Euclidean Minimum Spanning Trees Based on Well Separated Pair Decompositions

Chaojun Li

Advised by: Dave Mount

May 22, 2014

1 INTRODUCTION

In this report we consider the implementation of an efficient algorithm for computing Euclidean minimum spanning trees, which will be based in part on the well-separated pair decomposition, introduced by Callahan and Kosaraju. In order to compute the Euclidean Minimum Spanning Tree of n points in the space, one can naively link each pair of edges to build the complete Euclidean graph G=(P,E), where P stands for the vertex set, and E stands for set of edges, which consist of all pairs (p, q) ∀ p, q ∈ P. For such a graph we have |E|=$\binom{n}{2}$, which is Θ(n^2), and combining this with the O(ElogV)=O(n^2logn) running time for the execution of Kruskal's Algorithm, would result in an overall running time of O(n^2logn). In this paper we present a more efficient solution.

Our algorithm for computing the Euclidean minimum spanning tree will be based on a number of components. More specifically, the program provides the construction of a point region quadtree, which used to store input set of data points. We then implemented the well-separated decomposition upon the constructed point region quadtree to store each well-separated pair of cells as cross links in the tree. The next procedure is to implement projections for each pair of well-separated cells in order to find (approximately) the closest pair of points and build the graph G. Finally, we implement Kruskal's Algorithm to obtain the Minimum Spanning Tree from G.

This paper is organized as follows. Section 2 defines the basic concepts of the well-separated decomposition, as well as how it is applied in this program.

Section 3 presents a projection method for approximating the closest pair of points between each well-separated pair and how this is used in Kruskal's algorithm. Section 4 provides a pseudo-code description of the complete method an evaluation of merits of this application of well-separated decomposition theorem, along with the graph showing the ultimate Euclidean minimum spanning tree built from several combined procedures. Section 5 make a conclusion about this newly-tested approximation algorithm.

2 CONCEPTS OF WELL-SEPARATED DECOMPOSITION

From [1, p. 88], let P be a given set of n points in the Euclidean space, for any factor s ≥ 1, two non-overlapping subsets A and B of P are said to be well-separated if both A and B can be enclosed within two Euclidean balls of radius r such that the minimum distance between S_a and S_b is at least sr.

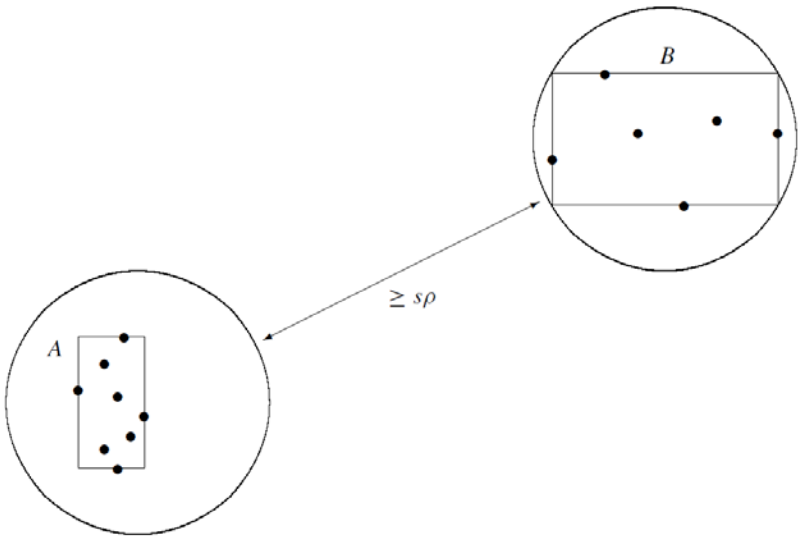

Figure 1: cell A and B are well-separated by the factor s

From [1, p.90], for a given point set P and a separation factor s, the well-separated decomposition is defined to be a collection of pairs of subsets of P, denoted $\{\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}\}$, such that

(1) $A_i, B_i \subseteq$ P, for $1 \leq i \leq m$
(2) $A_i \cap B_i = \emptyset$, for $1 \leq i \leq m$
(3) $\bigcup_{i=1}^{m}(A_i \otimes B_i)$ = P $\otimes$ P

(4) $A_i \; and \; B_i$ are s-well-separated, for $1 \leq \mathrm{i} \leq \mathrm{m}$

In order to construct the well-separated pair decomposition, we build a point quadtree in which to store input set data points. The quadtree is based on a recursive subdivision of two dimensional space which is based on repeatedly splitting a square into four quadrants. The result is a decomposition of space into squares, called cells, each of which contains a set of points. For the construction of WSPD, we make some modifications to the existing quadtree structure. In the general version of quadtree structure, each internal node serves as a placeholder, but does not store any data points, and each leaf node stores either zero or one data points in its associated cell.

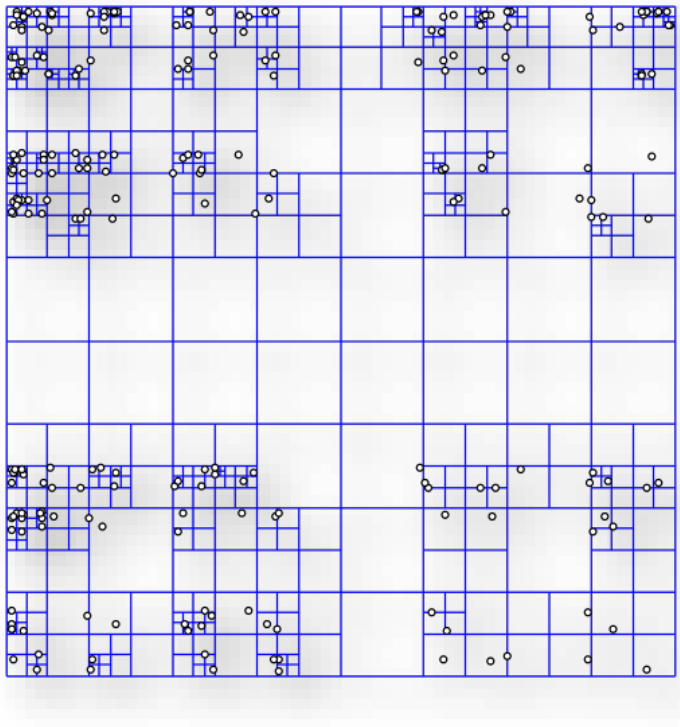

Figure 2: Point region quadtree partitions data points into cells.

In our augmented version of quadtree structure, we add an associated representative for each node u, which we denote by rep(u). As a leaf node, if it contains a point p, then rep(u)=$\{u\}$; otherwise it contains no point, then rep(u)=$\emptyset$. For the internal node, the case is different from that of the traditional version. Since each internal node contains more than one data point within its associated

cell, it must have two or more nonempty leaf nodes descended from it. In this case, we may randomly select one of such leaf children node v, and set rep(u)=rep(v) (see [1] for more detail)). Also, we associate each node with its depth in the tree, which we call its level. In my program, in order to compare levels among different nodes, we can first calculate the side-length of cell associated with each node, then determine the result by the key feature that the side length of a cell associated with u is no smaller than that of a cell associated with v if and only if level(u)≤level(v). The side length of a cell of is generated by the formula below. Let x be the side length of the cell associated with root node, each time we partition the cell, we divide the side length of the cell by 2. Accumulatively, the cell that is partitioned i times from the original cell, is smaller by the factor $1/2^i$. Thus, letting x denote the side length of the root cell, the side length of the associated cell of a node at level i is $x/2^i$.

Let us next consider the implementation of well-separated pair decomposition. The general idea of this algorithm is that each execution we check the pair of nonempty nodes if they can be decomposed into well-separated pairs, then that pair will be stored, otherwise we compare the level of this pair of nonempty nodes and decide which has larger cell size. We subdivide the node containing cell with larger size into its children nodes and recursively call that function for each child with the other node. (See [1] for more detail).

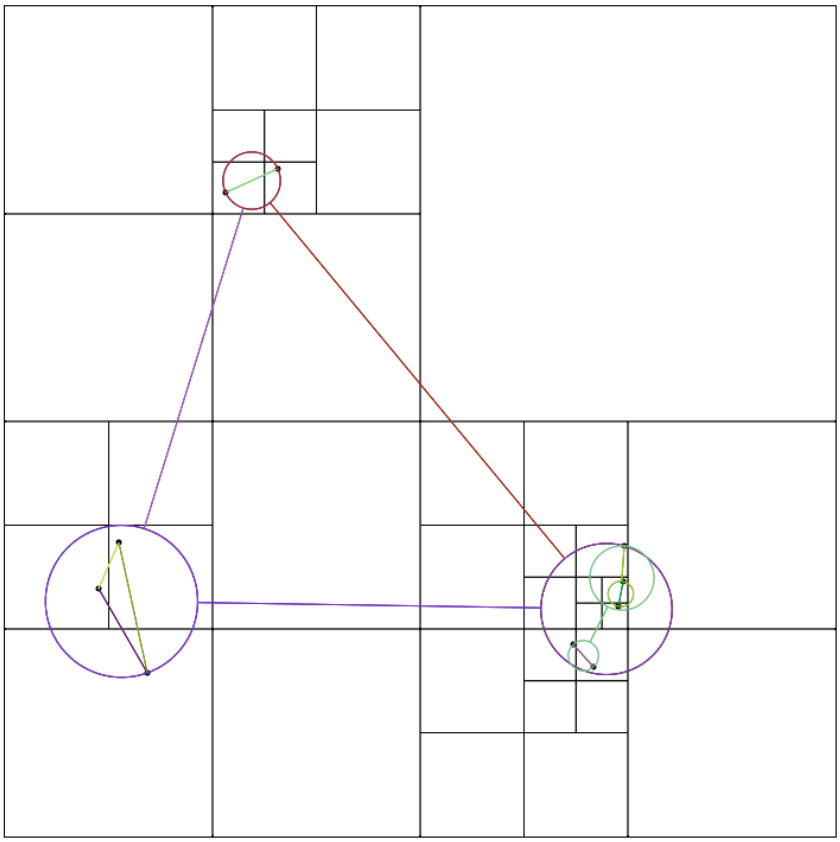

Figure 3: Some of the s-separated cells generated from the WSPD.

3 MINIMUM PROJECTION OF EACH WELL-SEPARATED PAIR AND KRUSKAL'S ALGORITHM

After we constructed each pair of well- separated pairs, the next step is to construct for each well separated pair P_1 and P_2 the closest pair of points in these two sets. We do this through an approximation. Consider the line segment joining the centers of the Euclidean balls containing P_1 and P_2, and let $p_1 \in P_1$ and $p_2 \in P_2$ be points with closet projections along this line. The essence of mathematical formula used in [2]. Let u_1 and u_2 be the two nodes defining any of the well-separated pairs each containing a set of data points and let (x_1,y_1), (x_2,y_2) be the center points of the cells associated with u_1 and u_2, respectively. First, define the vector between two center points by $\vec{V}$= (x_2-x_1, y_2-y_1). Then for each data point (x_p, y_p), we create another vector $w_p = (x_p - x_1, y_p - y_1)$. In order to calculate the projection distance from (x_p, y_p) to (x_1, y_1), we create the distance vector d_p by the inner product of $\vec{v}$ and w_p such that d_p=$\vec{v} \cdot w_p$ = $(v_x \cdot w_{p_x} + v_y \cdot$

w_{p_y}). After recursively executing the above calculation for each data point p= (p_x, p_y) ϵ u2, where $p_x \in [x_2 - r_{u2_x}, x_2 + r_{u2_x}], p_y \in [y_2 - r_{u2_y}, y_2 + r_{u2_y}]$, we can find the minimum projection value among all calculated vector values w_p and store the according $p_2 = (p_{x_2}, p_{y_2}) \in u_2$. We repeat the above steps with swapped u_1 and u_2 to obtain another data point $p_1 = (p_{x_1}, p_{y_1}) \in u_1$. The ultimate step for each separated pair is to add the edge (p_1, p_2) into the graph G with its weight w=distance (p_1, p_2). Following the above calculation method of finding minimum projections, we have added one edge into the graph G for each well-separated pair. Altogether, the number of edges |E| in the graph G is the same as the number of well-separated pairs.

Now, we just have one more step left to obtain the minimum spanning tree from this graph G. Among the best known algorithms for computing the minimum spanning tree of a graph are Kruskal's algorithm and Prim's algorithm. In my program, I chose the former.

Kruskal's algorithm is a greedy algorithm in the goal of building a subset tree which includes each vertex in the original graph with the minimized weight of all edges connected within (see details in [3]). In my version of the application of Kruskal's algorithm, I built a priority queue to sort the set of edges in graph G based on its associated weight. In that way, each stage of the while loop within the Kruskal's algorithm, the program will make the local decision of selecting the edge with the lowest weight among the set of all input edges with the hope of finding global optimal solution in a reasonable time (See detail in [4]).

There are two standard methods for implementing adding edge process: one is the union-find data structure, the other is the labeling method. In my program, I chose the latter. The structure I wrapped with this method is the treemap, the key of which has the type Integer, each index of the labeling, maps to a sequence of vertices in a cycle. During the execution of the loop, when we try to add a new edge into the minimum spanning tree, we check that if both vertices have not been added to any of the cycle yet, we create a new key, and map this Integer value to the new cycle containing just one edge. In the case that one vertex has been added to one of the cycles with other vertex not, we just add this new vertex into the cycle containing the other vertex. Another situation is that both vertices have been added to the same cycle, in which case this edge is redundant

for the new minimum spanning tree; hence, we do nothing. The last condition is bit complicated that both vertices are added to the treemap data structure but within different cycles. In this case, we will compare the labels associated with two different vertices and merge the cycle with larger label into the cycle with smaller label, removing larger key from the treemap. Finally, we will get the output of approximation minimum spanning tree, the sequence of edges.

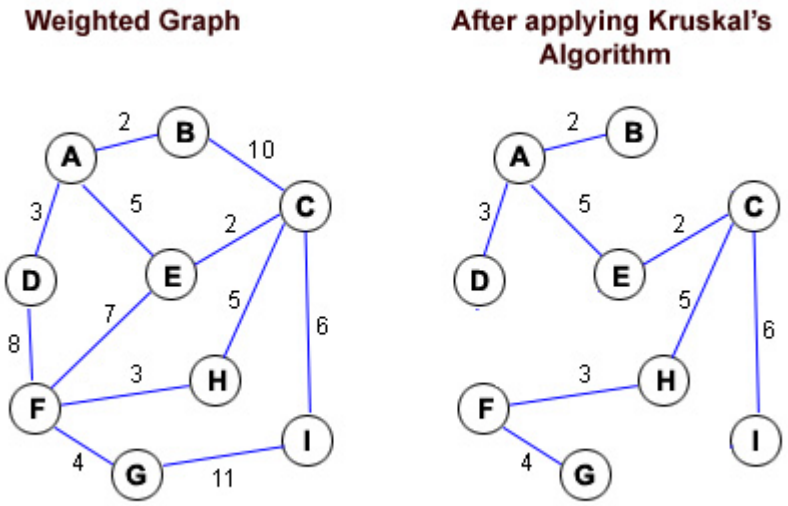

Figure 4: A sample of the minimum spanning tree built by Kruskal's Algorithm

4 ALGORITHM IN THE FORM OF PSEUDOCODE

Two algorithms, Well-Separated Pair Decomposition and Kruskal's algorithm, are presented in the following code block, each describing how the data structures are wrapped in the associated implementation.

Algorithm 1: Well-Separated Pair Decomposition Algorithm (from [1, p93])

```
1.  WS-pairs(u, v, s) {
2.          if (rep(u) or rep(v) is empty) return Ø;
3.          else if (u and v are s-well separated)
4.                  return {{u, v}};
5.          else {
6.                  if (level(u) > level(v))
```

7. Let $u_1, \ldots, u_m$ denote the children of u;
8. return $\cup_{i=1}^{m}$ WS-pairs(u_i, v, s);
9. }
10.}

Algorithm 2: Kruskal's Algorithm (from [3])

My implementation of this algorithm is based on the treemap data structure and the labeling method for maintaining connected components, each key in the map is the index of a label, and it maps to a sequence of vertices forming a cycle. At the beginning of execution, we initialize each label associated with the vertex with value -1.

---

1. A=∅
2. For each e ∈ $G.E$;
3. Make-set (E)
4. For each (u, v) ∈ E, ordered by weight (u,v), increasing, and stored in the priority queue
5. If Label(u)=-1 and Label(v)=-1, we add a new key that maps to this new edge, and put this pair into the treemap. A=A ∪ { (u, v)}
6. Else if Label(u)=-1 and Label(v)≠-1 we put u into the sequence of vertices that Label(v) maps to A=A ∪ {(u,v)}
7. Else if Label(u)≠-1 and Label(v)=-1, we put v into the sequence of vertices that Label(u) maps to A=A ∪ $\{(u, v)\}$
8. Else if Label(u)=Label(v), we continue to the next iteration
9. Else we compare the value of Label (u) and Label (v), and merge the sequence of vertices that larger key value maps to into the sequence of vertices that smaller key value maps to. Remove the larger key value from the treemap. A=A ∪ {(u,v)}

10.Return A

In this program, the algorithms described above are all linked with one another. The well-separated decomposition theorem provides a list of well-separated pairs served as the prerequisite of the minimum projection method. Compared with the naive way of building $\binom{n}{2}$ pairs of edges from the input set of n data points, this method makes local comparisons for each pair and construct the graph which contains the number of edges equal to that of the well-separated pairs. Such partially linked graph structure greatly saves space. The above two algorithms combined take time $\mathrm{O}(\mathrm{nlogn})$, which might also perform better for the efficiency concern. The last step is identical for either the traditional method or the newly application of well-separated decomposition theorem in my program. The Kruskal's algorithm executed on two different graphs with different size of edges. During the loop of this algorithm, it will check whether each existing edge should be added into the newly created minimum spanning tree, obviously, in our program, the graph G with fewer edges will have the great probability to save the total running time of minimum spanning tree algorithm, which costs $\mathrm{O}(n^2)$ in the naïve way of building the graph.

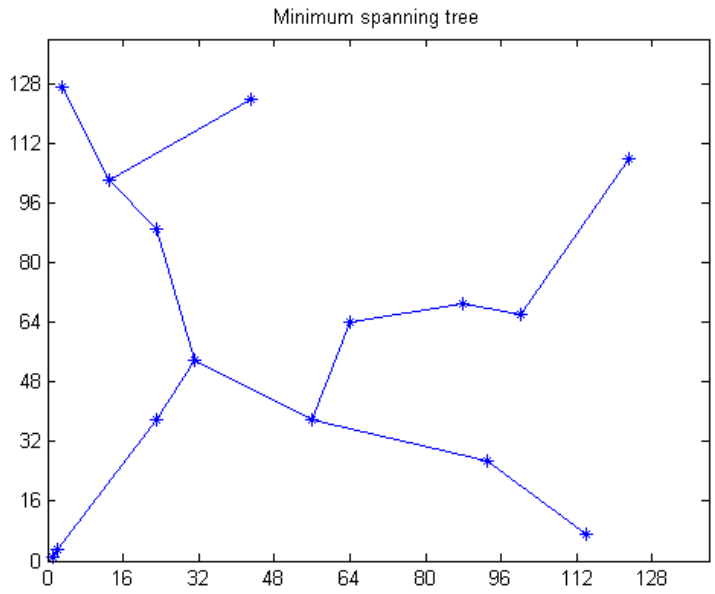

Figure 5: A minimum spanning tree generated from my program: the PR quadtree partitions data points, well-separated decomposition separates pairs, minimum projection method build edges, Kruskal's Algorithm find ultimately minimum spanning tree.

5 CONCLUSION

Well-separated decomposition theorem is widely applied in several fields such as astronomy, molecular dynamics, fluid dynamics, plasma physics, and even surface reconstruction. We present an algorithm that uses this concept to efficiently construct an approximation to the minimum spanning tree. This shows how geometry can be exploited to obtain algorithms that greatly improve the storage space and running time efficiency over traditional approaches.

References

1. David Mount. CMSC754 Lecture Notes. Mar. 2013. url: http://www.cs.umd.edu/~mount/754/Lects/754lects.pdf.

2. Meyer, Carl D. (2000). Matrix Analysis and Applied Linear Algebra. Society for Industrial and Applied Mathematics.

3. Wikipedia. Kruskal's_algorithm | Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Kruskal's_algorithm

4. Wikipedia. Greedy_algorithm| Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Greedy_algorithm