1 Introduction

Let $S$ be a set of $n$ points in $\mathbb{R}^3$. We study the discrete 2-center problem. The goal is to find two closed balls centered at some points in $S$ whose union contains all points of $S$ and such that the radius of the larger ball is minimized.

The discrete 2-center problem in its two dimensional version has been studied in [3], where the authors obtained an $O(n^{4/3} \log^5 n)$ running time.

In another version of the problem, called the continuous version, the centers of the disks are not confined to be at points in $S$. For this version, an $O(n^2 \log n)$ time result was given in [7] and was later improved by Sharir [10] to a near linear $O(n \log^9 n)$ time. Subsequently, using randomization, Eppstein achieved an expected time of $O(n^{3/2} \log n)$ [6].

The first solution to the continuous 2-center problem in $\mathbb{R}^3$ was given in [2], and their algorithm has a running time of $O(n^{3+\epsilon})$. Very recently this problem was the main focus of another work by Agarwal et al. [1], who described an $O(n^3 \log 8 n)$ expected time algorithm and an $O(n^2 \log^2 n / (1 - r^*/r_0)^3)$ expected time algorithm, where $r^*$ is the radius of the two centers of $S$ and $r_0$ is the radius of the smallest enclosing ball of $S$. The second algorithm would outperform the first except when the centers of the two spheres are very close to each other.

Here, we give an $O(n^2 \log^2 n)$ time, $O(n^2)$ space algorithm for the discrete 2-center problem in $\mathbb{R}^3$.

1.1 Definitions and terminology

In this section we introduce some terminology and definitions. For two points $a$ and $b$, $|ab|$ denotes the Euclidean distance from $a$ to $b$. Let $\Pi$ denote a plane such that $\Pi$ bisects the line segment $[pq]$ and $[pq] \perp \Pi$, and $h_{pq}$ denote the open halfspace bounded by $\Pi$ and containing $p$.

Given $p, q \in S$, the $q$-farthest point $f_{pq}$ is defined as the farthest point from $p$ that is contained in the open halfspace $h_{pq}$ (see Figure 1). A critical step in our solution is finding $f_{pq}$ for a fixed $p$ and all $q \in S \setminus \{p\}$ efficiently.

Given a sphere $\xi$ and a point $p$ on $\xi$, an inversion of $\xi$ with center of inversion $p$ maps $\xi$ into a plane by inverting the distance from $p$ to all points on the sphere [8]. Through inversion, using polar coordinates, a point $q = (\rho, \phi, \theta)$ is mapped to the point $q^* = (1/\rho, \phi, \theta)$, where $\rho$ is the distance from $p$ to $q$ and $\phi$ and $\theta$ are the polar angles. Notice that both polar angles are maintained through inversion. If $I$ denotes the inversion then $I(I(q)) = q$. Similarly, when applied to a plane that does not pass through $p$, the inversion yields a sphere which passes through $p$.

Consider the ball $\Sigma$ bounded by $\xi$. Let $\Pi$ be the plane corresponding to the inversion $I$. Then the interior of $\Sigma$ corresponds to one of the halfspaces bounded by $\Pi$.

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figure1.png}
\caption{p, q, and the q-farthest point $f_{pq}$.}
\end{figure}
2 The Algorithm

We begin by following the method presented in [4]. For a fixed point \( p \in S \), we can label all points \( q \in S \setminus \{ p \} \) with the \( q \)-farthest point \( f_{pq} \) as follows.

First, sort \( S \) in order of non-increasing distance from \( p \). Second, set \( f_{pq} \) for all points in \( S \) to be NULL. Third, pass through the sorted array and for each point \( q_i \), in order, set \( f_{pq} \) to \( q_i \) for all points \( q \in S \) that are not contained by the ball \( \Sigma(q_i, p) \) and for which \( f_{pq} \) is set to NULL. That is, all points of \( S \) that are in \( \bigcap_{k=1}^{i-1} \Sigma(q_k, p) \) but not in \( \Sigma(q_i, p) \), are labeled with \( q_i \), where \( i = 1, 2, \ldots, n - 1 \), and \( \Sigma(q_i, p) \) is a ball centered at \( q_i \) and having \( p \) on its bounding sphere.

After the above actions, the last value in the sorted array of points in \( S \setminus \{ p \} \) is the point which has minimum Euclidean distance from \( p \) (and \( \Sigma(q_n, p) = \Sigma(p, p) = 0 \)). This implies that \( f_{pq} \) is set for all points in \( S \setminus \{ p \} \). The sorted ordering ensures that at any step in the algorithm \( f_{pq} \), for any point \( q_i \), is the point corresponding to the smallest index \( j \) for which \( q_i \in \bigcap_{k=1}^{j-1} \Sigma(q_k, p) \) and \( q_j \notin \Sigma(q_j, p) \). This means that given \( q_i \) as \( q \), \( q_j \) is the farthest point from \( p \) whose ball \( \Sigma(q_j, p) \) does not contain \( q_i \), which matches the definition of \( f_{pq} \) for the given pair \( p, q_i \).

Without loss of generality, assume that \( n = 2^k \) for some integer \( k \). We build a complete binary tree \( T \) with \( k \) levels as follows. The leaves of \( T \) are associated with the balls \( \Sigma(q_i, p), i = 1, 2, \ldots, n \), in order. That is, the leftmost leaf of \( T \) stores \( \Sigma(q_1, p) \) and the rightmost leaf of \( T \) stores \( \Sigma(q_n, p) \).

Each internal node \( v \) of \( T \) stores a data structure associated with the common intersection of the balls that are leaf descendants of the sub-tree of \( T \) rooted at \( v \) (see Figure 2). Given a point \( q \), to find the smallest index \( j \) for which \( q \in \bigcap_{k=1}^{j-1} \Sigma(q_k, p) \) and \( q \notin \Sigma(q_j, p) \) start at the root of \( T \) and follow a path to a leaf of \( T \), at each node \( v \) along the path performing the following test: if \( q \) is in the common intersection stored at the left child of \( v \) then go to the right child of \( v \), else go to the left child of \( v \). Clearly, the index associated with the leaf where this search ends corresponds to the sought \( j \).

**Lemma 1.** The tree \( T \) can be constructed in \( O(n \log n) \) time and uses \( O(n \log n) \) space.

**Proof.** Invert all balls using \( p \) as the center of inversion and store the resulting halfspaces at the corresponding leaf nodes. This takes \( O(n) \) time and space. The balls in the original problem become halfspaces in the inversion space. The intersection of balls corresponds to the convex polytope that is the intersection of the halfspaces obtained through inversion. For \( n \) balls the complexity of this polytope is \( O(n) \). The faces of the polytope can be inverted back in \( O(n \log n) \) time to obtain the spherical portions of the intersection of the balls in the problem space.

Let \( v \) be an internal node of \( T \). We can obtain the polytope associated with \( v \) by computing the common intersection of the polytopes associated with the left and right children of \( v \), which takes linear time and space in the complexity of the children [5].

We can store both the polytope and the ball intersection at \( v \) without asymptotically increasing the space requirements. However, for our purpose, we only need to store the polytope. By performing a bottom-up traversal of \( T \), the overall time to compute the common intersections (the polytopes) for all nodes in \( T \) is thus \( O(n \log n) \). The space requirement is \( O(n \log n) \).

**Lemma 2.** Given a set of \( n \) balls all having a common point \( p \) on the surfaces of their respective bounding spheres, and a query point \( q \), the smallest index \( i \) such that \( \Sigma_i \) does not contain \( q \) can be found in \( O(\log^2 n) \) time.

**Proof.** Let \( q^* \) be the inversion of \( q \) with center \( p \). We use the complete binary tree \( T \) described earlier enhanced with point location capability at each internal node.

Specifically, when traversing \( T \) on a path from the root to a leaf, we need to decide at each in-
ternal node \( v \) on the path whether \( q^* \) is inside the convex polytope associated with the left child of \( v \). If it is, we descend to the right child, otherwise we descend to the left child.

Assume the left child \( u \) of \( v \) has \( m \) leaf descendants. Having stored at \( u \) a point location data structure that requires \( O(m) \) space and can be constructed in \( O(m) \) time, the query at \( v \) can be answered in \( O(\log m) \) time \[9\]. Over all nodes in \( T \) the point location data structures can be built in \( O(n \log n) \) time using \( O(n \log n) \) space, which is done as a preprocessing step.

The data structure for \( p \) can be discarded after \( f_{pq} \) is found for each \( q \in S \setminus \{p\} \). We repeat this computation replacing \( p \) with each of the points in \( S \). Thus, we obtain an \( n \times n \) array \( A \) of \( f_{pq} \) points for all \( p \) and \( q \) in \( S \).

By performing a traversal of \( A \) we find \( p \) and \( q \) such that we minimize \( \max\{|pf_{pq}|, |qf_{qp}|\} \) over all possible combinations of \( p \) and \( q \). Thus, we obtain:

**Theorem 1.** Given a set \( S \) of \( n \) points in \( \mathbb{R}^3 \), the discrete 2-center of \( S \) can be found in \( O(n^2 \log^2 n) \) time using \( O(n^2) \) space.

**References**


