Finding HSP Neighbors via an Exact, Hierarchical Approach

Cole Foster[1](✉), Edgar Chávez[2], and Benjamin Kimia[1]

[1] Brown University, Providence, RI 02912, USA
{cole_foster,benjamin_kimia}@brown.edu

[2] Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Mexico
elchavez@cicese.edu.mx

Abstract. The Half Space Proximal (HSP) graph is a low out-degree monotonic graph with wide applications in various domains, including combinatorial optimization in strings, enhancing kNN classification, simplifying chemical networks, estimating local intrinsic dimensionality, and generating uniform samples from skewed distributions, among others. However, the linear complexity of finding HSP neighbors of a query limit its scalability, except when sacrificing accuracy by restricting the test to a small local neighborhood estimated through approximate indexing. This compromise leads to the loss of crucial long-range connections, introducing false positives and excluding false negatives, compromising the essential properties of the HSP. To overcome these limitations, we propose a fast and exact HSP Test showing sublinear complexity in extensive experimentation. Our hierarchical approach leverages pivots and the triangle inequality to enable efficient HSP search in general metric spaces. A key component of our approach is the concept of the *shifted generalized hyperplane* between two points, which allows for the invalidation of entire point groups. Our approach ensures the desired properties of the HSP Test with exactness even for datasets containing hundreds of millions of points.

Keywords: Similarity Search · Half-Space Proximal · Exact

1 Introduction

The antidote to complexity in search is organization. In similarity search in a metric space, the local topology of a dataset element allows for efficient query search. Traditional approaches to capturing this local topology in the form of a graph have often focused on pairwise distances, such as the kNN graph, where each element is connected to its k nearest neighbors. Proximity graphs such as the Relative Neighborhood Graph (RNG), the Gabriel Graph, *etc.*, use triplets of elements at a time to determine the occupancy of a neighborhood region, *e.g.*, a lune defined by two elements, by a third element. While kNN graphs and

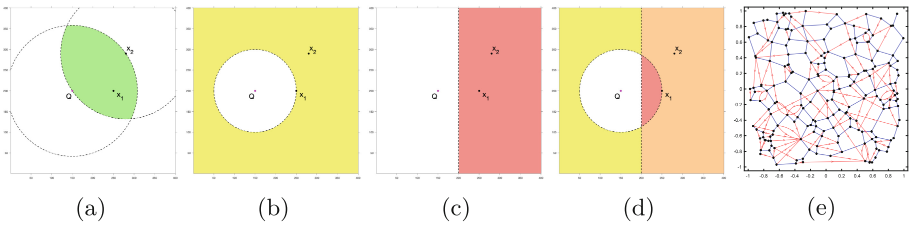


Fig. 1. (a) A link between Q and x_2 is disrupted by the presence of x_1 in the RNG lune between Q and x_2. The region of x_2 whose link with Q is invalidated is the intersection of the region corresponding to Eq. 1(a) shown in (b) and the region region corresponding to Eq. 1(b) shown in (c), with the intersection shown in (d). The RNG and HSP Graph on a 2D dataset of uniformly distributed points with the RNG links are shown in blue and the additional, directed HSP links are shown in red. (Color figure online)

other pairwise-distance graphs can unevenly distribute neighbors, clustered in one direction, the proximity graphs embed a sense of direction in their neighborhood definition, albeit requiring additional cost. Similarly, the Yao graph [20] and the Theta graph [4] construct a sense of direction by defining equal angle cones around each point through which links can be made between two points, but this is restricted to a Euclidean space.

The Half-Space Proximal (HSP) graph [6] aims to capture a sense of direction, but without the computational requirements of the RNG and without the Yao/Theta graphs' restrictions to a Euclidean space. Like an RNG, it connects two elements if the lune between them is empty, but unlike the RNG which checks occupancy by *all* other elements, HSP checks occupancy by those elements already established as HSP neighbors, Fig. 1(a).

This recursive definition requires a treatment of elements rank-ordered by distance to the query. Consider a dataset $\mathcal{S}$ of elements $\{x_1, x_2, \ldots, x_N\}$ and a query point Q. The only element whose lune with Q is unconditionally empty is the nearest neighbor, which is denoted as x_1. Then, the set of elements x_2 whose RNG lune between Q and x_2 is occupied with x_1 is the intersection of two regions,

$$\begin{cases} d(Q, x_1) < d(Q, x_2) & \text{(1a)} \\ d(x_1, x_2) < d(Q, x_2) & \text{(1b)} \end{cases}$$

with the first inequality defining the outside-disc yellow area in Fig. 1(b) and the second inequality defining the right half-space red area in Fig. 1(c), with their intersection denoted by the orange region in Fig. 1(d). This test is effective in discarding nearly half the space. The nearest neighbor among the surviving elements (later called the active list) is now also an undisputed HSP neighbor which in turn discards members in its own half-space, Fig. 2. The process is repeated until all members are considered. It should be clear that the HSP is a superset of the RNG, Fig. 1(e).

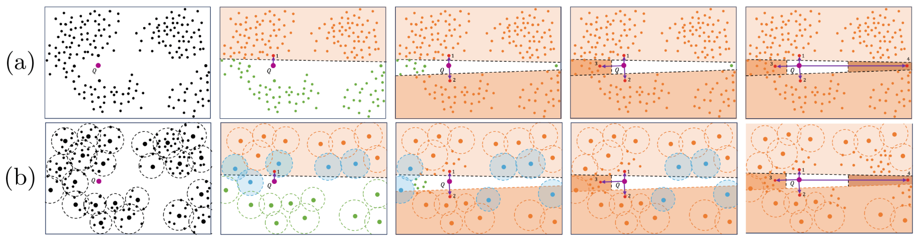
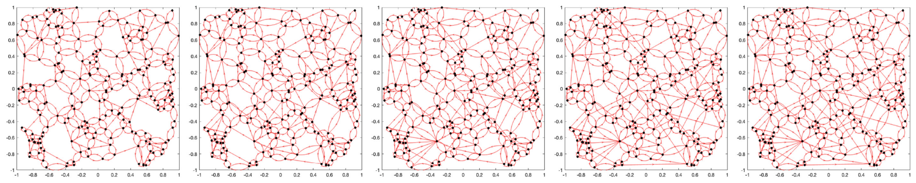
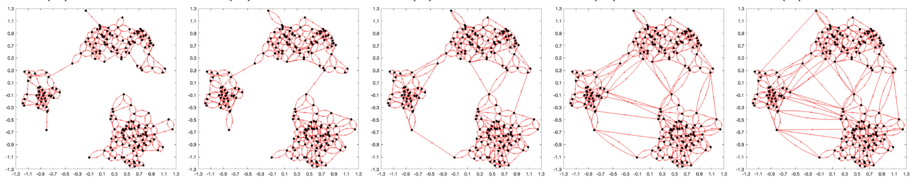


Fig. 2. (a) A visualization of finding the HSP neighbors of Q among the points shown on the left. Each of the nearest neighbors rules out a half-space (shaded area). (b) A visualization of the proposed Hierarchical HSP algorithm, where pivots are used to invalidate entire groups of points without having to calculate distances to query or testing inequalities (Color figure online)



(a) $k = 10$ (b) $k = 20$ (c) $k = 40$ (d) $k = 80$ (e) Exact



(f) $k = 10$ (g) $k = 20$ (h) $k = 40$ (i) $k = 80$ (j) Exact

Fig. 3. A dataset of $N = 200$ 2D points is used to compare the exact HSP Graph to the approximate HSP Graph where the HSP neighbor candidates are restricted to the k closest points. The top and bottom rows correspond to uniformly distributed and clustered data, respectively (Color figure online)

Efficient Approaches to Computing the HSP: Despite the numerous benefits and applications of the HSP graph (see below), there has been surprisingly few attempts at reducing the $O(N)$ complexity in finding HSP neighbors of an element. All existing approaches restrict the HSP neighbors to a local area around Q: in the original paper [6], the HSP Test is performed on the Unit Disc Graph, thus constraining the test to a small radius around Q. More recently, an approximate kNN search (k = 300) by HNSW [11] was used to provide a similar constraint in the application of instance-based classification [19].

However, any method of restricting the HSP Test to some local area will inevitably result in an approximation. Consider Fig. 2, where the fourth HSP neighbor lies very far away from Q. These types of links, called *shooters*, are often those occupied-lune neighbors that are a result of near-parallel generalized hyperplanes invalidating nearby points and thus preventing them from invalidating further ones. Figure 3 shows how approximate methods miss these long range links which are significant as they preserve some of the essential qualities of the HSP Graph, including its monotonic property described next.

HSP is a Monotonic Graph: This idea of a fully monotonic graph, namely one where each pair of nodes (u, v) in the graph has a *monotonic path* between them, *i.e.*, a path $u = u_0, u_1, u_2, \ldots, u_{\ell+1} = v$ where $d(u_{i+1}, v) \leq d(u_i, v)$ for all i on the path, has been an elusive character in the history of similarity search. The notion emerged in 1985 when Dearholt [9] proposed the Monotonic Search Network (MSNET) as the ideal graph for Computer Vision databases to enable "search without backtracking", constructed by adding edges to the RNG Graph. In 2002, Navarro [17] proposed the Spatial Approximation Tree (SA-Tree) which constructs a monotonic path from the root node and to every point in the dataset. The conditions used to construct these monotonic paths in the SA-Tree are exactly the same as used for the HSP Test [6], which was introduced just a few years later. Chavez *et. al.* [18] later proved the HSP Graph to be a monotonic graph [18]. Although a monotonic graph does not guarantee exact search for a query outside of the dataset, the conditions used to provide monotonicity in the SA-Tree and the HSP are leveraged to provide diverse links in state-of-the-art graph-based approximate nearest neighbor search [10,11].

Application of HSP: The initial application of the HSP graph was for routing between nodes in *ad-hoc networks* where the challenge is using only local information without central control. Another application is the challenging optimization problems in strings [15], where the HSP is used for selecting a central string from a set. Firstly the median string of a set is obtained and from the median string, which with high probability will not be in the set, the HSP neighbors are computed. The HSP test provides sufficient diversity within the members of the subset while at the same time fulfilling the centrality criterion.

The HSP neighbor finding has been used to enhance the majority-rule neighborhood classifiers where the kNN neighbors are replaced with the HSP neighbors, eliminating the need to set the parameter k. The candidate neighbors come from a probabilistic index such as the HNSW [11].

Another application of the HSP graph is in representing chemical networks [1, 2]. The typically used complete graph or an α-similarity graph are significantly reduced in size by retaining only HSP neighbors, reducing quadratic memory requirements to linear ones.

The HSP neighbors have also been used in local intrinsic dimensionality estimation [14]. The connection between the maximum degree of the HSP graph and the kissing or sphere packing number (the maximum number of mutually touching spheres in a Euclidean space) is used to define the indexability of a set.

In [13], the authors define the *hubness* HSP (HubHSP) graph. The geometric structure of the HSP neighbor definition remains intact, but instead of nearest neighbors in each step they use the node with the highest measure of "hubness", which they define. This tilts the process in favor of hubs, important to sampling skewed distributions, for example.

Exact Computation of HSP is Significant: Finding exact neighbors is key to local intrinsic dimension as well as local density estimation. Also, a standing conjecture about the HSP is about it being t-spanner of finite stretch t. This has a potential application as a distance oracle. Storing the entire distance matrix of a metric database implies $O(N^2)$ space of precomputed distances, while storing the HSP graph uses space proportional to $O(N \times \delta(HSP(S))$, with $\delta(.)$ the degree of the graph. Using the monotonic property of the HSP the oracle can be consulted in time proportional to the diameter of the graph. For the above oracle to work, it is mandatory to compute the HSP exactly to obtain a proper bound.

Hierarchies in Exact Metric-Space Similarity Search: Brute force exact metric-space search is avoided by using indices which leverage *pivots*, select points in the dataset, and the triangle inequality to bound the distance between the query point and other members of the dataset, removing a large portion from consideration. For example, the List of Clusters (LOC) [7] organizes the dataset into an ordered list of pivots, where each pivot is responsible for a group of points within a radius of that pivot. The query can traverse the list, only considering the clusters that may contain the nearest neighbor. By the observation that an increase an N also increases the number of points in each cluster, the Recursive List of Clusters [12] brings greater efficiency by organizing each cluster into its own LOC. In fact, this recursive organization of the dataset into smaller and smaller groups is the basic concept of tree structures, which achieve logarithmic search complexity. Some metric-space tree structures of interest include the M-Tree [8], the Cover Tree [3], and several others [5,16,21].

Overview: The proposed approach constructs a multilayer hierarchy of coarse-scale pivots "owning" finer-scale pivots in their pivot domains defined by a radius. The hierarchy is then used to (i) find the nearest neighbors to a query without computing distance to each element by relying on conditions on pivots and (ii) use pivot conditions to discard entire domains from being HSP neighbors, preserve entire domains as active, or declare domains as indeterminant which require examination of its members. This results in significant savings and leads to scalable HSP computation on large-scale, high-dimensional datasets featuring clustered data. Our implementation is publicly available at https://github.com/cole-foster/HHSP.git.

2 Finding the Exact, Hierarchical HSP Neighbors (HHSP)

The main bottleneck in finding the HSP neighbors of a query is (i) the computation of the distance to *all* dataset members, and (ii) to a lesser extent, the validation of the inequalities in Eq. 1. The key intuition is to use a hierarchy where pivots representing a group of dataset points can be examined and used to either discard or retain from consideration the entire elements in the pivot domain, thus avoiding explicit computation of distances or validation of inequalities.

Specifically, recall that in the HSP algorithm the distance between Q and all dataset members are calculated to return x_1 as the closest point to Q. Since x_1 is the nearest neighbor of Q, it becomes the first HSP neighbor and is used to discard a half-space from being HSP neighbors of Q by checking Eq. 1 on the remaining dataset members, Fig. 2(a). Next, the closest point among the active list $\mathcal{A}$ which is the set of surviving dataset members becomes the second HSP neighbor and is used to check and discard the remainder of the active list that satisfy Eq. 1. This process is repeated until all members are labeled either as HSP neighbors or discarded, with overall complexity of $O(N)$.

A first key savings can be achieved through the computation of the nearest neighbor of Q by using pivots in a hierarchy. Consider, at first, a two-layer hierarchy where the "bottom" layer is the dataset and the "top" layer is a select group of elements of the dataset called pivots p_i such that each data point x_j is in the domain of one, and only one, p_i, namely, $d(x_j, p_i) \leq r$, where r is a fixed parameter of the hierarchy, Fig. 4(a). There are numerous ways to construct this hierarchy. The approach used here is to randomly consider the dataset members one by one and assign them to either belonging to an existing pivot or if no pivot can be found, assign it as a pivot. The parameter r determines the size of the pivot domain.

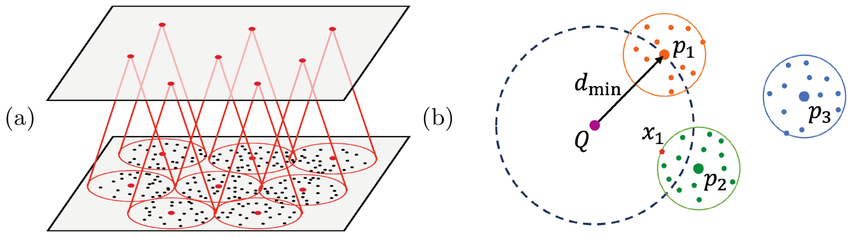


Fig. 4. (a) A two-layer hierarchy where the bottom layer contains all of $\mathcal{S}$ and the top layer has a select few pivots where the distance of dataset elements to its pivot parent is less than r. (b) Given an upper-bound distance $d_{\min}$, a pivot p may contain the nearest neighbor x_1 if $d(Q, p) \leq d_{\min} + r$ (Color figure online)

The pivot structure can be used to significantly reduce the computational effort in finding the nearest neighbor of Q. Let $d_{\min}$ denote the minimum distance of the query Q to the data elements already considered. Then, the distance from Q to any other element x in the pivot domain of p satisfies the triangle inequality

$$d(Q,p) - r \leq d(Q,p) - d(p,x) \leq d(Q,x) \leq d(Q,p) + d(p,x) \leq d(Q,p) + r. \quad (2)$$

Thus, if for a pivot p, $d(Q,p) > d_{\min} + r$,

$$d_{\min} < d(Q,p) - r \le d(Q,x), \tag{3}$$

and the pivot domain cannot contain the nearest element, see the blue pivot domain in Fig. 4(b). On the other hand, if $d(Q,p) \le d_{\min} + r$, the green pivot domain in Fig. 4(b), the elements in the domain of p must be explicitly considered and if one has a lower distance than $d_{\min}$, it updates $d_{\min}$. This prevents the computation of distances to a vast majority of dataset elements. The process is repeated until all pivots have been considered in this way. A better performance can be achieved by using tighter bounds if each pivot would maintain the distance to its most distant member, r^*.

A second key savings can be achieved through wholesale examination of Eq. 1 for all members of a pivot domain *without* calculating distance to query or validation of inequalities by examining $d(Q,p)$. The following proposition prevents member-wise validation of the second inequality in Eq. 1 if the pivot satisfies certain conditions:

Proposition 1. *Let Q be a query, x_1 the furthest HSP neighbor of Q thus far, and p_2 a pivot with domain radius r satisfying the following:*

$$\begin{cases} d^2(Q,p_2) - d^2(x_1,p_2) > 2r\ d(Q,x_1) & \text{(4a)} \\ d(Q,p_2) \ge r. & \text{(4b)} \end{cases}$$

Then, all points $x_2 \in \mathcal{D}(p_2,r)$, i.e., where $d(x_2,p_2) \le r$, satisfy $d(x_1,x_2) < d(Q,x_2)$.

Proof. By the triangle inequality and $d(x_2,p_2) \le r$,

$$d^2(x_1,x_2) \le [d(x_1,p_2) + d(p_2,x_2)]^2 \le [d(x_1,p_2) + r]^2 = d^2(x_1,p_2) + 2r\ d(x_1,p_2) + r^2. \tag{5}$$

Applying the given Eq. 4,

$$\begin{aligned} d^2(x_1,p_2) + 2r\ d(x_1,p_2) + r^2 &< d^2(Q,p_2) - 2r\ d(Q,x_1) + 2r\ d(x_1,p_2) + r^2 \\ &= d^2(Q,p_2) - 2r\,[d(Q,x_1) - d(x_1,p_2)] + r^2 \\ &\le d^2(Q,p_2) - 2r\ d(Q,p_2) + r^2 \\ &= [d(Q,p_2) - r]^2 \\ &\le [d(Q,x_2) + d(x_2,p_2) - r]^2 \\ &\le [d(Q,x_2) + r - r]^2 \\ &= d^2(Q,x_2). \end{aligned} \tag{6}$$

■

It is intriguing that the region where p_2 satisfies Eq. 4(a) is the half-space to the right of a shifted generalized hyperplane in a Euclidean space, because the quadratic terms involving coordinates of p_2 cancel out leaving a linear equation which represents a hyperplane. Returning now the first inequality in Eq. 1, the following proposition identifies the condition on a pivot so that the entire pivot domain can be discarded.

Proposition 2. *Let Q be a query and x_1 the furthest HSP neighbor of Q thus far, then a pivot p_2 with radius r satisfying both of the following inequalities*

$$\begin{cases} d^2(Q,p_2) - d^2(x_1,p_2) > 2r\ d(Q,x_1) & \text{(7a)} \\ d(Q,x_1) < d(Q,p_2) - r, & \text{(7b)} \end{cases}$$

invalidates all points $x_2 \in \mathcal{D}(p_2,r)$, i.e., $d(p_2,x_2) \leq r$, as a HSP neighbors of Q.

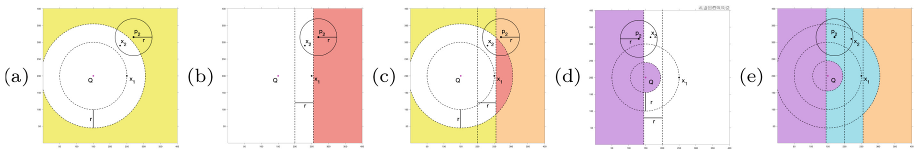


Fig. 5. (a) The point x_2 will satisfy Eq. 1(a) when its parent pivot p_2 is outside of the circle of radius $d(Q,x_1)+r$ centered at Q. (b) The point x_2 satisfies Eq. 1(b) when p_2 is to the right of the generalized hyperplane shifted by r. (c) The intersection of the two regions (orange) which defines locations for p_2 where its pivot domain members cannot be an HSP neighbor of Q. (d) The point x_2 does not satisfy both inequalities of Eq. 1 when p_2 falls into the purple region. (e) When p_2 falls into the blue region, it is undetermined if x_2 satisfies both inequalities of Eq. 1 (Color figure online)

Proof. First, let's show that $d(Q,x_1) < d(Q,x_2)$:

$$d(Q,x_1) < d(Q,p_2) - r \leq d(Q,x_2) + d(x_2,p_2) - r \leq d(Q,x_2) + r - r = d(Q,x_2). \tag{8}$$

Second, since $0 \leq d(Q,x_1)$, Eq. 7(b) shows that $d(Q,p_2) > r$ which together with Eq. 7(a) satisfy Proposition 1 which states that $d(x_1,x_2) < d(Q,x_2)$, the second inequality of Eq. 1 holds. ■

The regions corresponding to Eqs. 7(b) and Eqs. 7(a) are shown in Fig. 5(a) and 5(b), respectively, leading to their common intersection in Fig. 5(c).

In addition to determining which pivot domains are entirely ruled out, it is also possible to determine which pivot domains cannot get ruled out in their entirety because their members do not satisfy either the first or the second inequalities in Eq. 1 and can therefore remain on the active list in their entirety.

Proposition 3. *Let Q be a query, x_1 be the furthest HSP neighbor of Q thus far, and p_2 a pivot satisfying either of the following:*

$$\begin{cases} d^2(x_1,p_2) - d^2(Q,p_2) \geq 2r\ d(Q,x_1) \quad \textit{and} \quad d(x_1,p_2) \geq r & \text{(9a)} \\ d(Q,p_2) \leq d(Q,x_1) - r. & \text{(9b)} \end{cases}$$

Then, all points $x_2 \in \mathcal{D}(p_2,r)$, i.e., $d(p_2,x_2) \leq r$, violate one of the inequalities of Eq. 1, i.e., either $d(Q,x_1) \geq d(Q,x_2)$ or $d(x_1,x_2) \geq d(Q,x_2)$.

Proof. First, Eq. 9(b) implies that

$$d(Q,x_2) \leq d(Q,p_2) + d(p_2,x_2) \leq d(Q,p_2) + r \leq d(Q,x_1). \tag{10}$$

Second, by the triangle inequality, Eq. 9(a), and $d(p_2,x_2) \leq r, r \geq 0$,

$$\begin{aligned} d^2(Q,x_2) \leq [d(Q,p_2) + d(p_2,x_2)]^2 &\leq [d(Q,p_2) + r]^2 \\ &= d^2(Q,p_2) + 2r\ d(Q,p_2) + r^2 \\ &\leq d^2(x_1,p_2) - 2r\ d(Q,x_1) + 2r\ d(Q,p_2) + r^2 \\ &= d^2(x_1,p_2) - 2r\,[d(x_1,Q) - d(Q,p_2)] + r^2 \\ &\leq d^2(x_1,p_2) - 2r\ d(x_1,p_2) + r^2 \\ &= [d(x_1,p_2) - r]^2 \\ &\leq [d(x_1,x_2) + d(x_2,p_2) - r]^2 \\ &\leq d^2(x_1,x_2). \end{aligned} \tag{11}$$

■

Figure 5(d) visualizes the regions defined by the inequalities in Eqs. 9(a) and 9(b), which is the union of the shifted half-space and a reduced radius disc. A pivot p_2 in the purple region is retained in the active list without detailed examination of its elements.

The pivots that are neither fully discarded (orange area) nor fully accepted as surviving in their entirety (purple area) can potentially contain elements which can be discarded and elements that survive (cyan area), Fig. 5(e). The elements in these pivot domains must be individually tested with the inequalities of Eq. 1. However, this determination can be delayed until the point where their elements need to be examined. It is entirely possible that this entire pivot domain would be discarded in the next steps.

The details of the procedure are in Algorithm 1. Basically, the hierarchy is used to efficiently find the nearest neighbor, Proposition 2 and 3 are used to discard entire pivot domains and retain an active pivot list $\mathcal{A}_1$ (purple area) of pivots, and an indeterminant list $\mathcal{I}$ (cyan area). The procedure is then repeated by finding the next nearest element by exploring pivots in $\mathcal{A}_1 \bigcup \mathcal{I}$. In the process, some of the pivots in $\mathcal{I}$ may have to be explicitly examined. The pivots are removed from $\mathcal{I}$, and added to an active point list $\mathcal{A}_2$ of elements. The process is repeated by finding the nearest element in $\mathcal{A}_1 \bigcup \mathcal{I} \bigcup \mathcal{A}_2$ until they are all exhausted.

Multi-layer Hierarchies: This two-layer hierarchical approach achieves efficiency by using pivots to avoid the consideration of a vast number of points. As N increases, the number of pivots and the number of points in each pivot domain must both increase. This motivates the use of additional layers, similar to other hierarchical indices [3,5,8,16,21]. Just as pivots are able to discard or

Algorithm 1: Hierarchical HSP Search on a 2-Layer Hierarchy

```
Input: $Q$ as the query point, $\mathcal{P}$ as the set of top layer pivots in the 2-Layer Hierarchy
       constructed on $\mathcal{S}$ with top layer radius $r$.
Output: HSP($Q$) as the exact HSP neighbors of $Q$ in $\mathcal{S}$.
begin
    Initialize HSP($Q$) $= \emptyset$, $\mathcal{A}_1 = \mathcal{P}$, $\mathcal{I} = \emptyset$, $\mathcal{A}_2 = \emptyset$.

    while $|\mathcal{A}_1| > 0$ or $|\mathcal{I}| > 0$ or $|\mathcal{A}_2| > 0$ do

        /* Finding the Next HSP Neighbor */
        Find $x_i$ as the closest active point:
          · Initialize $d_{\min}$ with the distance to the closest $p \in \mathcal{A}_1$ or $x \in \mathcal{A}_2$.
          · Update $d_{\min}$ with $d(Q,p)$ for $p \in \mathcal{I}$ if $p$ does not satisfy Eq. 1 for any
        $x_j \in$ HSP($Q$).
          · Iterate through $p \in \mathcal{A}_1$; search the domains of pivots where $d(Q,p) \leq d_{\min} + r$,
        updating $d_{\min}$.
          · Iterate through $p \in \mathcal{I}$; if $p$ satisfies $d(Q,p) \leq d_{\min} + r$, then remove $p$ from $\mathcal{I}$
        and validate each member of the pivot domain against Eq. 1 for all $x_j \in$ HSP($Q$).
        Those points that are retained are added to $\mathcal{A}_2$ and may update $d_{\min}$.
          · The closest active point $x_i$ becomes the next HSP neighbor, $x_i$ is added to
        HSP($Q$).

        /* Validation of the Active Points */
        For each $p \in \mathcal{I}$:
          1. If Prop. 2 satisfied for $Q$ and $x_i$, remove $p$ from $\mathcal{I}$.
          2. Otherwise, continue.
        For each $p \in \mathcal{A}_1$:
          1. If Prop. 2 satisfied for $Q$ and $x_i$, remove $p$ from $\mathcal{A}_1$.
          2. If Prop. 3 satisfied for $Q$ and $x_i$, continue.
          3. Otherwise, remove $p$ from $\mathcal{A}_1$ and add $p$ to $\mathcal{I}$.
        For each $x \in \mathcal{A}_2$:
          1. If Eq. 1 satisfied for $Q$ and $x_i$, remove $x$ from $\mathcal{A}_2$.
          2. Otherwise, continue.
    end
end
```

retain an entire pivot domain of elements, coarse-scale (large radius) pivots discard or retain pivot domains of finer-scale pivots, *e.g.*, Fig. 6. Additional layers can achieve enhanced efficiency when N is increased further, Fig. 7.

3 Experiments

Exact HSP Complexity on Uniformly Distributed Data: The HHSP's complexity is examined on uniformly distributed data of varying dimension, Fig. 7. Observe that the indexing construction time, number of distances for search, and search time all depict an approximately linear profile against N across dimension for varying numbers of layers in a log-log domain. A straight line in log-log is $\log(y) = \alpha \log(N) + \gamma$ or $y = \beta N^{\alpha}$. While theoretical complexity bounds have not yet been defined, the above experiments suggest an approach to characterizing complexity of the HHSP, Table 1.

Comparison to Brute Force: The traditional concern in using exact query search is the curse of dimensionality, where as the intrinsic dimension of the dataset grows the index becomes less effective, eventually being no more effective than brute-force search. In such arguments, the size of the dataset is kept

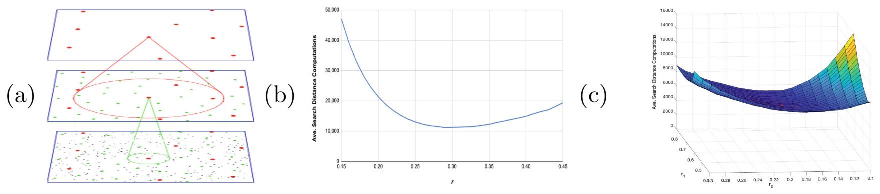


Fig. 6. (a) Depiction of a three-layer hierarchy. (b) The radius for a 2-layer hierarchy may be chosen to minimize the average number of distance computations on search. (c) Similarly, choosing radii for a three-layer hierarchy can be posed as a 2D optimization.

Table 1. The experimental complexity of the HHSP based on uniformly distributed data is captured by the value α for the experimental complexity $O(N^\alpha)$ using a least-squares fit of the function $y = \beta N^\alpha$.

Complexity	2D	4D	6D	8D	10D
Index Construction Distances	1.060	1.113	1.189	1.534	1.542
Index Construction Time	1.131	1.150	1.221	1.605	1.687
Index Memory Usage	0.960	0.966	0.968	0.970	0.983
Search Distances	0.108	0.203	0.311	0.447	0.553
Search Time	0.979	0.968	0.917	0.840	0.810

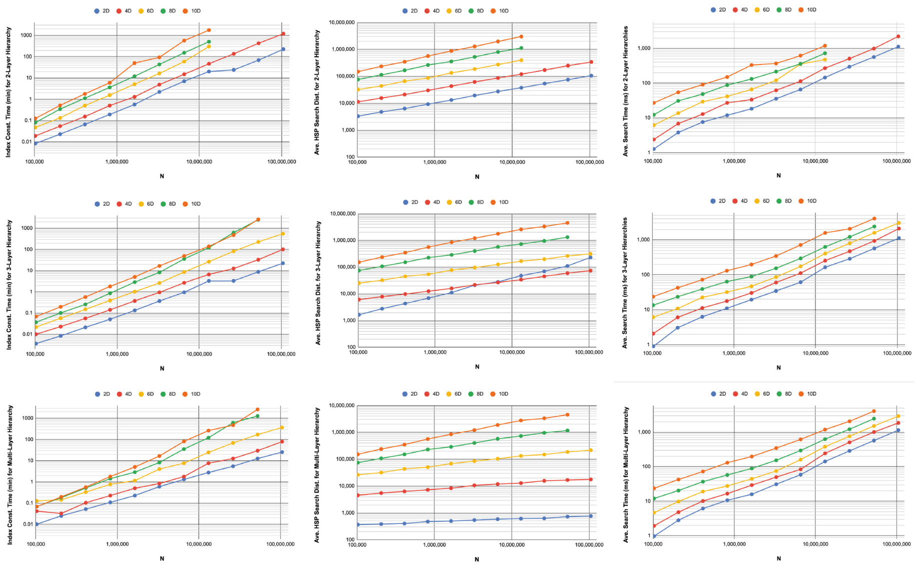
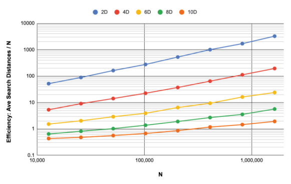
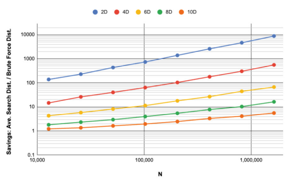


Fig. 7. Index construction time, number of distances, and search time for HHSP on 2D-10D data using 2, 3, and multilayer indices. Observe that these plots are approximately linear in the log-log domain

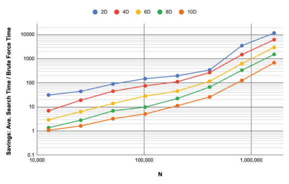
constant as the "volume" over which the dataset elements are spread becomes larger and thus density approaches zero. We posit that the key to determining the effectiveness of an index is to keep "density" constant. Our analysis shows that the index remains effective over a variety of dimensions and dataset sizes, Fig. 8. For example, in datasets of $N = 1,638,400$ points, the HHSP achieves time savings of 686.7x in 10D, enabling exact search on datasets of even 100 million points, Fig. 7.



(a)



(b)



(c)

Fig. 8. The savings of the HHSP over the brute force HSP algorithm in ratios of (a) the average number of search distances per N, (b) the ratio of average distance computations in comparison over those required for the brute force HSP test, and (c) the ratio of the average search time and that required for brute force search. It is evident that the index remains effective and its efficiency increases with N

HHSP Search on Clustered Data: Realistic datasets are typically not uniformly distributed. Rather, data points are often clustered, *e.g.*, object categories. The performance of the HHSP is measured on a synthetic clustered dataset created by initializing 100 uniformly distributed clustered centers from $[-1, 1]^D$ and using Gaussians with variance=0.05 to create 1,000 perturbations of each center, Fig. 9. Note that the rate of increase with D for the search time and the number of distance computations is significantly lower.

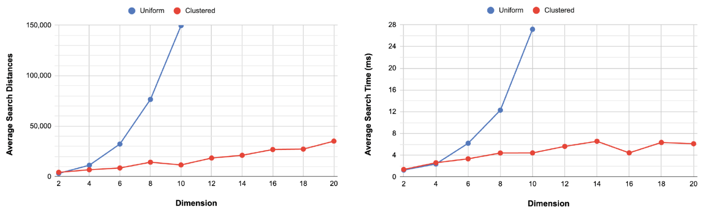


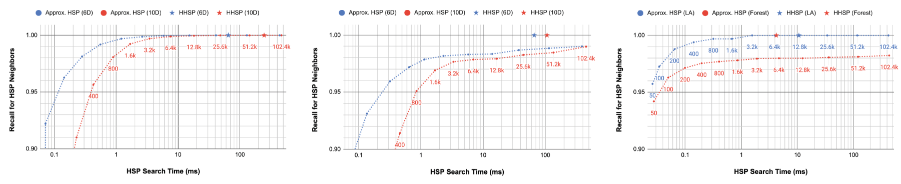
Fig. 9. A comparison of HHSP performance on uniformly distributed data vs. clustered data for $N = 100,000$ shows a significantly reduced rate of increase.

Real World Datasets: The performance of HHSP on real world datasets, Table 2. (1) *LA*, a 2D dataset of the geographic locations in Los Angeles with $N = 1,073,827$ points; (2) *Forest*, a 6D dataset of quantitative variables used for classifying forest cover types; and (3) *Corel*, a $N = 67,840$ collection of 32D color histograms of images. The HHSP significantly outperforms the brute force HSP Test in all cases, leading to 5,218x, 2,598x, and 8.64x savings over brute force time for LA, Forest, and Corel, respectively.

Table 2. Comparing the HHSP to the original HSP Test on Real-World Datasets.

Dataset	N	Ave. HSP Neigh	BF Time (ms)	BF Distances	Index Const. Time (s)	HHSP Time (ms)	HHSP Distances	Ratio of Times
LA (2D)	1,073,727	3.32	55,315.248	2,757,279.67	2.744	10.627	1,124.15	5,205.16
Forest (6D)	580,812	5.34	10,913.616	1,630,106.17	4.498	4.207	3,194.70	2,594.16
Corel (32D)	67,840	9.71	67.669	192,751.53	7.161	7.832	64,420.57	8.64

Comparison to Approximate HSP Search: The only existing, scalable approach at reducing the complexity of the HSP algorithm relies on performing an approximate kNN search to retrieve a large neighborhood around the query, and then apply the HSP Test on that neighborhood [19]. This approach leverages a state-of-the-art graph-based approximate search index, the Hierarchical Navigable Small World (HNSW) Graph [11]. This approach to HSP Search involves an inherent trade-off between search time and accuracy, requiring a large neighborhood to obtain exactness at the cost of longer search time, Fig. 10.



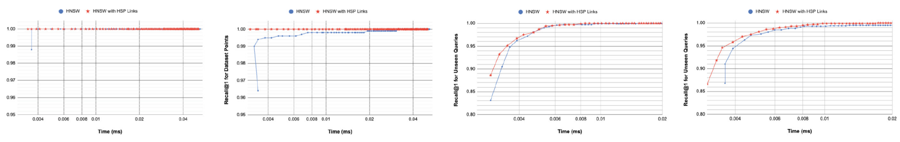
(a) Uniform (N=1.6M) (b) Clustered (N=1.6M) (c) Real Datasets

Fig. 10. Comparing the approximate HSP Search by HNSW to the exact HSP search by the HHSP Test. The point labels on the curves correspond to the size of the neighborhood returned by HNSW for the approximate HSP.

The results show that, while there is no advantage to the HHSP on uniformly distributed data, the approximate method saturates performance while HHSP is able to reach 100% recall at a reasonable time. A similar result is shown for the high-dimensional Forest data, but not for the low-dimensional LA data where there is negligible difference.

Improving Graph-Based Nearest Neighbor Search: Since the HNSW achieves state-of-the-art results as an approximate monotonic graph, there remains a question as to whether its performance may be improved by being a fully, exact monotonic graph. To showcase the impact of the exact HSP neighbors, we replace the HNSW links on each layer with the exact HSP links. Figure 11 shows the comparison between the HNSW and the HNSW with HSP links on a 6D uniform and clustered datasets with $N = 1,000,000$ points. The construction of the full HSP Graphs by the HHSP took just over 10 h, while it would take the brute force approach an estimated 200 d!

First, as guaranteed by the monotonic property of the HSP, Fig. 11(a,b) shows the HSP Graph ensure exact search for any member of the dataset, which is not the case for the original HNSW, especially in clustered data. Secondly, Fig. 11(c,d) shows that the HSP links provide a slight, yet modest improvement over the original HNSW links for queries that are not members of the dataset.



(a) Uniform (6D) (b) Cluster (6D) (c) Uniform (6D) (d) Cluster (6D)

Fig. 11. Comparing the HNSW to HNSW with links are replaced with the exact HSP links. (a,b) Recall for the nearest neighbor when dataset items are queried. The monotonic property of the HSP graph guarantees a perfect recall, but this is not always the case for the original HNSW. (c,d) Recall for the nearest neighbor when items not in the dataset are queried. Although the monotonic property of the HSP does not guarantee perfect recall in this case, we see it provides a modest improvement over the original HNSW.

Conclusion: This hierarchical approach outlines a fast, efficient method of finding the exact HSP neighbors of a query in a metric space: by the novel definition of the shifted generalized hyperplanes between two points, pivots are able to discard or retain entire groups of points as consideration for being an HSP neighbors. While approximate methods to the HSP are able to achieve good recall with fast search times, they miss the vital, long-range links essential to the monotonic property of the HSP. By constructing the exact HSP Graph on a dataset of one-million points, which is a feat in itself, we show that the monotonic property can improve the performance of graph-based approximate nearest neighbor search.

Acknowledgement. We gratefully acknowledge the support of NIH Award 1S10OD025181 and NSF award 1910530.

References

1. Aguilera-Mendoza, L., et al.: Automatic construction of molecular similarity networks for visual graph mining in chemical space of bioactive peptides: an unsupervised learning approach. Sci. Rep. **10**(1), 18074 (2020)
2. Ayala-Ruano, S., et al.: Network science and group fusion similarity-based searching to explore the chemical space of antiparasitic peptides. ACS Omega **7**(50), 46012–46036 (2022)
3. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: Proceedings of the 23rd ICML, pp. 97–104 (2006)
4. Bose, P., Morin, P., van Renssen, A., Verdonschot, S.: The θ5-graph is a spanner. Comput. Geom. **48**(2), 108–119 (2015)
5. Brin, S.: Near neighbor search in large metric spaces. In: VLDB, vol. 95, pp. 574–584. Citeseer (1995)
6. Chavez, E., et al.: Half-space proximal: a new local test for extracting a bounded dilation spanner of a unit disk graph. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 235–245. Springer, Heidelberg (2006). https://doi.org/10.1007/11795490_19
7. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recogn. Lett. **26**(9), 1363–1376 (2005)
8. Ciaccia, P., Patella, M., Zezula, P.: M-tree: an efficient access method for similarity search in metric spaces. In: VLDB, vol. 97, pp. 426–435 (1997)
9. Dearholt, D.W., Schvaneveldt, R.W., Durso, F.T.: Properties of Networks Derived from Proximities. New Mexico State University, Computing Research Lab. (1985)
10. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. Proc. VLDB Endowment **12**(5), 461–474 (2019)
11. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. **42**(4), 824–836 (2018)
12. Mamede, M.: Recursive lists of clusters: a dynamic data structure for range queries in metric spaces. In: Yolum, I., Güngör, T., Gürgen, F., Özturan, C. (eds.) ISCIS 2005. LNCS, vol. 3733, pp. 843–853. Springer, Heidelberg (2005). https://doi.org/10.1007/11569596_86
13. Marchand-Maillet, S., Chávez, E.: HubHSP Graph: effective data sampling for pivot-based representation strategies. In: Skopal, T., Falchi, F., Lokoč, J., Sapino, M.L., Bartolini, I., Patella, M. (eds.) Similarity Search and Applications: 15th International Conference, SISAP 2022, Bologna, Italy, October 5–7, 2022, Proceedings, pp. 164–177. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17849-8_13
14. Marchand-Maillet, S., Pedreira, O., Chávez, E.: Structural intrinsic dimensionality. In: Reyes, N., et al. (eds.) SISAP 2021. LNCS, vol. 13058, pp. 173–185. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-89657-7_14
15. Mirabal, P., Abreu, J., Seco, D., Pedreira, Ó., Chávez, E.: Boosting perturbation-based iterative algorithms to compute the median string. IEEE Access **9**, 169299–169308 (2021)
16. Moore, A., Gray, A., et al.: New algorithms for efficient high dimensional non-parametric classification. In: Advances in NIPS 16 (2003)
17. Navarro, G.: Searching in metric spaces by spatial approximation. VLDB J. **11**(1), 28–46 (2002)

18. Ruiz, G., Chávez, E.: Proximal navigation graphs and t-spanners. arXiv preprint arXiv:1404.1646 (2014)
19. Talamantes, A., Chavez, E.: Instance-based learning using the half-space proximal graph. Pattern Recogn. Lett. **156**, 88–95 (2022)
20. Yao, A.C.C.: On constructing minimum spanning trees in k-dimensional spaces and related problems. SIAM J. Comput. **11**(4), 721–736 (1982)
21. Yianilos, P.N.: Data structures and algorithms for nearest neighbor. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, vol. 66, p. 311 (1993)