Output-Sensitive Well-Separated Pair Decompositions for Dynamic Point Sets*

Eunhui Park
Department of Computer Science
University of Maryland
College Park, Maryland
ehpark@cs.umd.edu

David M. Mount
Department of Computer Science
University of Maryland
College Park, Maryland
mount@cs.umd.edu

ABSTRACT

The well-separated pair decomposition (WSPD) is a fundamental structure in computational geometry. Given a set P of n points in d-dimensional space and a positive separation parameter s, an s-WSPD is a concise representation of all the $O(n^2)$ pairs of P requiring only $O(s^d n)$ storage. The WSPD has numerous applications in spatial data processing, such as computing spanner graphs, minimum spanning trees, shortest-path oracles, and statistics on interpoint distances. We consider the problem of maintaining a WSPD when points are inserted to or deleted from P.

Worst-case arguments suggest that the addition or deletion of a single point could result in the generation (or removal) up to $\Omega(s^d)$ pairs, which can be unacceptably high in many applications. Fortunately, the actual number of well separated pairs can be significantly smaller in practice, particularly when the points are well clustered. This suggests the importance of being able to respond to insertions and deletions in a manner that is output sensitive, that is, whose running time depends on the actual number of pairs that have been added or removed. We present the first output-sensitive algorithms for maintaining a WSPD of a point set under insertion and deletion. We show that our algorithms are nearly optimal, in the sense that these operations can be performed in time that is roughly equal to the number of changes to the WSPD.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms

*This work has been supported by the National Science Foundation under grant CCF-1117259 and the Office of Naval Research under grant N00014-08-1-1015.

SIGSPATIAL'13, November 05 - 08 2013, Orlando, FL, USA

Keywords

Well-separated pair decomposition, output-sensitive algorithms, dynamic algorithms.

1. INTRODUCTION

The well-separated pair decomposition (WSPD) is a fundamental structure in computational geometry and processing of spatial data. A set P of n points determines $\Theta(n^2)$ pairs of points. The WSPD provides a concise representation of these pairs using only $O(n)$ space, subject to a given approximation error in the accuracy to which each distance is represented. (See definitions below.) In this paper, we are interested in the problem of maintaining the WSPD of a dynamic point set, where insertions and deletions are possible.

The concept of the WSPD was introduced by Callahan and Kosaraju [2], but it was anticipated in earlier work by Greengard and Rokhlin on the fast multipole method [8], which is widely used in physical simulations. Given a positive parameter s, called the *separation factor*, we say that two sets A and B in a metric space are *s-well separated* if they can each be enclosed within balls of equal radii such that the closest distance between these balls is at least s times their common radius (see Fig. 1).

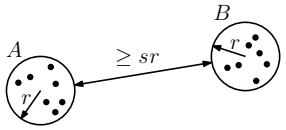

Figure 1: A well-separated pair.

Given an n-element point set P and $s > 0$, a *well-separated pair decomposition* of P with respect to s (an *s-WSPD*) is a collection of pairs $\{A_1, B_1\}, \ldots, \{A_k, B_k\}$ of non-empty subsets of P such that

(1) for $1 \le i \le k$, A_i and B_i are s-well separated, and

(2) for any two distinct points $p, q \in P$, there exists exactly one pair $\{A_i, B_i\}$ such that p lies in one of these sets, and q lies in the other.

Throughout, we assume that the points reside in $\mathbb{R}^d$, for a fixed constant d under the L_∞ distance function. (Later, we will show that replacing the metric with any Minkowski metric, such as the Euclidean or Manhattan distance, affects only the constant factors. Well-separated pair decompositions can also be readily extended to any metric space

of constant doubling-dimension [10].) We treat s and n as asymptotic quantities, and we will assume that $s \geq 1$.

WSPDs have numerous applications in the design of both exact and approximate algorithms for a spatial data sets. Given an n-element point set P in $\mathbb{R}^d$, it is possible to use WSPDs to compute the (exact) closest pair of points of P in $O(n \log n)$ time, and generally for $1 \leq k \leq \binom{n}{2}$, it is possible to enumerate the (exact) k closest pairs of points $O(k + n \log n)$ time [18]. It is also possible to compute the nearest neighbor for every point in the set in $O(n \log n)$ time [18]. WSPDs are also used in geometric approximations. For example, it is possible compute an ε-approximation to the diameter of a points set (and generally to approximate the kth smallest interpoint distance) in $O(n \log n + (1/\varepsilon)^d)$ time. It is also possible to use WSPDs for computing approximations to the Euclidean minimum spanning tree [19], constructing geometric spanner graphs (sparse networks whose shortest paths are not much longer than the straight-line distance between points) [12], and building distance oracles for spatial networks [16]. In order to apply any of these algorithms a dynamic context, it is natural to ask how quickly can the WSPD be updated when points are inserted or deleted.

It is well known (see, e.g., [2, 9, 19]) that, given an n-element point set in $\mathbb{R}^d$, its possible to build an s-WSPD of size $k = O(s^d n)$ in time $O(n \log n + k)$. This is nearly optimal in the worst case, since it is easy to prove that a uniformly distributed point set is expected to require $\Omega(s^d n)$ pairs. It is conventional wisdom, however, that typical point sets are far from uniform. Effects such as clustering, correlations among coordinates, and boundary effects tend to reduce the number of well-separated pairs. This reduction can be quite significant, even in spaces of moderate dimension [13]. For this reason, it is important to consider *output-sensitive algorithms*, that is, algorithms whose running time depends on the number of changes to the WSPD structure. While the aforementioned batch WSPD construction is output sensitive, existing algorithms for maintaining a WSPD are not.

Our interest in WSPDs arises from compression and clustering applications involving points derived from digital terrain elevation maps. Given an $m \times m$ elevation map and an integer $k < m$, we can partition the map into $k \times k$ square blocks, each of which is treated as a point in k^2-dimensional space. These points can be clustered in order to identify regions of similar local topography, and generally WSPD decompositions can be used to analyze local topographical relationships. To demonstrate the value of an output sensitive approach, we considered two data sets. The first involved a 2400×2400 elevation map from the area near Washington, D.C. (A small random perturbation was added to each elevation to avoid duplicate values.) The second involved a synthetically generated elevation map of the same size in which the elevations were sampled from a uniform distribution over a similar range of elevations. In each case, the map was decomposed into 2×2 blocks, which were then interpreted as a set of $(2400/2)^2 = 1.44$ million points in $\mathbb{R}^4$. The plot on the left of Figure 2 shows the number of newly created well-separated pairs by the insertion of a single point into both the Washington, D.C. ("real") data set and the random ("uniform") data set as a function of the separation factor. The plot on the right shows the ratio between them. As the separation factor grows, the ratio grows to over two orders of magnitude.

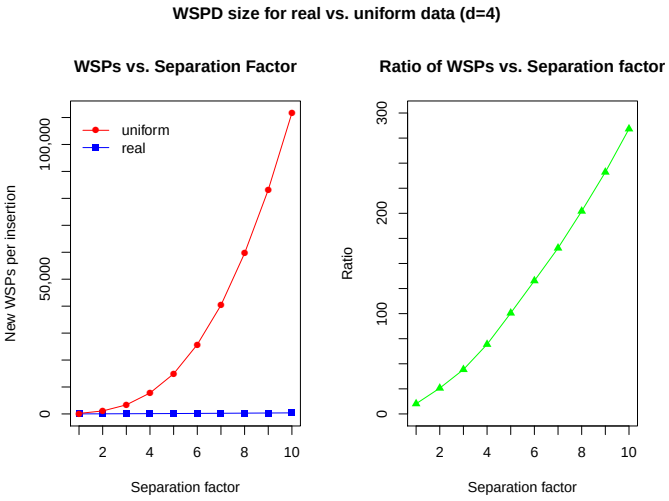

Figure 2: New well-separated pairs resulted from the insertion of a single point for terrain data and uniformly distributed data in $\mathbb{R}^4$.

WSPD constructions in $\mathbb{R}^d$ are based on hierarchical spatial subdivisions, and in particular, variants quadtrees and kd-trees [15]. Callahan and Kosaraju used a fair-split tree. Fischer and Har-Peled [4] point out that the construction can be based on the simpler compressed quadtree, which is the approach that we take here (see Section 2 for definitions). WSPD construction algorithms all have the same general structure. The points are stored in the leaves of an appropriate partition tree of size $O(n)$. The k sets of the WSPD are generated by a recursive top-down algorithm and are represented as pairs of nodes of the tree, $\{u_i, v_i\}$ for $1 \leq i \leq k$ (see Fig. 3). The corresponding pairs $\{A_i, B_i\}$ of the decomposition consist of the points of the subtrees rooted at u_i and v_i, respectively (that is, contained within their respective cells). Construction algorithms generate only *maximal* pairs, meaning that the parent pair, $\{\text{par}(u_i), \text{par}(v_i)\}$, is not s-well-separated.

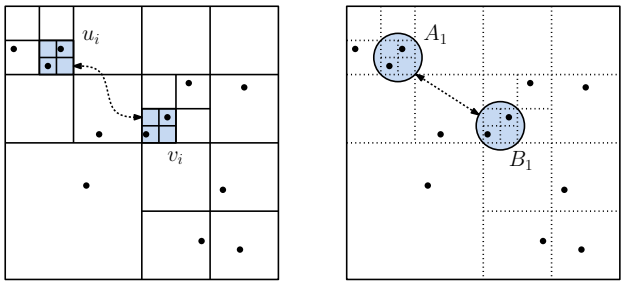

Figure 3: Each WSP is represented by a pair of nodes in the quadtree.

Clearly, a given point set does not have a unique s-WSPD. Fischer and Har-Peled [4] observed this issue in their algorithm for maintaining the WSPD of a dynamic point set. They observed that the number of well separated pairs can vary based on the choice of the coordinate system's origin,

which in turn affects the quadtree structure. They showed that this instability could be ameliorated in expectation by applying a random translation to the point set. In particular, they showed that it is possible to maintain an s-WSPD of a point set under insertions and deletions in time $O(s^d(\log n + \log s)\log n)$ with high probability. WSPDs have also been considered in a dynamic context mostly in the context of metric spaces. Examples include the deformable spanner of Gao, Guibas, and Nguyen [6] and dynamic spanners by Gottlieb and Roditty [7] and Roditty [14].

Since the output size is not uniquely determined by the input, this raises the question of what is meant by an "output sensitive" algorithm. Our approach will be to fix a precise notion of s-WSPD procedurally, that is, as the output of a particular algorithm. In order to achieve the best running times, we will strengthen the notion of separation to one that is easy to compute in a quadtree setting. We will show that for any given point set, the size of the s-WSPD, under our notion of strong separation is at most a constant factor larger than the size of any (standard) s-WSPD for this point set in any Minkowski L_p metric, for $p \geq 1$. Our approach can be readily generalized to any quadtree-like subdivision.

We relate the update time to the number of changes made to the WSPD. In our algorithm (as in [4]), there are two ways that well-separated pairs can change. When a point p is inserted, new pairs of the form $\{\{p\}, B_i\}$ can be *created*, and an existing pair $\{A_i, B_i\}$ can be *modified* to become $\{A_i \cup \{p\}, B_i\}$ (see Figure 4). Deletion is just reverse.

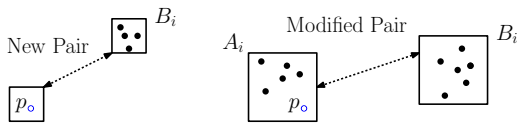

Figure 4: New and modified pairs.

Because only newly created or deleted pairs affect our quadtree-based representation, our output-sensitive bounds will be based on their number. Our running times do not depend on the number of modified pairs. This is significant, because newly created or deleted pairs tend to arise in the locality of the inserted or deleted point, respectively, whereas modified pairs can arise at all scales. Of course, depending on the application, the modified pairs might also be of interest. If desired, it is easy to modify our algorithm to report the modified pairs in an output sensitive manner.

The following theorem presents our main result. We assume that point coordinates are represented as fixed-point binary numbers that support bit-wise operations (such as boolean-and, boolean-or, and shift) in $O(1)$ time. Our deletion operation employs a form of "lazy updating," where some housekeeping operations are deferred until later, which results in amortized running time bounds.

THEOREM 1. *Given an n-element dynamic set $P \subset \mathbb{R}^d$ and a parameter $s \geq 1$, it is possible to maintain an s-WSPD for P in the L_∞ metric that supports the following operations and running times:*

- Insert point: $O(\log n + m)$ *(worst case)*
- Delete point: $O(\log n + m)$ *(amortized)*

where m denotes the number of newly created (resp., deleted) pairs in the case of insertion (resp., deletion). (Hidden constants depend on d, but are independent of s.)

The paper is organized as follows. In Section 2, we present basic definitions and introduce our stronger notion of separation. In Section 3 we present static algorithms and utilities for constructing the WSPD. These are used by our update algorithms, which are given in Section 4. Conclusions follow in Section 5.

2. PRELIMINARIES

We begin by recalling some basic concepts related to quadtrees, upon which our constructions are based. Let $\mathbb{U}^d = [0,1)^d$ denote the half-open d-dimensional unit hypercube. We define a *quadtree box* by the following recursive process. $\mathbb{U}^d$ is a quadtree box, and given any quadtree box b, each of the 2^d regions that result by subdividing b into congruent (half-open) hypercubes is also a quadtree box. These 2^d boxes are called the *children* of b. The subdivision process naturally defines a 2^d-ary partition tree whose nodes correspond to quadtree boxes. Henceforth, we use the term "box" to mean "quadtree box."

Consider an n-element point set $P \subset \mathbb{U}^d$. Such a set naturally induces a quadtree subdivision, by repeatedly subdividing each box that contains two or more points of P. Because the size of the resulting subdivision may be much larger than n, it is common to compress *trivial paths*, in which each internal node has only one nonempty child. A *compressed quadtree* is obtained by (i) removing all leaf boxes that contain no point of P and then (ii) replacing any maximal trivial path with a single edge that goes to the first descendent with two or more children (see Figure 5). In any fixed dimension, it is well known that such a quadtree is of size $O(n)$ and can be constructed in time $O(n \log n)$ (see, e.g. [3, 9]). Given a node u in a compressed quadtree, let $\mathrm{par}(u)$ denote its parent.

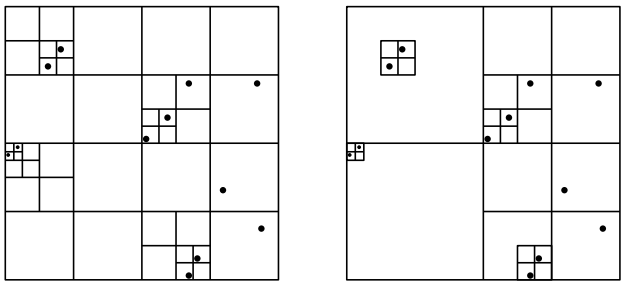

Figure 5: Standard and compressed quadtrees.

Each node of a compressed quadtree is naturally associated with a box, which is often called its *cell*. Following [9], we define the *level* of a box u, denoted $\mathrm{lev}(u)$, to be the base-2 logarithm of its side length, and we define the level of a node to be the level of its associated box. (Throughout, we use lg to denote the base-2 logarithm. Because we start with $\mathbb{U}^d$, a node's level is the negation of its depth in a standard quadtree.) It is convenient to think of a point as a degenerate box of side length zero, that is, a box at level $-\infty$. (We will often blur the distinction between a node and its associated box by using the same name for both.) Given an integer $k \geq 0$ and a box u such that $\mathrm{lev}(u) \leq -k$, let $u^{\uparrow k}$ denote the (unique) box at level $\mathrm{lev}(u) + k$ that contains u. (Due to compression, this box need not correspond to any node of the tree.)

Given a compressed quadtree for P and a box b, define node(b) to be node of the highest level in the tree whose associated box is contained within (or is equal to) b. If no such node exists (which happens if b contains no point of P) then we assume that node(b) returns a special *null* value. By storing the quadtree in a balanced structure (e.g., as a topology tree [5], link-cut tree [17], or quadtreap [11]) node(b) can be computed in $O(\log n)$ time. This is called a *box query*.

As mentioned in the introduction, we compute the WSPD assuming the L_∞ metric. Given two boxes u and v, define $\text{dist}(u, v)$ to be the minimum L_∞ distance between any two points, one from u and one from v. Since the radius of the smallest L_∞ ball enclosing a box is half the box's side length, it follows that u and v are s-well separated if $\text{dist}(u, v) \geq s \cdot 2^\ell/2$, where $\ell = \max(\text{lev}(u), \text{lev}(v))$.

2.1 Strong Separation

For the sake of simplicity and efficiency, we will need a stronger notion of separation in our algorithms. (We shall see in Lemma 3 below that this alters the size of the WSPD by only a constant factor.) We start by introducing a quadtree-based notion of separation. We say that two boxes u and v are *homogeneously s-well separated* if there exist two (quadtree) boxes u' and v' of equal level such that $u \subseteq u'$ and $v \subseteq v'$, and u' and v' are s-well separated. If two boxes u and v are homogeneously s-well separated, then clearly they are s-well separated. (As shown in Figure 6(a), the converse does not hold.)

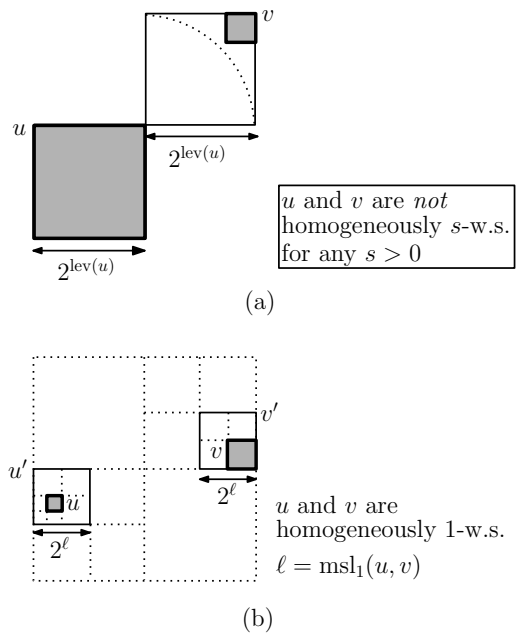

Figure 6: Homogeneous separation.

To minimize the size of the WSPD, it is important that the well-separated pairs are maximal in some sense. If u and v are homogeneously s-well separated, there exists a unique maximum level ℓ at which u' and v' reside. We call this the *maximum separation level*, and we denote it by $\text{msl}_s(u, v)$ (see Figure 6(b)).

Assuming a model of computation supporting bit manipulations, $\text{msl}_s(u, v)$ can be computed in constant time, for any boxes u and v. (For our algorithms, we will only need $\text{msl}_1(u, v)$.)

LEMMA 1. *Assume we have a model of computation that supports in $O(1)$ time the bitwise operations boolean-or ("$\vee$"), complement ("$\overline{x}$"), and exclusive-or ("$\oplus$"), left shift ("$\ll$"), and integer base-2 logarithm (that is, the index of the most significant bit) on point coordinates. Given two boxes u and v that are homogeneously 1-well separated, it is possible to compute $\text{msl}_1(u, v)$ in $O(1)$ time.*

PROOF. Consider a homogeneous 1-well separated pair $\{u, v\}$. To motivate our construction, let $\ell = \text{msl}_1(u, v)$, and let u' and v' be the boxes of level ℓ containing u and v, respectively. Let $p' = (p'_1, \ldots, p'_d)$ and $q' = (q'_1, \ldots, q'_d)$ denote the lower left corners of u' and v', respectively. Since u' and v' are 1-well separated, we have $\text{dist}(u', v') \geq 2^\ell/2$, and by basic properties of boxes and the definition of the L_∞ distance, we have $\max_i |p'_i - q'_i| > 2^\ell$. Therefore, $\max_i \left|\frac{p'_i}{2^\ell} - \frac{q'_i}{2^\ell}\right| > 1$. Note that for any point p in u', $\forall i$, $\left\lfloor \frac{p_i}{2^\ell} \right\rfloor = \frac{p'_i}{2^\ell}$. Thus, letting p and q be the lower left corners of u and v, respectively, we have $\forall i$, $\left\lfloor \frac{p_i}{2^\ell} \right\rfloor = \frac{p'_i}{2^\ell}$, and $\left\lfloor \frac{q_i}{2^\ell} \right\rfloor = \frac{q'_i}{2^\ell}$. Thus, to compute $\ell = \text{msl}_1(u, v)$ it suffices to compute the largest $\ell \leq 0$ such that

$$\max_{1 \leq i \leq d} \left| \left\lfloor \frac{p_i}{2^\ell} \right\rfloor - \left\lfloor \frac{q_i}{2^\ell} \right\rfloor \right| > 1. \tag{1}$$

Algorithm 1 Find $\text{msl}_1(u, v)$

Input: Homogeneously 1-well separated boxes u and v.
Output: The maximum separation level of u and v, that is, $\text{msl}_1(u, v)$.

```
1: function findMaxLevel(u, v)
2:     p : the lower left corner of u
3:     q : the lower left corner of v
4:     m ← arg max_i |p_i − q_i|
5:     x_1 ← max(p_m, q_m)    y_1 ← min(p_m, q_m)
6:     j_1 ← ⌊lg(x_1 ⊕ y_1)⌋
7:     x_2 ← x_1 ≪ j_1    y_2 ← y_1 ≪ j_1
8:     j_2 ← ⌊lg(x_2 ∨ ȳ_2)⌋
9:     return j_1 + j_2
```

The code implementing this is presented in Algorithm 1. Let p and q denote the lower left corners of u and v, respectively. The procedure first computes the largest absolute coordinate m of $p - q$ ($1 \leq m \leq d$). We let x_1 and y_1 be the maximum and minimum of p_m and q_m, respectively. Recalling that the points lie within the half-open unit hypercube $[0, 1)^d$, $\left\lfloor \frac{x}{2^j} \right\rfloor$ yields the first j bits in x's bit representation. We compare the leading j bits of x_1 and y_1 for increasing values of j. Let j_1 be the first position where x_1 and y_1 have different bit values (see Figure 7). Since x_1 is greater than y_1, the j_1th bits of x_1 and y_1 are 1 and 0, respectively. Line 6 finds this position j_1 by computing the index of the most significant bit of $x_1 \oplus y_1$. We then shift these matching bits off the left end of the bit string. (We assume that bits shifted to the left of the decimal point are discarded.) We let x_2 and y_2 be the results shifting x_1 and y_1 by j_1, respectively (Line 7).

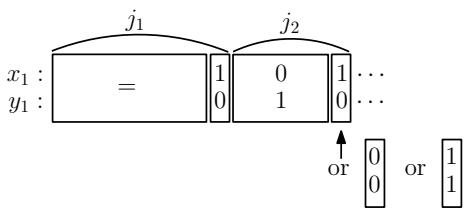

Figure 7: Find the maximum separation level of homogeneous 1-well separated pair

Next, we find the first position, j_2 following this where x_2's bit is not 0 and y_2's bit is not 1 (see Figure 7). This is done by computing the index of the most significant bit of $x_2 \vee \overline{y_2}$ (Line 8). The result corresponds to the first position where their difference is greater than 1. It follows that $j_1 + j_2$ is the desired level. □

In order to define our stronger notion of separation, let $s \geq 1$ denote the WSPD separation factor, which will be fixed henceforth, and define

$$\sigma \;=\; \max\left(\left\lceil \lg \frac{s}{2} \right\rceil, 0\right).$$

We say that two boxes u and v are *strongly s-well separated* if $u^{\uparrow\sigma}$ and $v^{\uparrow\sigma}$ are homogeneously 1-well separated. As in homogeneous well separation, there exists a unique maximum separation level ℓ at which two such boxes are strongly s-well separated, which we denote by $\mathrm{msl}^*_s(u,v)$. Clearly, $\mathrm{msl}^*_s(u,v) = \mathrm{msl}_1(u^{\uparrow\sigma}, v^{\uparrow\sigma}) - \sigma$. Our next lemma shows that this is indeed a stronger notion of separation.

LEMMA 2. *If two boxes u and v are strongly s-well separated, then they are s-well separated.*

PROOF. If u and v are strongly s-well separated, then by definition, $u^{\uparrow\sigma}$ and $v^{\uparrow\sigma}$ are homogeneously 1-well separated. Note that if any two boxes of equal level are not adjacent, they are separated from each other by an (L_∞) distance of at least their side length. Letting $\ell = \max(\mathrm{lev}(u), \mathrm{lev}(v))$, we have

$$\begin{aligned}\mathrm{dist}\big(u^{\uparrow\sigma}, v^{\uparrow\sigma}\big) &\geq 2^{\max(\mathrm{lev}(u^{\uparrow\sigma}),\mathrm{lev}(v^{\uparrow\sigma}))} \\ &= 2^{\max(\mathrm{lev}(u)+\sigma,\mathrm{lev}(v)+\sigma)} \\ &= 2^{\sigma}\cdot 2^{\ell} \;\geq\; \frac{s}{2}\cdot 2^{\ell}.\end{aligned}$$

Note that since $u^{\uparrow\sigma}$ and $v^{\uparrow\sigma}$ contain u and v, respectively, we have $\mathrm{dist}(u,v) \geq \mathrm{dist}\big(u^{\uparrow\sigma}, v^{\uparrow\sigma}\big)$. Therefore, $\mathrm{dist}(u,v) \geq s\cdot 2^{\ell}/2$, which implies that u and v are s-well separated. □

The notion of maximality can be generalized to this context. Consider two nodes u and v that are strongly s-well separated. Let $u' = \mathrm{node}(u^{\uparrow\sigma})$ and $v' = \mathrm{node}(v^{\uparrow\sigma})$. By definition, u' and v' are homogeneously 1-well separated. We say that the pair (u,v) is *maximal* if $\mathrm{par}(u')$ and $\mathrm{par}(v')$ are not homogeneously 1-well separated. Our algorithms generate only maximally well-separated pairs.

If we modify the definition of s-WSPD by replacing the standard notion of separatedness with this stronger notion, we obtain a structure called a *strong s-well separated pair decomposition* (strong s-WSPD). The following lemma shows that by using this stronger notion of separation, we sacrifice only a constant factor in the size of the WSPD, in comparison to *any* standard s-WSPD and in *any* Minkowski metric.

LEMMA 3. *Given a points set P in $\mathbb{R}^d$ and $s \geq 1$, let Ψ be a strong s-WSPD (in the L_∞ metric) consisting of maximal pairs. Let Ψ' be any (standard) s-WSPD in the L_p metric for any $p \geq 1$. Then, there exists a constant c (depending on d but independent of s and p) such that $|\Psi| \leq c\cdot|\Psi'|$.*

PROOF. Because Ψ consists only of maximal pairs, it suffices to show that each well-separated pair $\{A_i, B_i\} \in \Psi'$ can be covered by at most a constant number of strong s-well-separated pairs. By definition, A_i and B_i can each be enclosed within an L_p ball of radius r such that the closest L_p distance between these balls is at least sr. In $\mathbb{R}^d$, if the L_p distance between two points is z, then their L_∞ distance is at least $z/d^{1/p} \geq z/d$ (since $p \geq 1$). Therefore, the closest L_∞ distance between these balls is at least sr/d.

Let ℓ be the largest integer (possibly negative) such that $2^\ell \leq sr/(3d)$. Consider the grid of quadtree boxes of side length 2^ℓ. The L_∞ diameter of any box of the grid is 2^ℓ. Let u' and v' be any boxes of this grid that have a nonempty intersection with A_i and B_i, respectively. By the triangle inequality, the closest L_∞ distance between u' and v' is at least

$$\begin{aligned}\frac{sr}{d} - \mathrm{diam}(u') - \mathrm{diam}(v') &= \frac{sr}{d} - 2\cdot 2^{\ell} \\ &\geq \frac{sr}{d} - 2\frac{sr}{3d} \;=\; \frac{sr}{3d} \\ &\geq 2^{\ell}.\end{aligned}$$

It follows that u' and v' are homogeneously 1-well separated.

Let u and v be any quadtree boxes at level $\ell-\sigma$ that have a nonempty intersection with A_i and B_i, respectively. Clearly, $u^{\uparrow\sigma}$ and $v^{\uparrow\sigma}$ intersect A_i and B_i, respectively, and so by the above analysis, they are homogeneously 1-well separated. Therefore u and v are strongly s-well separated. The side length of these boxes is

$$2^{\ell-\sigma} \;\geq\; \frac{2^\ell}{2^{1+\lg\frac{s}{2}}} \;=\; \frac{2^\ell}{s}.$$

By definition of ℓ, we have $2^\ell > sr/(6d)$, and therefore u and v are both of side length at least $r/(6d)$. It follows from a standard packing argument, that the number of such boxes that can have a nonempty intersection with an L_p ball of radius r is a constant γ. (In general, γ depends on the dimension but not on s or p. A more careful analysis reveals that $\gamma \leq \lceil 1+6d\rceil^d$ [1].)

Since at most γ quadtree boxes at level $\ell-\sigma$ suffice to cover each of A_i and B_i, it follows that the number of strong s-well-separated pairs that cover the pairs $A_i \times B_i$ is at most γ^2. In summary, each (standard) s-well-separated pair of Ψ' can be covered by at most $\gamma^2 = O(1)$ strong s-well-separated pairs of Ψ, which completes the proof. □

Throughout the remainder of the paper, we will use the term "s-well separated" to mean strongly s-well separated and we use "s-WSPD" to mean strong s-WSPD. It is natural to wonder whether the constant factor increase in the size of the WSPD due to strong separation is large enough to wipe out the performance gains obtained through the use of output sensitivity. Of course, this depends the distribution of the data set. However, in our experience with data from our terrain clustering application, the increase in the size of the WSPD due to strong-separation was typically an order of magnitude smaller than increase in the number of WSPDs due to failure to consider output sensitivity.

3. CONSTRUCTION AND UTILITIES

In this section, we present the basic construction algorithm for computing an s-WSPD for any $s \geq 1$. Our construction algorithm employs the standard recursive process for computing WSPDs (see Algorithm 2). Given a pair of nodes u and v, it first dispenses with the trivial cases (either node is empty or the two nodes are the same leaf). If the nodes are homogeneously 1-well separated, then it computes $\ell_1 = \text{msl}_1(u, v)$ and calls a utility function, findDescendants, that computes all the maximal descendants of u and v at level $\ell_1 - \sigma$, where σ is the value used in the definition of strong separation (Section 2.1). It returns the cross product of the resulting nodes. Otherwise (if u and v are not homogeneously well-separated), it subdivides the node that is at the higher level of the tree and applies the procedure recursively to the children of this node.

Each well-separated pair is represented as $(\{x, y\}, \ell)$, where x and y are the nodes defining the well-separated pair, and $\ell = \text{msl}^*_s(x, y)$ is their maximum separation level. The initial call is findWSPD(r, r, s), where r is the root of the compressed quadtree, and s is the separation factor.

Algorithm 2 Construction of WSPD

Input: Two nodes u and v and a separation factor $s \geq 1$
Output: All s-well separated pairs between the subtrees of u and v

```
1: function findWSPD(u, v, s)
2:     if u = v and lev(u) = −∞ then return ∅
3:     if u and v are homogeneously 1-w.s. then
4:         ℓ1 ← msl1(u, v)
5:                         ▷ See findMaxLevel in Algorithm 1
6:         σ ← max( ⌈lg s/2⌉ , 0)
7:         X ← findDescendants(u, ℓ1 − σ)
8:         Y ← findDescendants(v, ℓ1 − σ)
9:         return ⋃_{x∈X, y∈Y} ({x, y}, ℓ1 − σ)
10:    if lev(u) ≥ lev(v) then
11:        return ⋃_i findWSPD(u_i, v, s)
12:    else return ⋃_i findWSPD(u, v_j, s)

13: function findDescendants(u, ℓ)
14:    if lev(u) ≤ ℓ then return {u}
15:    return ⋃_i findDescendants(u_i, ℓ)
```

We test whether two nodes u and v are homogeneously 1-well separated as follows. Let $\ell' = \max(\text{lev}(u), \text{lev}(v))$. Let u' and v' denote the respective containing boxes at this level. Then u and v are 1-well separated if and only if u' and v' are not adjacent. This can be determined in $O(1)$ time, using the same bit manipulations given in Lemma 1. Recall that each leaf is associated with a point, which we treat as a box at level $-\infty$. Therefore, two distinct leaves are always homogeneously 1-well separated.

We observe the following properties of the well-separated pairs generated by Algorithm 2. Recall that $\text{par}(u)$ denotes the parent of node u in the compressed quadtree.

LEMMA 4. *Consider any nodes r_1 and r_2 of the compressed quadtree (possibly $r_1 = r_2$). Consider any pair $\{x, y\}$ generated by the call findWSPD(r_1, r_2, s). Let $\{u, v\}$ be the pair of homogeneously 1-well separated nodes that resulted in the generation of the pair (x, y) at Line 9 of Algorithm 2. Then*

(i) $\text{lev}(u) \leq \text{msl}_1(u, v)$, *and*
if $u \neq r_1$, *then* $\text{msl}_1(u, v) < \text{lev}(\text{par}(u))$

(ii) $\text{lev}(x) \leq \text{msl}^*_s(x, y)$, *and*
if $x \neq r_1$, *then* $\text{msl}^*_s(x, y) < \text{lev}(\text{par}(x))$

(Symmetrical bounds hold for v and y.)

PROOF. First, consider (i). By the definition of homogeneous well separation, there exist u' and v' on level $\text{msl}_1(u, v)$ such that $u \subseteq u'$ and $v \subseteq v'$, and u' and v' are homogeneously 1-well separated. Therefore, $\text{lev}(u)$ is not greater than $\text{msl}_1(u, v)$, which establishes the first inequality. To prove the second inequality, suppose to the contrary that $u \neq r_1$ and $\text{msl}_1(u, v) \geq \text{lev}(\text{par}(u))$. This would imply that $\text{par}(u) \subseteq u'$, and so $\text{par}(u)$ and v would be homogeneously 1-well separated. By the nature of findWSPD, $\text{par}(u)$ would never have been recursively subdivided, which yields the desired contradiction.

To prove (ii) observe that, by the definition of findDescendants, x and y are maximal nodes on a level not more than $\text{msl}^*_s(x, y)$. Therefore, $\text{lev}(x) \leq \text{msl}^*_s(x, y) < \text{lev}(\text{par}(x))$. The analogous bounds for v and y hold by symmetry. □

The following lemma establishes the correctness of the algorithm.

LEMMA 5. *Consider a point set P, a separation factor $s \geq 1$, and a compressed quadtree T storing P. Letting r denote T's root, the call findWSPD(r, r, s) returns an s-WSPD for P. In particular, for any pair $\{x, y\}$ returned, resulting x and y are maximally s-well separated.*

PROOF. Consider any pair $\{x, y\}$ generated by the algorithm, and let $\{u, v\}$ be the pair of homogeneously 1-well separated nodes that resulted in the generation of this pair on Line 9. We have $\text{msl}^*_s(x, y) = \text{msl}_1(u, v) - \sigma$.

We first show that two boxes, $x^{\uparrow\sigma}$ and $y^{\uparrow\sigma}$ are homogeneously 1-well separated, and in particular, $\text{msl}_1(x^{\uparrow\sigma}, y^{\uparrow\sigma}) = \text{msl}_1(u, v)$. Let u' and v' be the boxes at level $\text{msl}_1(u, v)$ containing u and v, respectively. Note that $\text{lev}(x) \leq \text{msl}^*_s(x, y)$ and $\text{lev}(y) \leq \text{msl}^*_s(x, y)$, and $x \subseteq u \subseteq u'$ and $y \subseteq v \subseteq v'$. Therefore

$$\begin{aligned} \text{lev}\big(x^{\uparrow\sigma}\big) &= \text{lev}(x) + \sigma \\ &\leq \text{msl}^*_s(x, y) + \sigma \;=\; \text{msl}_1(u, v) \;=\; \text{lev}(u'), \\ \text{lev}\big(y^{\uparrow\sigma}\big) &= \text{lev}(y) + \sigma \\ &\leq \text{msl}^*_s(x, y) + \sigma \;=\; \text{msl}_1(u, v) \;=\; \text{lev}(v'). \end{aligned}$$

Clearly, $x^{\uparrow\sigma}$ and $y^{\uparrow\sigma}$ are contained within u' and v', respectively. Since $x^{\uparrow\sigma} \subseteq u'$, $y^{\uparrow\sigma} \subseteq v'$, and u' and v' are homogeneously 1-well separated, it follows that $x^{\uparrow\sigma}$ and $y^{\uparrow\sigma}$ are homogeneously 1-well separated. The maximum separation level of $x^{\uparrow\sigma}$ and $y^{\uparrow\sigma}$ is $\text{msl}_1(u, v)$.

Next, we show that $\{x, y\}$ is maximal. By Lemma 4(ii), $\text{par}(x)^{\uparrow\sigma}$ is on a level higher than $\text{msl}^*_s(x, y) + \sigma = \text{msl}_1(u, v)$, and therefore it is not homogeneously 1-well separated with any node containing y. The same applies to $\text{par}(y)^{\uparrow\sigma}$. Therefore the pair $\{x, y\}$ is maximal, as desired. □

The following lemma is implied by Lemmas 3 and 5.

LEMMA 6. *Given a separation factor $s \geq 1$, the number of pairs generated by findWSPD is $O(s^d n)$.*

The following lemma shows that, after constructing the compressed quadtree, the algorithm's running time is linear in the number of pairs generated.

LEMMA 7. *Given an n-element point set P in $\mathbb{R}^d$ stored in a compressed quadtree and $s \geq 1$, findWSPD computes an s-WSPD in time linear in the number of pairs.*

PROOF. Consider the computation tree whose nodes correspond the calls findWSPD. We say that a call is *terminal* if it makes no recursive calls to findWSPD, and we say that a terminal call is *effective* if it returns on Line 9. We will show that the total number of nodes in the resulting computation tree can be bounded by the number of effective terminal calls. To do this, we charge each noneffective terminal to the nonterminal call that generated it, that is, its parent in the computation tree. Recalling that each internal node in the compressed quadtree has at least two children, it follows that each nonterminal call generates at least two calls, neither of which is a noneffective terminal. In other words, if we prune all noneffective terminals from the computation tree, each internal node has at least two children. Thus, the total number of nodes in the computation tree is linear in the number of effective terminals.

Now consider the running time of effective terminal call. For a 1-well separated pair, $\{u, v\}$ on Line 3, it finds all descendants of u and v of level less than or equal to $\text{msl}_1(u, v) - \sigma$, where $\sigma = \max(\lceil \lg \frac{s}{2} \rceil, 0)$, by descending the tree (using recursive calls to findDescendants). It generates s-well separated pairs consisting of the cross product of these two sets of descendants. Clearly, the total running time of all the recursive calls to findDescendants is bounded by the total number of pairs it outputs. Thus, each effective terminal call runs in time linear in the resulting number of s-well-separated pairs. Therefore, the total running time to construct the WSPD is proportional to the final output size. □

4. UPDATES

In this section, we present the algorithms for inserting and deleting points into the WSPD. We first consider insertion in Section 4.1, deletion in Section 4.2, and we discuss maintenance of an internal data structure in Section 4.3.

4.1 Insertion

Consider the insertion of a new point p into the compressed quadtree. We first locate the node whose box contains the newly inserted point. By storing the compressed quadtree using an auxiliary structure, such as a topology tree [5], this can be done in $O(\log n)$ time. As observed by Fischer and Har-Peled [4], there are two cases depending on the location where the new point is inserted.

Case 1: (no new internal node) The new point p is stored in a leaf, denoted by x, that is hung directly beneath its parent (see Figure 8(a)).

Case 2: (between the cells of the compressed node w and its parent) We create new leaf, x containing p, and a new node z, which has w and x among its children (see Figure 8(b)).

After the insertion of p, we need to update the well-separated pairs. Our objective is that updated pairs will be the same as would result by running the static construction algorithm findWSPD on the updated point set. The insertion procedure is given in Algorithm 3.

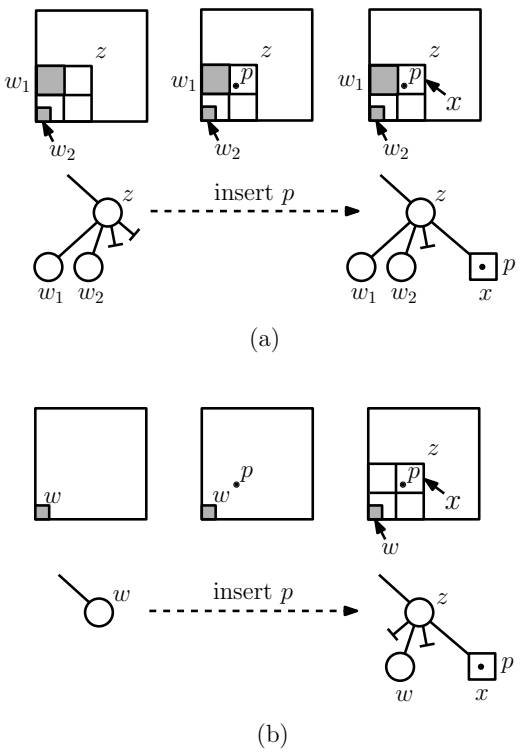

Figure 8: Inserting a point into the compressed quadtree.

First, let us consider Case 2 (Figure 8(b)) where the insertion resulted in the creation of node z. Note that z's cell contains the cell of x's sibling node w, which existed prior to the insertion. Recall that each well-separated pair is represented by a triple $(\{x, y\}, \ell)$, where x and y are the two nodes defining the pair, and ℓ is the maximum separation level. Consider any well-separated pair $(\{w, v\}, \ell)$ that existed prior to the insertion, where $\ell \geq \text{lev}(z)$. If findWSPD were to be run on the quadtree containing p, then because z will be encountered before w, and so it follows that (after insertion) this pair involves z rather than w. We replace each such pair with the new pair $(\{z, v\}, \ell)$. Note that since $\ell \geq \text{lev}(z)$, and ℓ is not affected by the insertion, ℓ is not changed.

Next, consider Case 1 (Figure 8(a)), where x denotes the leaf containing the new point. Consider any s-well separated pair $\{x, y\}$ generated by findWSPD on the updated point set. Since z and y are not strongly s-well separated (since otherwise the algorithm would have generated the pair $\{z, y\}$), we have $z^{\uparrow\sigma}$ and $y^{\uparrow\sigma}$ are not homogeneously 1-well separated. Let B denote the set of boxes at level $\text{lev}(z^{\uparrow\sigma})$ that are not homogeneously 1-well separated with respect to $z^{\uparrow\sigma}$. Clearly, $y^{\uparrow\sigma}$ is contained within some box of B.

To compute these new pairs, for each $b \in B$, we find the corresponding node in the compressed quadtree by invoking node(b). Recall that this returns the largest node in the compressed tree that is contained within b. (If the result is null, we simply ignore the result.) We determine the well-separated pairs involving the new leaf as follows. For each $b \in B$, we invoke findWSPD(x, node(b), s). We take the union of all the resulting pairs and add them to the WSPD.

Algorithm 3 Insertion of a point p for output sensitive WSPD

Input: A point, p and a separation factor, $s \geq 1$
Output: All s-well separated pairs related to new leaf including p

```
1: function insert(p, s)
2:     Insert p
3:     x : a new leaf including p
4:     z : parent of x
5:     if z is new then
6:         w : x's sibling
7:         ∀ s-WSPs ({w, v}, ℓ) with ℓ ≥ lev(z)),
8:             change ({w, v}, ℓ) to ({z, v}, ℓ).
9:     σ ← max(⌈lg s/2⌉, 0)
10:    B : set of boxes adjacent to z^{↑σ} including z^{↑σ}
11:    return ⋃_{b∈B} findWSPD(x, node(b), s)
```

The following lemma establishes the correctness this procedure. (We will present the running time later.)

LEMMA 8. *Function insert of Algorithm 3 correctly updates the s-WSPD.*

PROOF. Let T denote the compressed quadtree before p's insertion, and let T' denote the compressed quadtree after p's insertion. Let W be the set of well-separated pairs generated by findWSPD when run on T, and W' be the analogous set on T'. We will show that Algorithm 3 converts W into W'. We consider the same two cases mentioned above, and we use x, z, and w as defined above. There are only two types of pairs that would be affected by the insertion, those involving x itself and those involving the newly added internal node z.

First, let us consider the pairs involving z (Case 2). Consider any well-separated pair, $(\{z, y\}, \ell)$ in W', where $\ell \geq \text{lev}(z)$. Since w is contained within z, w is also well-separated with y. Therefore, $(\{w, y\}, \ell)$ exists in W. Conversely, consider $(\{w, y\}, \ell)$ in W, where $\ell \geq \text{lev}(z)$. If there exists a node whose level is less than or equal to ℓ, but higher than w, then findWSPD would have generated a well-separated pair involving it instead of w. Therefore, $(\{w, y\}, \ell)$ belongs as $(\{z, y\}, \ell)$ in W'. Line 8 in Algorithm 3 changes each such pair to $(\{z, y\}, \ell)$. (Note that the other well-separated pairs related to w whose maximum separation level is less than $\text{lev}(z)$ are related only with w, and they are included in both W and W'.)

Second, consider any well-separated pair, $(\{x, y\}, \ell)$ involving x in W' (Case 1). Such x and y were generated by invoking findDescendants (Line 7 and 8 of Algorithm 2) on nodes u and v of T' that are 1-well separated, respectively. By the nature of the algorithm, u and v are maximally homogeneously 1-well separated. Recall that B is the set of boxes adjacent to $z^{\uparrow\sigma}$ including $z^{\uparrow\sigma}$ in Line 10 of Algorithm 3. We will show that v is contained in some box $b \in B$ (see Figure 9). Assumming this for now, if we let v' be the largest node containing v at level not more than $\text{lev}(z^{\uparrow\sigma})$, for such a b, node(b) in Line 11 of Algorithm 3 will return v'. In the same line, findWSPD(x, v', s) will first find homogeneous 1-well separated pairs involving x and descendants of v', and then invoke findDescendants on each. We are interested in these calls involving ancestors of v. By the maximality of u and v, we assert that no proper ancestor of v will be homogeneously 1-well separated with respect to x. To see this, suppose that there existed a proper ancestor v, denoted by v'' that was homogeneously 1-well separated with x. By definition, a box x' on the same level as v'', containing x is 1-well separated with v''. Since v'' is an ancestor of v, by Lemma 4(i), $\text{lev}(u) \leq \text{msl}_1(u, v) < \text{lev}(\text{par}(v)) \leq \text{lev}(v'')$. This implies that $\text{lev}(u) < \text{lev}(x')$, and therefore, u is contained in x'. This yields that u and v'' are also homogeneously 1-well separated. This would contradict the fact that u and v are maximally well-separated. Therefore, x is not homogeneously 1-well separated with any proper ancestor of v. Thus, findWSPD(x, v', s) will find the 1-well separated pair $\{x, v\}$, and will generate the desired s-well separated pair $\{x, y\}$ by invoking findDescendants.

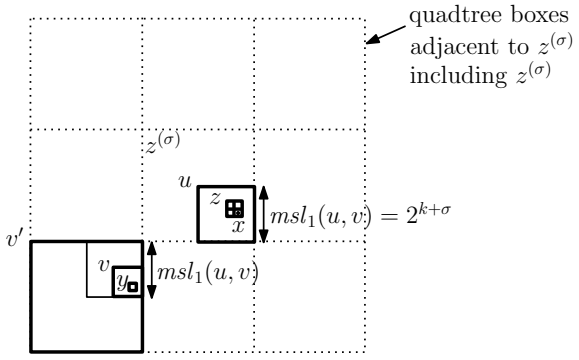

Figure 9: A well-separated pair of new leaf x

To complete the proof, it remains to show that v is contained in some box $b \in B$. To do this, it suffices to show that v is not homogeneously 1-well separated with $z^{\uparrow\sigma}$, and $\text{lev}(v) \leq \text{lev}(z^{\uparrow\sigma})$. By Lemma 4(ii), $\ell < \text{lev}(z)$. Thus,

$$\text{lev}(u) \leq \text{msl}_1(u, v) = \ell + \sigma < \text{lev}(z) + \sigma = \text{lev}(z^{\uparrow\sigma}).$$

By the maximality of u and v, v is not homogeneously 1-well separated with any box containing u on a higher level than $\text{msl}_1(u, v)$. Clearly, $z^{\uparrow\sigma}$ is such a box. Also, we have $\text{lev}(v) \leq \text{msl}_1(u, v) < \text{lev}(z^{\uparrow\sigma})$. This completes the proof. □

4.2 Deletion

Deletion essentially is the reverse of the insertion procedure, and is presented in Algorithm 4. Let x be the leaf containing the deleted point, p. If its parent, z has only two nodes, x and w, the node z will be deleted from the compressed quadtree. Thus, in a symmetrical manner with insertion, we associate the well-separated pairs with w, instead of z (see Line 5). Also, all well-separated pairs associated with x are deleted (see Line 7). The correctness proof is symmetrical with the proof for insertion and is omitted.

4.3 Maintenance of the WSPD for Updates

In order to analyze the running time of our update algorithms, we need to show how to maintain the s-well separated pairs and the maximum separation level associated with each. We store all the well-separated pairs in a hash table, denoted by H. Given a box u', let $W(u')$ denote the

Algorithm 4 Deletion of a point p for output sensitive WSPD

Input: A point, p and a separation factor, $s \geq 1$

```
1: function delete(p, s)
2:     Find a leaf x including p
3:     z : parent of x
4:     if z has two nodes, x and x's sibling, w then
5:         Change ({z, v}, ℓ) to ({w, v}, ℓ).
6:         Delete z
7:     Delete all well-separated pairs related to x
8:     Delete x
```

set of all well-separated pairs (u, v) such that u' is the box at level $\text{msl}_s^*(u, v)$ containing u. Note that a box may be uniquely identified by the Morton code (see [15]) of its lower left corner and its depth. We combine these to form the box's associated key for the hash table. For a box u', let $H(u')$ denote the hash table entry associated with u'. The entry, $H(u')$ stores a linked list of the elements of $W(u')$. The element of this linked list associated with (u, v) stores a cross reference to an element of the list in $H(v')$, where v' is the box at level $\text{msl}_s^*(u, v)$ containing v. Note that a well-separated pair (u, v) is stored as the pair (u', v'). In addition, we store a bit for each entry in the hash, which we call the *dirty bit*. If $H(u')$ is empty, this bit is set. This will be used for lazy deletion, which will be described below. In this way, the space of the hash table is proportional to the number of well-separated pairs.

For each node u, let L_u denote the set of levels ℓ, such that there exists a well-separated pair (u, v) so that $\text{msl}_s^*(u, v) = \ell$. We assume that L_u is stored as a bit vector indexed by levels. If there exists a well-separated pair involving u at level ℓ, bit ℓ of L_u is set. Through this bit vector, we can compute boxes u' associated with u in the manner presented above. Note that given u and such a level number ℓ, it is possible to compute u' and its associated hash key in $O(1)$ time. This key is then used to access the hash table. Recall that each point's coordinates are stored as fixed-point binary numbers, and bitwise operations can be performed on them in $O(1)$ time. We assume that each coordinate can be stored in a constant number of words, and hence the total space required for the coordinates is proportional to the number of nodes in the tree.

LEMMA 9. *Assume we have a model of computation that supports in $O(1)$ time the bitwise operations boolean-and ("$\wedge$"), boolean-or ("$\vee$"), complement ("$\overline{x}$"), left shift ("$\ll$"), and integer base-2 logarithm (that is, the index of the most significant bit) on point coordinates. Through the bitwise operations, the following operations for bit vectors can be performed in $O(1)$ time.*

- add(L_u, ℓ)*: Add bit ℓ to L_u.*
- remove(L_u, ℓ)*: Remove bit ℓ from L_u.*
- isMember(L_u, ℓ)*: Return true if L_u has bit ℓ set. Otherwise, return false.*
- split(L_u, ℓ, L_v)*: Split the bit vector L_u into two bit vectors, such that one contains bits whose positions are greater than or equal to ℓ, and the other contains those that are less than ℓ. The higher bits are stored in a new bit vector L_v, and the lower entries remain in L_u.*
- merge(L_u, L_v)*: Merge the bits of L_u to L_v.*
- getMembers(L_u)*: Enumerate all the bits of L_u.*

PROOF.

- add(L_u, ℓ): Set the bit at position ℓ by taking the bitwise-or of L_u and a bit string whose only set bit is at position ℓ.
- remove(L_u, ℓ): This is equivalent to setting bit ℓ of the complement of L_u, and returning the complement of the result.
- isMember(L_u, ℓ): Return true if the bitwise-and of L_u and a bit string whose only set bit is at position ℓ is not zero. Otherwise, return false.
- split(L_u, ℓ, L_v): We do this by first copying L_u to L_v. We then mask out all bits below level ℓ from L_v and mask out all bits at or above ℓ from L_u.
- merge(L_u, L_v): Merge two bit vectors L_u to L_v by taking their bitwise-or.
- getMembers(L_u): First copy L_u to the temporary bit string x. Output the integer base-2 logarithm of x, which is the index of x's most significant bit. Then mask this bit out. Repeat this until x has no more set bits.

The above operations, except the operation getMembers, can be performed in $O(1)$ time. The operation getMembers takes $O(|L_u|)$ time. Letting p denote the representative point of u, the size of a bit vector, L_u is less than the size of p's coordinates, and hence $O(|L_u|) = O(1)$. Thus, all of above operations can be performed in $O(1)$. □

Let us consider how to implement point insertion, deletion, and node queries.

Point Insertion:.

As mentioned above, the first step is to insert the new point p into the compressed quadtree. Following Fischer and Har-Peled [4] this can be performed in $O(\log n)$ time by representing the (possibly unbalanced) compressed quadtree as a topology tree [5].

To bound the time needed to update the WSPD, recall nodes x, z, and w from Algorithm 3. If x's parent, z, is new (Case 2), we update the well-separated pairs involving x's sibling, w. That is, each well-separated pair $(\{w, v\}, \ell)$, where $\ell \geq \text{lev}(z)$ is changed to $(\{z, v\}, \ell)$. We do this by performing split$(L_w, \text{lev}(z), L_z)$. This can be done in $O(1)$ time.

Next, we find all the new well-separated pairs involving the leaf x. We first find the set B of boxes at level $\text{lev}(z^{\uparrow\sigma})$ that are not homogeneously 1-well separated with respect to $z^{\uparrow\sigma}$ (see Line 10 of Algorithm 3). Observe that because we assume the L_∞ metric, B consists of boxes of side length equal to the side length of $z^{\uparrow\sigma}$. There are at most 3^d such boxes, and they can be computed in $O(1)$ time. For each $b \in B$, we invoke node(b) to compute the highest level node of the compressed quadtree contained within b (see Line 11). This query can be performed in $O(\log n)$ time through the use of the topology tree. Then, we invoke findWSPD$(x, \text{node}(b), s)$. For each newly generated

pair $\{x, y\}$ resulting from findWSPD, we first update both L_x and L_y by $\text{add}(L_x, \text{msl}_s^*(x, y))$, and $\text{add}(L_y, \text{msl}_s^*(x, y))$. These take $O(1)$ time. Let x' and y' be boxes on level $\text{msl}_s^*(x, y)$ containing x and y, respectively. We add elements referencing this well-separated pair into both $H(x')$ and $H(y')$. The dirty bits of both $H(x')$ and $H(y')$ are then reset. This can also be performed in $O(1)$ time. Thus, including the $O(\log n)$ time to insert the point, the addition of all new well-separated pairs involving x can be performed in worst-case time $O(\log n + m)$, where m is the number of newly generated pairs.

Point Deletion:.

Next, let us consider the deletion of a point p. We first find the leaf x containing p (see Line 2 of Algorithm 4). This can be performed in $O(\log n)$ time, again using the topology tree. Recall z and w from the algorithm. If z would have only one child w after the deletion of x, z will be also deleted. We then merge all the well-separated pairs involving z to w by $\text{merge}(L_z, L_w)$ in $O(1)$ time.

Next, we remove all well-separated pairs involving x. First, we get all bits of L_x that are set by $\text{getMembers}(L_x)$. This takes $O(1)$ time. For each bit ℓ of L_x that is set, let x' denote the box of level ℓ containing x. We access the linked list in $H(x')$. If the dirty bit of $H(x')$ is not set, that is, $H(x')$ has a nonempty list, we remove all the elements of the list. For each associated pair (x', y'), we also remove the corresponding element from $H(y')$. If $H(y')$ is now empty, we set its dirty bit. If we attempt to access $H(y')$ as part of a future operation, it is because the corresponding bit of L_y was set, for some node y. Thus, whenever we access any entry $H(y')$, if its dirty bit is set, we reset the corresponding bit of L_y as well as the dirty bit. By charging this to the earlier deletion that set the dirty bit, we can easily see that this takes the $O(1)$ amortized time. After removing all the elements of the list in $H(x')$, we remove ℓ from L_x by $\text{remove}(L_x, \ell)$ in $O(1)$ time. If the dirty bit of $H(x')$ is set, we remove the correponding bit ℓ from L_x, and reset the dirty bit like above in $O(1)$ amortized time. Combining this with the initial $O(\log n)$ time to remove the point, deletion can be performed in amortized time $O(\log n + m)$, where m is the number of deleted pairs. Combining these results, we have Theorem 1.

5. CONCLUSIONS

Well-separated pair decompositions have a number of applications in the processing of spatial data. Efficient maintenance of WSPDs under point insertions and deletions is an algorithmic problem of fundamental importance. Because the number of pairs can be significantly lower than worse-case bounds (especially when points are well clustered), it is desirable to consider an output sensitive approach. We have presented the first output-sensitive algorithm for maintaining the WSPD in Euclidean space. Except for an overhead of $O(\log n)$, our algorithms are linear in the number of created/removed pairs. The focus of this work is on establishing the correctness and efficiency of our algorithms theoretically. In future work, we will develop an implementation in order to establish the algorithms' practical efficiency.

6. REFERENCES

[1] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17:135–163, 2001.

[2] 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. Assoc. Comput. Mach.*, 42:67–90, 1995.

[3] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983.

[4] J. Fischer and S. Har-Peled. Dynamic well-separated pair decomposition made easy. *Proc. 17th Canad. Conf. Comput. Geom.*, pages 235–238, 2005.

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

[6] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and applications. *Comput. Geom. Theory Appl.*, 35(1):2–19, 2006.

[7] L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th Annu. European Sympos. Algorithms*, volume 5193/2008, pages 478–489. Springer, 2008.

[8] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–ï£¡348, 1987.

[9] S. Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, Providence, Rhode Island, 2011.

[10] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics and their applications. *SIAM J. Comput.*, 35:1148–ï£¡1184, 2006.

[11] D. M. Mount and E. Park. A dynamic data structure for approximate range searching. In *Proc. 26th Annu. Sympos. Comput. Geom.*, pages 247–256, 2010.

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

[13] G. Narasimhan and M. Zachariasen. Geometric minimum spanning trees via well-separated pair decompositions. *J. Exper. Algorithmics*, 6, 2001.

[14] L. Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62:1073–1087, 2012.

[15] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan-Kaufmann, 2006.

[16] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *IEEE Trans. on Knowledge and Data Engr.*, 22:1158–1175, 2010.

[17] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Sys. Sci.*, 26:362–391, 1983.

[18] M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*, pages 877–935. Elsevier Science, Amsterdam, 2000.

[19] M. Smid. The well-separated pair decomposition and its applications. In T. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 53–1–53–12. Chapman & Hall/CRC, Boca Raton, 2007.