii) to a structure obtained by removing from such an S one of its own subtrees (the only unusual case is that of $S = T$).

Operations of *edge insertion* and *edge deletion* on unrooted trees T are defined in [Fre97a, Fre97b]: the former joins two trees into one by adding an edge to bridge the trees, and the latter creates two trees from one by removing an edge within the tree. The time required to update (merge or split) the appropriate topology trees is shown to be $O(\log m)$, where m is the number of nodes of the original tree.

Trees of Bounded Degree. Defining topology trees over trees of arbitrary bounded degree directly tends to make updates baroque [Fre97a]. Instead, we adopt the general approach employed by [Fre97a]: conceptually replace each quadtree node with a complete d-level bi-

nary tree, with each level of this tree corresponding to a split of the quadtree cell along a different principal axis; then, define the topology tree over this binary variant of the compressed quadtree.

3.2 Dynamic Maintenance Operations

Point Location. Given a point p, a compressed quadtree Q, and its topology tree $\mathcal{T}$, we wish to find, in $O(\log n)$ time, the lowest-level node v of Q for which the point p lies in the region of space corresponding to v. We can do so by simply traversing $\mathcal{T}$ downward from its root. The straightforward details are omitted.

Lemma 6 *Suppose that $\mathcal{T}$ is the topology tree built over a compressed quadtree T, and that the depth of $\mathcal{T}$ is h. Given a point $p \in [0,1]^d$, one can find the deepest node of T whose region contains p in $O(h)$ time.*

Cell Queries. Though we often seek the smallest compressed quadtree node whose region contains a given point, we may also seek the opposite: given a canonical cell C relative to a compressed quadtree Q defined over an n-point set P, we wish to determine quickly whether or not C is devoid of points of P. We refer to this operation as a *cell query*; it can also be done by traversing $\mathcal{T}$ downward from its root. The details are omitted.

Lemma 7 *Given a compressed quadtree Q of an n-point set P with corresponding topology tree $\mathcal{T}$, and a canonical cell C relative to Q, one can find, in $O(\log n)$ time, the node $w \in Q$ whose region R satisfies $R \subseteq C$ and $P \cap C = P \cap R$.*

Since edge insertions/deletions on a compressed quadtree with topology tree can be performed in $O(\log n)$ time, it suffices to describe point (i.e. leaf node) insertions/deletions by a finite series of edge insertions/deletions.

Insertion. We first use point location to find v, the node from which we must hang a leaf node x representing p. (The topology tree over x can be created in constant time.) The two scenarios described earlier again apply. In the first, we simply hang x from v, by performing a single edge insertion (from x to v). In the second, we create a second new node u, use an edge deletion and two edge insertions to place u between v and its appropriate child w, determine from the coordinates of p which subcell of u contains p, and hang x from u in the corresponding location.

Deletion. To remove a point p from within Q, first perform point location to find the lowest node v' whose space region contains p; then, perform an edge deletion on the edge connecting v to its parent w. If w has only one child, perform two edge deletions and an edge insertion to cut w out of the quadtree.

4 Limiting WS Search by Random Shifting

Before building the compressed quadtree, we shift the grid system to be placed over the point set randomly. For instance, rather than having the root node of a quadtree for P correspond to the cube $[0,1]^d$, we have it instead correspond to $\bigotimes_{i=1}^{d}[t_i - 1, t_i + 1]$ for $t_i \in [0,1]$. (The actual implementation of this effect involves a simple update of the the global scale and shift factors previously mentioned.)

Path distance between a pair of nodes in a compressed quadtree need not correspond, however approximately, to actual distance between the nodes; but by shifting the cells randomly, intuitively one expects it to become less likely that a close pair of points will lie across a boundary between large cells.

Lemma 8 *Suppose a pair of points $\{p, q\}$ is separated by distance λ. Upon randomly shifting a quadtree grid of sidelength 2^i over p and q, the probability that p and q are separated by the quadtree cell boundary at level i is at most $d\lambda/2^i$.*

Proof. Let us instead translate the segment pq over a fixed grid of sidelength 2^i. We consider positions of p for which pq may intersect the grid. Supposing $e_1, \ldots, e_d$ are basis vectors aligned with the grid, consider a particular e_i. Along this direction, we overestimate the region R of a grid cell in which intersection might occur by making R of width λ in the e_i-direction, and of width 2^i along the $d-1$ other principal directions. With d such regions, one for each basis vector, we again upper-bound the size of a grid cell's intersection region by simply adding the sizes of the d component regions. Thus, the volume of the region within a grid cell within which placement of p would lead to intersection of pq with the grid is at most $d\lambda(2^i)^{d-1}$. Hence the probability that a randomly placed p will effect an intersection is at most $[d\lambda(2^i)^{d-1}]/(2^i)^d = d\lambda/2^i$. □

The probability that p and q are separated by the quadtree cell boundary at level $i = \lg\lambda + 4\lg n$ is at most d/n^4. If p and q are cut by the boundary at level i, then they will also be cut by all finer boundaries. Hence the probability that $\lg\lambda + 4\lg n$ is the *highest* level on which p and q are cut is also at most d/n^4.

We may extend this observation to a statement about the entire point set P. Given Y is the event that for *each* pair $\{p, q\}$, p and q are cut on no level higher than $\lfloor \lg\lambda + 4\lg n \rfloor$, $Pr[Y] \leq \binom{n}{2} \cdot d/n^4 \leq d/n^2$.

Lemma 9 *With high probability, for any given pair $\{p, q\}$ of points lying in the d-dimensional unit cube, the highest level of a randomly shifted quadtree defined over this cube on which p and q lie in separate canonical cells is $\lfloor \lg\lambda + 4\lg n \rfloor$, where $\lambda = \|pq\|$.*

5 Maintaining WSPD Dynamically

We have seen how to update the compressed quadtree Q of an n-point set P upon removal or addition of a point p to P. It remains to examine how the WSPD changes due to these updates.

5.1 Insertion and WSPD Extraction

Given compressed quadtree Q rooted at r, the WSPD creation algorithm of Figure 1 is run by calling $\mathsf{WSPD}(r, r, Q)$. Recursion ends once a pair π of nodes is found to be well-separated; this pair π is then output. Recursing further is possible, but unnecessary; since π is well-separated, any pair whose members are subsets of the members of π must necessarily be a WS pair as well. Given this, one can verify that the algorithm exhausts all possible pairs of nodes within the tree.

We have seen that inserting a new point p into a quadtree Q yields a new quadtree $\hat{Q}$ that is almost identical to Q. Recall the general scenario of inserting new node x under existing node v: either x is hung directly under v, or x is hung under a new node u inserted between v and its child w. Hence $\hat{Q}$ again contains all the cells of Q, except for the possible introduction of u. The cell of u subsumes the cell of w; hence the WSPD algorithm given in Figure 1, when run on $\hat{Q}$, will output every WS pair output when the algorithm is run on Q, with the exception that some pairs in Q involving w may now involve the more general node u instead, since u is encountered before w when recursively testing for WS pairs.

More formally, we seek to modify a list L of WS pairs for P (which was generated by the algorithm of Figure 1) in order to yield $\hat{L}$, a list of WS pairs for $\hat{P} = P \cup \{p\}$. By the argument above, in order to do this so that $\hat{L}$ matches exactly the list that would be produced by Figure 1 on $\hat{P}$, two modifications are required: (i) add to L all new WS pairs that involve p; and (ii) update every pair of L in which u should appear in place of w.

5.2 Finding New Pairs

Suppose that the nearest neighbor to p in P is q, and that $\|pq\| = \lambda$. Two observations help us in our search for new WS pairs (all of which somehow involve p).

Observation 10 *Consider a quadtree cell n containing p of sidelength $c\varepsilon\lambda$. For a sufficiently small positive constant c, n will form a WS pair with any node of Q (i.e. any cell of Q containing a point of P) of level $\lfloor \lg\lambda + \lg\varepsilon \rfloor$ or lower.*

Observation 11 *By Lemma 9, with high probability, a quadtree node in which p appears alone can be of level at most $\lfloor \lg\lambda + 4\lg n \rfloor$.*

We therefore need only search for new WS pairs among quadtree cells of level between $\lfloor \lg\lambda + \lg\varepsilon \rfloor$ and $\lfloor \lg\lambda + 4\lg n \rfloor$. This search can be performed exhaustively using quadtree cell queries. The details of this search, and of the maintenance of the list of WS pairs, are omitted for lack of space; the result follows:

Theorem 12 *Let L be a list of WS pairs for an n-point set P, generated using randomly shifted compressed quadtree Q and topology tree T. With high probability, in time $O([\log n + \log\varepsilon^{-1}]\varepsilon^{-d}\log n)$, one can modify L to yield $\hat{L}$, the list of WS pairs that is generated using the compressed quadtree $\hat{Q}$ and topology tree $\hat{T}$ defined for the point set $\hat{P} = P \cup \{p\}$.*

Deletion involves similar searches to identify WS pairs; again, the details are omitted for brevity.

Theorem 13 *Let L be a list of WS pairs for an n-point set P, generated using compressed quadtree Q and topology tree T. With high probability, in time $O([\log n + \log\varepsilon^{-1}]\varepsilon^{-d}\log n)$, one can modify L to yield $\hat{L}$, the list of WS pairs that is generated using the compressed quadtree $\hat{Q}$ and topology tree $\hat{T}$ defined for the point set $\hat{P} = P \setminus \{p\}$.*

References

[AMN+98] 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(6), 1998.

[Aro98] S. Arora. Polynomial-time approximation schemes for euclidean TSP and other geometric problems. *J. ACM*, 45(5), 1998.

[Bar96] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proc. 37th Annu. IEEE Symp. Found. Comput. Sci.*, pages 183–193, 1996.

[Cal95] P. B. Callahan. *Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications.* PhD thesis, Johns Hopkins University, 1995.

[CK95] 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.

[Fre97a] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge connectivity and k smallest spanning trees. *SIAM J. Comput.*, 26(2):484–538, April 1997.

[Fre97b] G. N. Frederickson. A data structure for dynamically maintaining rooted trees. *Journal of Algorithms*, 24:37–65, 1997.

[Ind01] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, pages 10–31, 2001.