Fast Heuristic Algorithms Using Well-separated Pair Decompositions

(Extended Abstract)

Jianlin Zhu * Giri Narasimhan †

June 27, 1998

Abstract

Callahan and Kosaraju [2] proposed a well-separated pair decomposition for points (WSPD) in d-dimensional Euclidean space. We extend their ideas to introduce a well-separated pair decomposition for bichromatic point sets (WSPDB) in d-dimensional Euclidean space. We show how to compute such a decomposition of small size ($O(n)$ pairs of sets of points) in $O(n \log n)$ time.

We use the new decomposition (WSPDB) to develop fast heuristics for the *Bichromatic Closest Pair* problem, which is described as follows: given a set of n points in d-dimensional Euclidean space, each of which is colored either red or blue, determine the closest pair of points such that one is colored red and the other blue. The bichromatic closest pair problem is important because an efficient algorithm for it can be used to design fast algorithms for the *Euclidean Minimum Spanning Tree* problem, and many others.

Experimental results indicate that our heuristic algorithms for the bichromatic closest pair problem and the Euclidean minimum spanning tree problem are, in practice, very fast.

*`zhuj@ceri.memphis.edu`; Dept. of Math. Sciences, The University of Memphis, Memphis, TN 38152.

†`giri@msci.memphis.edu`; Dept. of Math. Sciences, The University of Memphis, Memphis, TN 38152; Partially funded by NSF grant CCR-940-9752 and by a grant from Cadence Design Systems, Inc.

1 Introduction

The *Minimum Spanning Tree* problem is a well-known and important problem. A *Euclidean minimum spanning tree* (EMST) of a point set in $\Re^d$ is simply a minimum spanning tree of the complete Euclidean graph on the point set. Since there are $O(n^2)$ edges in the complete Euclidean graph on n points, any algorithm for the Euclidean minimum spanning tree problem that explicitly constructs this graph is doomed to be inefficient. The Euclidean minimum spanning tree problem has been extensively studied. In the case of two dimensions, a Euclidean minimum spanning tree can be computed efficiently by using the Delaunay triangulation. Unfortunately, it becomes inefficient to use the Delaunay triangulation for higher dimensions ($d > 2$), since the Delaunay triangulation itself could have $O(n^2)$ edges even for points in $\Re^3$. Yao [6] presented an algorithm with complexity $O((n \log n)^{1.8})$ for points in 3-dimensional Euclidean space.

Agarwal et al. [1] showed that Euclidean minimum spanning tree can be reduced to the *bichromatic closest pair* (BCP) problem, which can be stated as follows: given a set P of n points in $\Re^d$, each of which is colored red or blue, find a red point and a blue point such that the distance between the two points is minimum among all red-blue pairs. The naive method takes $O(n^2)$ time. Agarwal et al. [1] showed an algorithm to compute a Euclidean minimum spanning tree of a set of n points in $\Re^d$ in time $O(T_d(n,n) \log^d n)$, where $T_d(n,m)$ is the time required to compute a bichromatic closest pair between n red and m blue points. If $T_d(n,n) = \Omega(n^{1+\varepsilon})$, for some fixed $\varepsilon > 0$, then the running time improves to $O(T_d(n,n))$. Furthermore, they described a randomized algorithm to compute a bichromatic closest pair in expected time $O((nm \log n \log m)^{2/3} + m \log^2 n + n \log^2 m)$ in 3-dimensional space, which leads an $O((n \log n)^{4/3})$ expected time algorithm for computing a Euclidean minimum spanning tree of n points in 3-dimensional space.

Callahan and Kosaraju used a data structure called the *Well-separated pair decomposition* [3] to design an algorithm for the Euclidean minimum spanning tree problem [2] that runs in time $O(T_d(n,n) \log n)$. They, too used a reduction to the bichromatic closest pair problem. Krznaric et al. [4] showed that a Euclidean minimum spanning tree on n points in $\Re^d$ can be computed in optimal $O(T_d(n,n))$ time under any fixed L_t-metric. For $d = 3$, it is shown that a Euclidean minimum spanning tree can be found in optimal $O(n \log n)$ time under the L_1 and L_∞-metric.

The well-separated pair decomposition (WSPD) of Callahan and Kosaraju [3] is defined as follows: For a set P of n points in $\Re^d$, a well-separated pair decomposition is a set of pairs of subsets of P,

$$\{A_1, B_1\}, \ldots, \{A_k, B_k\},$$

such that every pair of points in P is represented in exactly one of the pairs and every pair of sets is well-separated. Intuitively, a pair of subsets of P is well-separated if the enclosing balls for each of the two sets is small as compared to the distance between them. They showed that for a set P of n points in $\Re^d$, a well-separated pair decomposition of P with $O(n)$ pairs can be constructed in $O(n \log n)$ time. The most significant advantage that the well-separated pair decomposition has over the *Delaunay triangulation* is that its size is independent of dimension, and is hence better suited for higher-dimensional geometric problems.

The well-separated pair decomposition is a way of representing all pairs of points. However, the bichromatic closest pair only needs a restricted decomposition. We introduce a variant of the well-separated pair decomposition, which we refer to as a well-separated pair decomposition for a bichromatic point set (WSPDB) and, which provides a way of representing precisely all the bichromatic pairs of points. For a set P of n bichromatic points in $\Re^d$, a WSPDB is a set of pairs of subsets of P

$$\{A_1, B_1\}, \ldots, \{A_k, B_k\},$$

such that every bichromatic pair of points in P is represented in exactly one of the pairs, and every pair $\{A_i, B_i\}$ of the decomposition is such that A_i is a subset of the red points in P, B_i is a subset of the blue points in P, and the sets A_i and B_i are well-separated. Like a WSPD, a WSPDB can also be computed in $O(n \log n)$ time and is of size $O(n)$.

We develop an efficient heuristic algorithm for the bichromatic closest pair problem using a WSPDB. Although the theoretical upper bound of the algorithm is $O(n^2)$ in the worst case, experimental results show that it works every efficiently in practice. We also develop an efficient algorithm for the Euclidean minimum spanning tree problem by using Callahan and Kosaraju's [2] reduction to bichromatic closest pair and by plugging in our efficient heuristic algorithm for bichromatic closest pair.

In section 2 of this paper, we briefly review the well-separated pair decomposition of Callahan and Kosaraju. Then, we introduce the well-separated pair decomposition for bichromatic point sets. An efficient algorithm for bichromatic closest pair using the WSPDB follows in section 4. Finally, we present the algorithm for the Euclidean minimum spanning tree problem using a WSPD and our algorithm of bichromatic closest pair. In each of the cases, we describe experimental results. All our implementations were written in C++ on a SGI Indy Workstation. Our implementations were greatly facilitated by the use of the *Library of Efficient Data Structures* (LEDA) [5].

2 The Well-Separated Pair Decomposition

We quickly review some of the terms and concepts of the well-separated pair decomposition.

2.1 Definitions and notations

In this paper, we mostly follow the notation from Callahan and Kosaraju [3]. However, some terms are redefined for completeness and clarity.

Let P be a set of points in $\Re^d$. The *bounding rectangle* of P is defined as the smallest rectangle that encloses all points in P. The *bounding circle* of P is defined as the smallest circle that encloses the bounding rectangle. Here, the "rectangle" and "circle" actually mean the d-dimensional rectangle and ball in $\Re^d$. Refer to Figure 1 for the bounding rectangle and bounding circle of a point set in a plane.

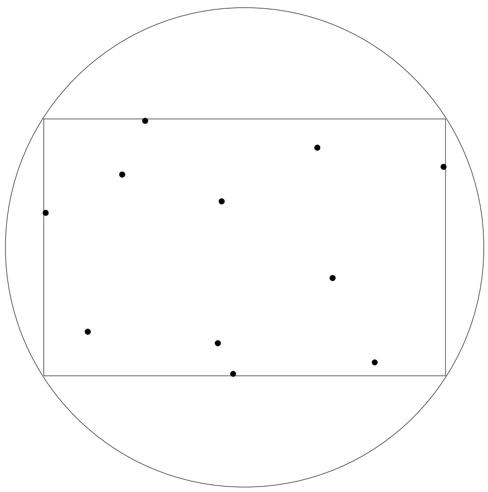

Figure 1: The bounding rectangle and bounding circle of a point set in plane

We use $d(p,q)$ to denote the Euclidean distance between the two points p and q. Given two point sets A and B, and the corresponding bounding circles with centers o_1 and o_2, respectively, and radii r_1 and r_2 respectively. We use $d(A,B)$ denoting the distance of the two point sets and is defined as the distance between the two bounding circles, i.e.,

$$d(A,B) = d(o_1,o_2) - r_1 - r_2.$$

Two point sets, A and B, are said to be *well-separated* with separation factor s if $d(A,B) \geq s \cdot max(r_1,r_2)$. This definition is a little different from the original one by Callahan and Kosaraju where the two point sets A and B are said to be well-separated with separation factor s if $d(o_1,o_2) - 2 \cdot max(r_1,r_2) \geq s \cdot max(r_1,r_2)$. Clearly, this definition gives rise to a tighter definition of well-separatedness, and does not affect the correctness of their algorithms for Euclidean minimum spanning tree as well as the other applications of well-separated pair decompositions. While this can be rigorously proved, here we omit the straightforward proof.

Definition 1 *Given a set P of n points in $\Re^d$ and a real number $s \geq 0$, a* Well-Separated Pair Decomposition (WSPD) *is a sequence of pairs of subsets of P*

$$\{A_1,B_1\},\ldots,\{A_k,B_k\},$$

such that:

1. *Every pair of points $\{p,q\}$ is in exactly one pair of the decomposition, and*
2. *For each $i = 1,\ldots,k$, the sets A_i and B_i are well-separated with separation factor s.*

A well-separated pair decomposition can be computed in $O(n \log n)$ time with size $k = O(n)$.

The decomposition is achieved by constructing a binary tree T associated with P. This tree is called a *fair split tree*. Every node of T corresponds to a subset of P. The root node corresponds to the whole set P and each internal node corresponds to a subset of P. The tree T has n leaf nodes, each of which corresponds to a singleton set with a unique point of P. Each of the subsets, A_i and B_i, in the decomposition corresponds to a node in T.

2.2 Computing the fair split tree

The root of the fair split tree T corresponds to the point set P. The two children of the root node correspond to the two subsets created by cutting the bounding rectangle of P at the middle along the longest dimension. The subsets are recursively split until the subset is a single point. The resulting tree is a binary tree, whose root corresponds to the point set P, each leaf node corresponds to a single point in P, and each internal node corresponds to some subset of P. Each node in T maintains the center and radius of the bounding circle of the corresponding subset. The expected height of the fair split tree T is $O(\log n)$. However, the height of T can be $O(n)$ in the worst case. Callahan and Kosaraju [3] showed that the fair split tree T can be constructed in worst-case time $O(n \log n)$.

2.3 Computing a WSPD

Given a set of points P and the corresponding fair split tree T, the following computation is performed for every internal node of T, each of which outputs one or more well-separated pairs of the decomposition. For the internal node v of T, the following computation is performed on the two subsets A and B corresponding to the two children of v (say, v_A and v_B): If A and B are well-separated, then $\{A, B\}$ is output as a well-separated pair of the decomposition; If A and B are not well-separated, and assuming that A has a bounding circle that is no smaller than that of B, the computation continues by recursively performing the same computation on the two pairs of subsets A_1, B and A_2, B, where A_1 and A_2 are the two subsets corresponding to the children of v_A. The union of all the well-separated pairs that are output constitutes a well-separated pair decomposition of P. The well-separated pairs can be computed in $O(n \log n)$ time and is of size $O(n)$.

2.4 Experimental results

As mentioned earlier all our experiments were run on a SGI Indy workstation and they were run with points generated uniformly at random from a unit square. All our programs were coded in C++. Callahan and Kosaraju [2] have shown that the number of well-separated pairs output by the algorithm is $O(n)$. A more careful analysis shows this number to be $O(s^d n)$, where s is the separation constant and d is the dimension of the space. We ran the program for different values of s, while fixing n at 10,000, and have tabulated the constant of proportionality. Table 1 confirms the non-linear dependence on s (for a fixed n) even in practice. Note that p refers to the number of well-separated pairs generated by the algorithm. Running times for this algorithm are discussed in section 3.2.

s	2	4	6	8	10	12	14	16	18	20
p/n	9.5	21.8	38.5	59.2	83.3	110	140	172	206	242

Table 1:

3 Well-Separated Pair Decompositions for Bichromatic Point Sets

We now introduce a variant of the well-separated pair decomposition, which we will refer to as the well-separated pair decomposition for bichromatic point sets (WSPDB).

Definition 2 *Given a bichromatic point set P of n points in $\Re^d$, a* Well-Separated Pair Decomposition for Bichromatic Point Sets (WSPDB) *is a sequence of pairs of subsets of P,*

$$\{A_1, B_1\}, \ldots, \{A_k, B_k\},$$

such that:

1. *every red-blue pair of points $\{p, q\}$ is in exactly one pair of the decomposition.*
2. *For each $i = 1, \ldots, k$, the set A_i (B_i, resp.) is a subset of the red (blue, resp.) points of P, and the sets A_i and B_i are well-separated with separation factor s.*

As shown below we can construct such a decomposition in $O(n \log n)$ time with $O(n)$ pairs in the decomposition.

3.1 Computing a WSPDB

There is a naive way to construct such a decomposition of size $O(n)$. Find a well-separated pair decomposition of the union of red and blue points using the method of Callahan and Kosaraju [3]. Then for each pair $\{X, Y\}$ of the decomposition, construct the 2 pairs $\{A_X, B_Y\}$ and $\{B_X, A_Y\}$, where A_X (B_X, A_Y, B_Y, resp.) are the red (blue, red, blue, resp.) points from X (X, Y, Y, resp.). Thus, the decomposition for bichromatic point sets has at most twice the size of a WSPD of the union of red and blue points.

We show another approach to compute a well-separated pair decomposition for bichromatic point sets. This approach has the same worst-case time complexity as the naive method described above. However, it is likely to be more efficient in cases where red and blue points appear in clusters. Let P be a bichromatic set of n points in $\Re^d$. Let A be the subset of all red points in P and B the subset of all blue points in P. We construct two fair split tree T_A and T_B of A and B respectively. The WSPDB of P is constructed using T_A and T_B. For the well-separated pairs $\{A_i, B_i\}$ output by the algorithm, A_i is a node in T_A and B_i is a node in T_B. The well-separated pairs are found by performing the following computation. We start from the roots of T_A and T_B and consider the two sets A and B corresponding to these nodes. If A and B are well-separated, $\{A, B\}$ is output as one well-separated pair of the decomposition. If A and B are not well-separated, and assuming that A has a bounding circle no smaller than that of B, we consider that pairs of subsets A_1, B and A_2, B recursively, where A_1 and A_2 are the sets corresponding to the two children of the node corresponding to A.

Callahan and Kosaraju [3] showed that the number of well-separated pairs in a regular (not bichromatic) decomposition is $O(n)$. They defined a natural ordering on the sets (based on the sizes of the bounding boxes) so that every pair of the decomposition can be thought of as an ordered pair of sets. Using a simple packing argument they showed that the number of ordered pairs that a set can be involved in is a constant (dependent only on s and d). The arguments can be easily extended to prove that the size of a WSPDB is also $O(n)$. Thus, a well-separated pair decomposition for bichromatic points can be constructed in $O(n \log n)$ time and is of size $O(n)$.

3.2 Experimental results

Experiments similar to that for the well-separated pair decompositions were performed for the well-separated pair decompositions for bichromatic points. The size of the decomposition is $O(n)$. Our experiments (for uniformly distributed points) show that the constant hidden in the $O(n)$ is smaller than the corresponding constant for producing a WSPD. The CPU time for constructing a well-separated pair decomposition for bichromatic points is also better than that for the regular WSPD. Table 2 compares the running times and the values of p/n as n is varied (two different values of s were studied).

		$s = 2$					$s = 10$				
	$n/10,000$	2	4	6	8	10	2	4	6	8	10
WSPD	p/n	19.4	39.5	59.7	79.6	100	180	375	574	776	980
	CPU Time (secs)	7.3	15.4	23.7	32.1	40.9	20.1	42.3	64.9	87.0	109.3
WSPDB	p/n	19.2	38.8	58.9	79.1	99.0	169	357	551	752	952
	CPU Time (secs)	6.28	13.5	20.7	28.3	36.0	18.2	38.9	60.1	82.2	103.8

Table 2:

4 Bichromatic Closest Pairs

Before considering the general bichromatic closest pair problem, we consider a special case where the red points and blue points are well-separated. For the general case when the red points and blue points may not be well-separated, a well-separated pair decomposition for the bichromatic point sets is first computed and the red points and blue points are well-separated in each pair of the WSPDB.

4.1 The closest pair between two well-separated point sets

We consider a special case of the bichromatic closest pair problem where the set of red points A and the set of blue points B are already well-separated. This case is important since this is what is really needed as a subroutine for the minimum spanning tree algorithm, as proposed by Callahan and Kosaraju [2]. In this case, the WSPDB of the bichromatic point set $A \cup B$ would have just one well-separated pair $\{A, B\}$.

The algorithm WS-BCP consists of two phases. In the first phase, the function ESTIMATE is used to quickly find the distance between an approximate bichromatic closest pair between A and B. This estimate acts as an upper bound, which is then used in the second phase by the function EXACT-BCP to prune the search.

We assume that we are given two fair split trees T_A and T_B for the two point sets A and B; if they are not given they can be computed as described in Section 2. First we introduce some notation that we need. If v_X is any internal node in T_A or T_B with a corresponding point set X, then its two children v_{X_1} and v_{X_2} correspond to the two sets X_1 and X_2 obtained by splitting X. We will loosely refer to the sets X_1 and X_2 as the "children" of set X. Also, for two internal nodes v_X and v_Y in T_A (or T_B), we say that X is "larger" than Y if the radius of the bounding circle of X is at least as large as that of Y. We also say that a set X is "closer" to set Y than to set Z if $d(X, Y) \leq d(X, Z)$, i.e., the corresponding bounding circles are closer.

For the ESTIMATE function, we start from the roots of T_A and T_B. Assume that A is larger than B and that A_1 and A_2 are the two children of A. If B is closer to A_1 than to A_2, then the search for the approximate pair is recursively performed for the pair (A_1, B). Otherwise, it is recursively performed for the pair (A_2, B). The recursion is stopped when both sets A and B are singleton sets, in which case the distance between the pair of points is returned. The complexity of this phase is proportional to the height of the fair split tree since only one branch of the tree is pursued for each of the sets A and B. As mentioned earlier, the expected height of the fair split tree is $O(\log n)$. Note that the distances between bounding circles can be computed in constant time since each node in the fair split tree maintains the center and the radius of the bounding circle of the corresponding point sets.

As with ESTIMATE, the EXACT-BCP starts from the root nodes of the trees T_A and T_B. If A has a larger bounding circle than B, then the search is recursively performed on both the pairs (A_1, B) and (A_2, B). In each of the two cases, if their bounding circles are farther from each other than the threshold estimated in the first phase or the distance between the closest pair seen so far, then the pair is ignored. Note that the estimated threshold is refined any time the algorithm encounters a pair of points closer than the closest pair previously seen. The recursion is terminated when the pair of sets of points are both singleton sets, in which case the pair of points is considered as a potential candidate for the exact bichromatic closest pair. The algorithm is given in Figure 2.

The worst-case time complexity of constructing the fair split tree T_A (T_B, resp.) is $O(|A| \log |A|)$ ($O(|B| \log |B|)$, resp.). The worst-case time complexity of ESTIMATE is proportional to the height of the fair split trees, whose expected value is proportional to the logarithm of the sizes of the

```
WS-BCP(A, B)
   l = ESTIMATE(A, B)
   EXACT-BCP(A, B, a, b, l)
   return (a, b) as closest red-blue pair

ESTIMATE(A, B)
   Comment: Assume that A is larger than B.
   if (|A| = |B| = 1) then return d(A, B)
   Let A_1 and A_2 be the two children of A
   if (d(A_1, B) < d(A_2, B)) then
            return ESTIMATE(B, A_1)
   else     return ESTIMATE(B, A_2)

EXACT-BCP(A, B, a, b, l)
   Comment: Assume that A is larger than B.
   Comment: (a, b) is the closest red-blue pair seen so far.
   if (|A| = |B| = 1) and (d(A, B) < l) then
            (a, b) = {A, B}
            l = d(A, B)
            return
   if (d(A_1, B) < l) then
            EXACT-BCP(B, A_1, a, b, l)
   if (d(A_2, B) < l) then
            EXACT-BCP(B, A_2, a, b, l)
   return
```

Figure 2: Algorithm WS-BCP

sets, and could be as bad as a linear function in the sizes of the two sets. The worst-case time complexity of EXACT-BCP, and of WS-BCP, is $O(|A| \cdot |B|)$. In practice, however, the running time of WS-BCP is dependent on the quality of the value returned by ESTIMATE.

4.2 Bichromatic closest pairs for general point sets

In this section, we present an efficient algorithm for the general bichromatic closest pair problem where the red points and blue points are not well-separated. The obvious way to compute it is to compute a WSPDB of the union of the red and blue points and then to invoke WS-BCP on each well-separated pair. We make minor improvements to this basic algorithm.

As in the WS-BCP algorithm, we estimate the distance between the closest bichromatic pair and then use it to prune the search. Let P be a bichromatic set of points in $\Re^d$. Let A denote the subset of all red points and B denote the subset of all blue points. Let T_A and T_B represent the fair split trees of A and B, respectively. A WSPDB of P using T_A, T_B gives a sequence of well-separated pairs

$$\{A_1, B_1\}, \ldots, \{A_k, B_k\},$$

where $A_i \subseteq A$ with a corresponding node in T_A and $B_i \subseteq B$ with a corresponding node in T_B.

The estimate for the distance between the closest bichromatic pair is found by computing l =WS-BCP(A_m, B_m), where $d(A_m, B_m) \leq d(A_i, B_i)$, for $i = 1, \ldots, m$, i.e., $\{A_m, B_m\}$ is the pair in the WSPDB with the smallest distance between the corresponding bounding circles. Clearly, this can be computed in $O(k) = O(n)$ time. Note that if A_m and B_m are singleton sets, we have already found the exact closest red-blue pair. Finding the estimated distance, l, between the closest bichromatic pair helps to prune the search since we no longer need to perform WS-BCP on pairs $\{\{A_i, B_i\}$ for which $d(A_i, B_i) \geq l$. If $d(A_i, B_i) < l$, then WS-BCP(A_i, B_i) is invoked. The value l is also updated any time a closer bichromatic pair is found. The algorithm of bichromatic closest pair is given in Figure 3.

BCP(A, B)
    Let $\{A_1, B_1\}, \ldots, \{A_k, B_k\}$ be a WSPDB of (A, B)
    Let (p, q) =WS-BCP(A_m, B_m), where $d(A_m, B_m) \leq d(A_i, B_i)$, for $i = 1, \ldots, m$,
    if $(|A_m| = |B_m| = 1)$ **then**
      return (p, q) as the closest red-blue pair
    else
      $l = d(p, q)$
      for $i = 1$ **to** k **do**
          if $(d(A_i, B_i) < l)$
            (x, y) =WS-BCP(A_i, B_i)
            if $(d(x, y) < d(p, q))$ **then**
              $\{p, q\} = \{x, y\}$
              $l = d(p, q)$
      return (p, q)
end BCP

Figure 3: Algorithm BCP

4.3 Experimental results

The worst-case time complexity for algorithm BCP is $O(|A| \cdot |B|)$. Since the expected height of the fair split tree is proportional to the logarithm of the sizes of the two sets, we conjecture that the expected time complexity of BCP is $O((|A| + |B|) \log(|A| + |B|))$.

Experimental results show this algorithm to be extremely practical. Figure 4 shows the CPU time to compute a bichromatic closest pair of a point set P of n points in the plane. In all our tests, the number of red and blue points were the same. In the graph, the solid line is the time for computing a WSPDB and the dashed line is the time to find the closest red-blue pair after computing the WSPDB. The graph shows that the bulk of the time is spent on computing the well-separated pair decomposition, which is known to have a worst-case running time of $O(|A| \log(|A|) + |B| \log(|B|))$.

One note about the value of separation factor s. The algorithm is correct for any value of s, but the running time depends on it. A larger value of s will need more time for the decomposition, and less time for the subsequent search (since fewer pairs will be present in the decomposition). A smaller value of s has the effect of considerably speeding up the computation of the well-separated pair decomposition. In our experiments, we tried different values of s in the range -2.0 to 10.0. The running times are tabulated in Table 3 for different values of s. For this set of experiments n,

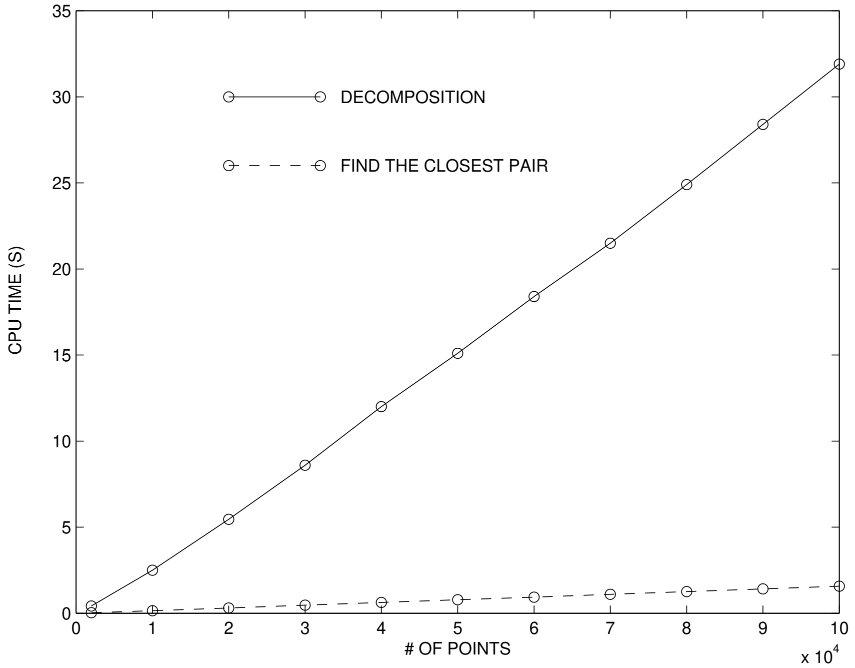

Figure 4: Running times for algorithm BCP

the number of points, was set at 40,000. The number of red and blue points were 20,000 each. The running time is broken into two parts T_1, which is the time in seconds to compute a well-separated pair decomposition for the bichromatic set of points, and T_2, which is the time to compute the closest bichromatic pair after the decomposition. In fact, T_1 can also be thought of as composed of two parts, T_1' and T_1''. T_1' is the time to build a fair split tree and this is independent of the value of s. $T_1' = 10.03$ seconds and is exactly equal to T_1 for $s = -2.0$. T_1'' is the time to compute the well-separated pairs after building the fair-split tree and this is what causes the differences in T_1 values for different s. Time T_2 decreases as s decreases until $s = 0$ after which it seems to rise again. What we observed was that for all values of $s > 0$, it always stopped after the first if-statement. In other words, it always a found the closest pair after only inspecting the values $d(A_i, B_i)$, for $i = 1, \ldots, m$, since $|A_m| = |B_m| = 1$ turned out to be true. Thus, for $s > 0$, time T_2 was directly proportional to the number of well-separated pairs. On the other hand, for $s \leq 0$, it had to compute WS-BCP for several well-separated pairs, and hence time T_2 starts to increase again as s becomes more negative. For $s = -2.0$, there is only one well-separated pair, since any two sets are well-separated with separation constant $s = -2.0$. The experiments suggest that the optimal value of s to use for this algorithm is some positive value between 0 and 0.5.

s	10.0	4.0	2.0	1.0	0.5	0.3	0.1	0.0	-0.2	-0.5	-1.0	-2.0
T_1	38.6	17.2	13.09	11.66	11.24	11.08	10.74	10.75	10.51	10.35	10.17	10.03
T_2	13.7	3.53	1.53	0.83	0.58	0.48	0.37	0.61	0.55	0.58	0.56	1.07

Table 3:

5 The Euclidean Minimum Spanning Tree

Callahan and Kosaraju [2] proposed the following simple algorithm to compute the Euclidean minimum spanning tree. Let P be a set of n points in $\Re^d$. Construct a well-separated pair decomposition of P with separation $s \geq 2.0$, giving a sequence of well-separated pairs $\{\{A_i, B_i\}, i = 1, \ldots, k\}$. For every pair $\{A_i, B_i\}$ in the decomposition, add the shortest edge between the point sets A_i and B_i to the graph. This produces a sparse graph. Now compute a minimum spanning tree in this sparse graph using any known algorithm for computing minimum spanning trees in weighted graphs.

By a result of Agarwal et al. [1] it is easy to see that if $\{A_i, B_i\}$is a well-separated pair (for $A_i, B_i \in P$ and $s > 2.0$), then any Euclidean minimum spanning tree of P contains at most one edge between A_i and B_i. Secondly, if there is one edge between A_i and B_i in a Euclidean minimum spanning tree, this edge must be the shortest edge between A_i and B_i. This is proved by constradiction. Suppose the edge e is not the shortest one, then its removal splits the Euclidean minimum spanning tree into two connected components. There are two cases here. In the first case, all the points from A_i lie in one connected component piece and all the points from B_i lie in the other. Then, we can use the shortest edge between A_i and B_i to reconnect the two pieces again, resulting in a tree of smaller weight (contradiction). In the second case, we assume without loss of generality that A_i is not entirely contained in one component. Say A_i is split into A_i^1 and A_i^2, and A_i^1 is in one component and A_i^2 is in the other. In this case, we can choose any edge between A_i^1 and A_i^2 and it would be shorter than the original edge e. So, once again adding an edge between A_i^1 and A_i^2 would result in a tree of smaller weight, resulting in a contradiction. Hence the sparse graph ($k = O(n)$ edges) constructed above includes a Euclidean minimum spanning tree of P and can be constructed using our fast heursitics.

Our main modification is to use the WS-BCP algorithm from section 3 to find the closest pair (i.e., shortest edge) between point sets A_i and B_i by assuming that all vertices in A_i (B_i, resp.) are colored red (blue, resp.).

5.1 Experimental results

Clearly the efficiency of this algorithm is based on that of computing well-separated pair decompositions of points and computing bichromatic closest pairs in well-separated point sets. As with the closest pair problem, experiments show that the running time is dominated by the computation of the well-separated pair decomposition. This is shown in Figure 5. Kruskal's algorithm was used in the last step to compute a Euclidean minimum spanning tree of the resulting sparse graph.

For points in the plane, our algorithm is inferior to the algorithm that runs Kruskal's algorithm on the Delaunay triangulation of the given point set. However, we expect our algorithm to be superior for points in higher dimensional space. We are in the process of experimenting with points in higher-dimensional space.

6 Conclusion

We have presented fast heuristic algorithms for the problems of bichromatic closest pair and Euclidean minimum spanning tree for points in $\Re^d$. We used the well-separated pair decomposition (WSPD) technique and introduced a variant, which we referred to as the well-separated pair decomposition for bichromatic points (WSPDB). The most significant advantage of well-separated pair decomposition and WSPDB is that their complexity (size and computation time) does not have a

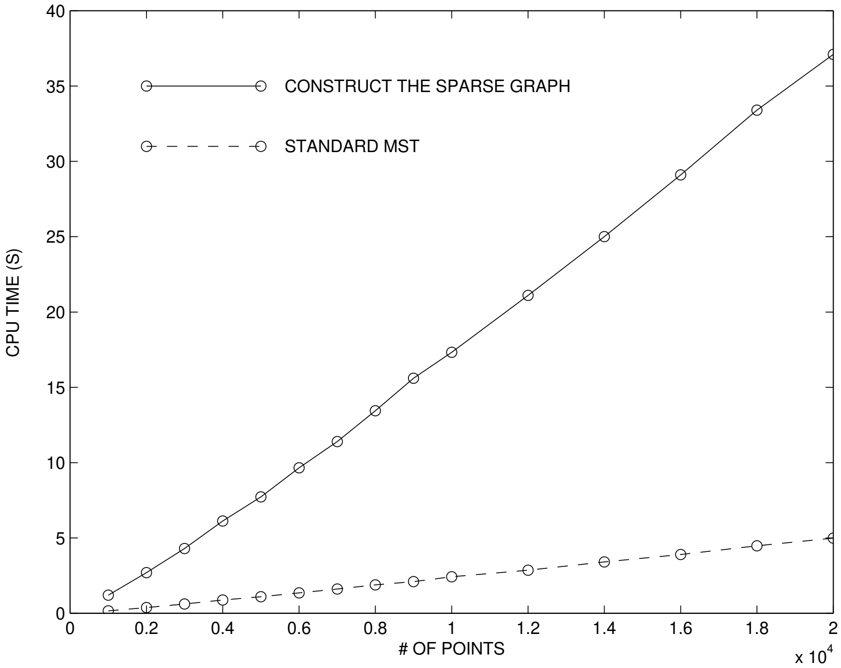

Figure 5: Running times to compute Euclidean minimum spanning tree

$O(n^{O(d)})$ dependence for points in d-dimensional space. Furthermore, these decompositions can be computed in $O(n \log n)$ time. Experiments indicate that these algorithms are fast in practice.

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[2] 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.

[3] 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.

[4] D. Krznaric, C. Levcopoulos, and B. Nilsson. Minimum spanning trees in d-dimensions. In *Proc. 5th European Sympos. on Algorithms*, pages 341–349, 1997.

[5] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38:96–102, 1995.

[6] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.