Finding Similar Shapes: Locality Sensitive Hashing of Real World Geometry

Matt Schwennesen
College of Computing
Michigan Technological University
Houghton, United States
mjschwen@mtu.edu

Dr. Satish Puri
Dept. of Computer Science
Marquette University
Milwaukee, United States
satish.puri@marquette.edu

***Abstract*—Similarity searches are a critical task in much of modern computer and data science. Given the ever increasing size of data sets, exact nearest neighbor searches quickly become impractical, leading to approximate nearest neighbor searches. However, little effort has been spent to develop these capabilities for polygons and other geometric data. This work adapts locality sensitive hashing, a form of approximate nearest neighbor searches with strong theoretical properties, to search for similar polygons under Jaccard distance without resorting to length scans of the full data set. This is enabled by a new sketching method which closely approximates the Jaccard distance between the original polygons when hashed with respect to weighted Jaccard similarity. Built upon the traditional locality sensitive hashing framework, an implementation of the system proved moderately successful in limited testing but since represents a new approach to approximate nearest neighbor searches for polygons.**

***Index Terms*—nearest neighbor, locality sensitive hashing, jaccard distance, polygons, similarity search**

I. Introduction

Nearest neighbor searches are as incredibly common task within computer science and in particular data science. The ability to tell if two objects are similar, and furthermore know whether an object is the closest to a query object is vital in some of the foundational algorithms in data science, such as clustering, information retrieval, text mining and recommender systems. As the size of data sets grow, the ability to perform these searches quickly and efficiently becomes increasing critical to the usability of the system in practical applications. The brute-force approach requires comparing either every object in the data set to every other object (an $O(n^2)$ operation before adding in the complexity of the comparison itself) or scanning the entire data set to compare it to the query object, which is at least an $O(n)$ operation per query.

To facilitate the performance need in a world full of data, much attention has been placed lately on approximate nearest neighbor searches, where a small trade-off is made between the accuracy of the search for a larger reduction in computational time. While there are several methods for performing these approximate searches, such as graph based methods [1], [2] and quantization [3]–[5], it is locality sensitive hashing which provides some of the best theoretical guarantees. Loosely speaking, locality sensitive hashing is a process by which objects in the data set are hashed in such as way where similar objects are more likely to generate the same hash. See section II for a more formal discussion of locality sensitive hashing.

Moreover, most of the current work on approximate nearest neighbor operates on the traditional data types, namely vector based data sets most commonly under Euclidean distance. Polygons are one type of data for which this traditional framework does not readily adapt, but for which there are a multitude of potential applications.

Polygons, or more broadly speaking, *shape*, is something which could be integrated into other metadata classifiers used for example in lake or natural feature classification or even the incorporation of floor-plans into a predictor of house price. Classification based on shape can even be used independently. Many types of natural objects from lakes to cells have classes for which shape can be a strong predictor and this work would facilitate the creation of a clustering system using that information. The inspiration for this work comes from a potential application for identifying where photographs without location data were take after extracting a dominate geometric shape such as the horizon. With regard to map data, some modifications to this work would lead to a system which could track changes in the shape of natural features over time, such as shifting lake or river borders.

Clearly then, applications exist for a system which can search for similar shapes over large (potentially global) data sets. As explored more thoroughly in Section III, such a system does not yet exist within the literature. The contributions of this work is the initial development of such a system which meets the below requirements.

- Is sensitive to the Jaccard distance between polygons. Jaccard distance is a distance metric well suited and intuitively defined for polygons.
- Can find approximate nearest neighbors close to the actual nearest neighbors without resorting to lengthy

This material is based upon work supported by the National Science Foundation under Grant #1950826.

linear or quadratic scans of memory.
- Can scale to handle massive data sets while still performing queries in a reasonable amount of time.

Within the limits of this work, a system was designed and implemented which meets the first two of the criterion established above. Using the traditional locality sensitive hashing framework originally established in [6], a method to create first an intermediate "sketch" of the each polygon which can be hashed using the method proposed in [7] while preserving a maximizing approximation of the Jaccard distance between the original polygons. The preservation of the Jaccard similarity between the original polygon and the sketch created from it is critical to the viability of the system and the heart of the new contribution of this work. Within our limited tests, the system achieves mediocre accuracy but a better mean squared error which suggests that when a member of the nearest neighbors is not found, an nearby approximate one can be found.

II. Definitions

Formally speaking, the system is performing a nearest neighbor search.

Definition 1 (Nearest Neighbor Search)**.** *Given a set P of objects and a distance measure d, return a set $S \subset P$ such that for some query q, $d(q,s) \leq d(q,p)$ for all $s \in S$ and $p \in P - S$.*

In this case, P is a set of polygons and the distance metric d is Jaccard distance. Jaccard distance arises from a closely related definition, that of Jaccard similarity.

Definition 2 (Jaccard Similarity)**.** *Given two sets X and Y, the Jaccard Similarity is*

$$\mathcal{J}_s = \frac{|X \cap Y|}{|X \cup Y|} \tag{1}$$

The Jaccard similarity reaches its maximum value of one when X and Y are equal and zero when X and Y are disjoint. Logically, Jaccard similarity is not useful within the context of a nearest neighbor search since under that definition the nearest neighbors would have no elements in common with the query object. To conform to the definition of the nearest neighbor search, an object should have a distance from itself of zero. Thus, we arrive at the definition of Jaccard distance, in which all of the values are inverted by subtracting the similarity value from its maximum of one. Furthermore, the definition of both Jaccard similarity and Jaccard distance can be adapted to polygons by viewing them as the set of points within their boundary.

Definition 3 (Geometric Jaccard Distance)**.** *Given two polygons, A and B, the Jaccard distance between A and B, $\mathcal{J}_d$ is defined as*

$$\mathcal{J}_d = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

Where $A \cap B$ represents the intersection of the polygons and $A \cup B$ the union, as traditional for set operations. Likewise, $|C|$ represents the total area of some polygon C.

The classical definition of both Jaccard similarity and Jaccard distance only work when it is clear an element is within a set or not. As explored in Section IV-A, a variant of Jaccard similarity will be required which can operate on vectors of real numbers, effectively as multi-sets that allow fractional weights as well. In that case, the quantity of interest is effectively the ratio between the minimum and maximum weights on each element, expressed formally below. This definition comes from [7].

Definition 4 (Weighted Jaccard Similarity)**.** *Given two d-dimensional vectors $x, y \in \mathbb{R}^d$, $x, y > 0$, their weighted Jaccard similarity is*

$$\mathcal{J}_w = \frac{\sum_{i=1}^{d} \min(x_i, y_i)}{\sum_{i=1}^{d} \max(x_i, y_i)} \tag{3}$$

As a final note about the various Jaccard indices, all three terms are used frequently in this work and care must be used to distinguish between them.

Now a formal definition of locality sensitive hashing (LSH) can be given. LSH methods rely on a hash family which operates on the principle that objects which are more similar are more likely to produce the same hash.

Definition 5 (Locality Sensitive Hash Family)**.** *A family of functions $\mathcal{H} = \{h : P \to U\}$ is considered (r, cr, p_1, p_2)-sensitive for some distance measure d if for any $q, p \in P$*

- *If $d(q,p) \leq r$ then $Pr[h(q) = h(p)] \geq p_1$.*
- *If $d(q,p) > cr$ then $Pr[h(q) = h(p)] \leq p_2$.*

To use this definition within the context of a nearest neighbor search, values for the sensitivity of the family must to chosen so that $c > 1$ and $p_1 > p_2$ [8], however the exact values need not be known to create a functioning family of hash functions.

III. Previous Work

Locality sensitive hashing was first introduced in 1998 by Piotr Indyk and Rajeev Motwani in [9] and has since become a popular tool for approximate nearest neighbor searches. Their method used what is known as a "ring-cover tree", where ring nodes geometrically represent hollow spheres and cover nodes are geometrically layered spheres with a bounded distribution of points within each layer. The ring-cover tree facilitated the creation of an approximate nearest neighbor algorithm operating in $\tilde{O}(n) \times O(1/\epsilon)^d$ preprocessing and $\tilde{O}(d)$ query time [9].

Later work removed the need for ring-cover trees via the introduction of a second layer of hashing [6], which has become the classic LSH framework. A number of locality sensitive hashes are generated for each object in the data set and the query process is to take the union of all other objects which share one or more locality sensitive hashes with the

query and perform a linear scan over this subset of the data. To help with the query process, after the locality sensitive hashes are generated they are hashed again and inserted into a hash map for constant time retrieval during the query process. Unlike the sensitive hashes, these hash maps use a standard hash function designed to generate a uniform distribution over the buckets of the hash map [6]. It is worth noting that [6] using chaining on their hash maps for a theoretical analysis, but in their implementation only allow for the chain to reach a fixed length so that it can be modeled with an array. After the chain length has been exceeded, do not add any more LSH hashes. This principle works "since it will be added to some other index with high probability. This saves us the overhead of maintaining the link structure" [6], however it is not employed in the implementation accompanying this work.

Both [9] and [6] developed applications for high dimensional vector data compared under the l_2 norm or Euclidean distance. However, LSH found success in text data while searching for similarly documents. Explored thoroughly in [10], this process replaced l_2 distance with Jaccard similarity by viewing each document simply as a set of words. Thus each word is either an element of a document or not. The creation of each LSH hash is dependent on the generation of a min-hash, the smallest possible hash and generally a single number or set element [10]. For sets of words, build a characteristic matrix where each set has one column and each element of the domain a row. Given a random permutation of the rows of the characteristic matrix, the min-hash value is the first element which is a member of each set when reading down the permuted characteristic matrix [10]. The useful application here is that the probability of the min-hashes of two sets being equal is the Jaccard similarity of the sets [10]. One final useful concept from [10] is that of the banding technique they employ on a signature matrix. However, it is not required to view the technique within that context.

Suppose that two sets X and Y have a Jaccard similarity of j_s. Using the min-hashing process from [10], the probability that any given min-hashes between X and Y are equal is j_s. If r min-hashes are generated for each X and Y, the probability that they are all equal is j_s^r and the probability that at least one is different is $1 - j_s^r$. Now if that process is repeated n times, the probability that the hashes disagree in at least position within each hash is $(1 - j_s^r)^n$. So the probability that X and Y produce one pair of hashes which are equal is $1-(1-j_s^r)^n$ [10]. Recall that since the query process is interested in all pairs which share a hash, that probability directly translates into the probability of a candidate pair being examined in the query process. Controlling the hash length and the number of hashes generated enables the system to suppress pairs with low similarity and artificially raise the probability of collision between two sets based on their Jaccard similarity.

Two other major improvements have been made to the LSH process since its introduction in [9]. The first was multi-probe LSH, introduced in [11]. When using the l_2 norm for a distance measure, the authors devised an adjustment to the query process which let their LSH implementation look in multiple buckets of hash map by employing the ability to estimate which other hashes an object was most likely to generate. The end result was an LSH implementation which is five to eight fold more space efficient [11]. Unfortunately, this work was unable to take advantage of this massive improvement since no other piece of literature has adapted this technique to Jaccard based measures and the authors where unable to do so in a reasonable amount of time despite some effort being spent.

The other major LSH improvement is targeted specifically at Jaccard similarity based hashing and was first proposed in [12]. Recall that for the Jaccard LSH in [10] a characteristic matrix is permuted to compute a min-hash. The critical idea here is that it is possible to take the min-hash by finding the first element in a set by starting at the top of the matrix and moving down but also by starting at the bottom and moving up [12]. By pulling two min-hashes from each permutation, half the number of permutation functions need to be generated with an otherwise equal computational cost. As will be explained in Section IV-B, this improvement was also not used in this work since it uses the weighted Jaccard similarity hashing method provided from [7].

While there are no pieces of literature that we could find specifically on nearest neighbor searches for polygons, the adjacent area of trajectory analysis has seen recent research. Trajectory analysis seeks to find similar "trajectories", which can be either physical or semantic. A physical trajectory would be data collected from a GPS, which details where a person or other object of interest has been traveling. One inspirational paper was [8], which was able to process GPS data under both Fréchet and Hausdorff distances using a multi-layered data structure. This work is different from [8] in multiple important ways, including:

1) The usage of Jaccard distance rather that either Fréchet or Hausdorff distances since it is a more powerful measure for polygons rather than curves.
2) The usage of more traditional LSH data structures, namely a set of hash maps rather than something derived from the Multi Resolution Trajectory Sketch which does offer some distinct advantages over the traditional structures by being able to quickly ascertain the potential level of similarity and remove candidates at every resolution step.
3) Their usages of GPS route data. This difference is rather large. The authors of [8] used GPS data taken within from taxis in Rome and Porto which enforces orientation constraints (i.e. routes have to follow the options defined by the city streets).
4) Their method is limited to routes taken from the same geographic area by the ability to make meaningful conclusions about the behavioral patterns of people in a city whereas this work can and in fact relies upon centering each polygon in the data set to make meaningful conclusions.

Do note that the potential adaption of a data structure which

is derived from the Multi Resolution Trajectory Sketch would potentially be advantageous to this work.

On the other hand is semantic trajectories which disregard the physical places and shapes by grouping them into categories like 'school' or 'transportation', such as in [13]. While the algorithm presented in [13] does have the ability to compare trajectories from multiple different locations, it is otherwise incompatible with this work since it achieves this by disregarding all of the geographic data which is what this work is exclusively concerned with.

One final important previous work is [14], which uses four-color raster signatures of polygons to estimate the overlapping area of a polygon join (or union) operations. The four-color raster signature overlays a grid on the polygon and shades each grid cell as empty, weakly intersecting, strongly intersecting and full based on the percentage of that cell which is fulled by the polygon [14]. In essence, this paper is similar to this work however they were focused on approximate set operations on the polygons. Our sketching method arguably uses an extension of the four-color raster signature idea where instead of only having four colors, we have an infinite number of colors by using the exact fill ratio in the signatures for this work. Since set operations are the core geometric operations for this work, incorporation of [14] to estimates of area of intersection and union could expedite operations like taking the Jaccard distance between polygons during the query process at the cost of some accuracy.

IV. Methods

A. Polygon Sketches

The first step in the LSH process employed by this work is the creation of a polygon sketch. There is no known way to move directly from a polygon to a LSH hash without an intermediate step to generate a unique vectorized representation of the polygon called either a sketch or in some works like [14], a signature. This standardized representation, once generated, it what is used throughout the rest of the LSH process.

Before a sketch can be generated, two important preprocessing steps are required. First, each polygon $p \in P$ is centered so that its centroid is located at $(0, 0)$. This step facilitates the comparison of geographic features which are not overlapping in the real world, such as the bodies of water in [15], the data set used to test this work. Under some data sets, like the GPS data used in [8], this step might by undesired since objects in the data set could overlap without being centered and that might reveal useful connections. However, this work is focused on a general polygon approximate nearest neighbor search and the definition of both Jaccard distance and Jaccard similarity breaks down when it is assured that no overlap is possible (such as a lake within another lake).

After all of the polygons are centered, it is necessary to create a minimum bounding rectangle (MBR) over the set of all polygons P. The global MBR defines the area of interest for the entire LSH process. It is assumed that all of the input polygons will lay within the global MBR. Portions of input polygons which do not will be ignored implicitly, so the global MBR is the area of interest for the LSH process.

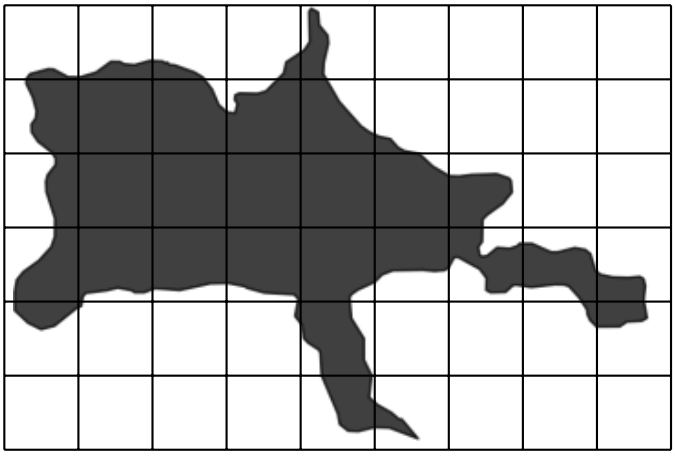

Fig. 1. Example polygon during the sketch process.

Once the global MBR is known, a uniform grid is generated over the area defined by it. Each grid cell will be transformed into one element of the sketch vector. While at one time the possibility of using a Z-order curves to transform the two dimensional grid into an one dimensional vector was discussed, ultimately the most important thing is that the order of grid cells within each sketch is the same and the extra complexity was not required. In Fig 1, the cells are processed left to right across the rows starting at the bottom row.

Between any two polygons, it is possible to define the fill ratio between a base polygon, the one being filled, and the overlay polygon which is doing the filling. Using the same set notation definitions from Section II, the following definition is reached.

Definition 6 (fill ratio)**.** *Between a base polygon B and an overlay polygon O the fill ratio of B, $f(B, O)$ is defined as*

$$f(B, O) = \frac{|B \cap O|}{|B|} \tag{4}$$

After all of the sketches have been computed, post-processing occurs. The post-processing consists of analysing how frequently each cell is used across the whole data set. Cells which are not used a set minimum number of times are removed from consideration. For the example polygon in Fig 1, those are likely to be the cells in the upper left corner which are far from the majority of polygons in the data set. Doing so achieves two goals. As pointed out in [7], this is a form of dimensionality reduction. It, by definition, reduces the total accuracy of the system by removing perfectly valid data in hopes of reducing the computational time needed to process the LSH hashes and query objects which is inversely related to the sparsity of the data [7]. This was also pointed out in [16], where by reducing from 50,000 commonly non-zero dimensions to only 200 a 150 times speed up was achieved with minimal loss of performance.

There is another benefit to removing infrequently used cells too. The runtime of the LSH hashing process (Section IV-B) is determined by the inverse of the sparsity of the data [7] so the more zeros in the sketch and longer the hashing process will take. Issues in the implementation actually revealed that in addition to increasing the time of the hashing process, the values of the hashes also increase and could lead to integer overflow errors when being inserted into the hash maps.

Notice that sketches produced using this method are comprised of floating point numbers between 0.0 and 1.0. Many other forms of LSH, including Jaccard based methods such as [10] and trajectory methods such as [8] use sketches which only contain discrete data. The classical form of Jaccard as defined in Section II (Definitions 2 and 3) cannot handle this distinction, forcing the use of a generalized variant of Jaccard to be used. The two common generalized variants are the weighted Jaccard from [7] and a probabilistic Jaccard introduced in [17] which is scale invariant. Both serve potential places within the context of polygon based nearest neighbor searches depending on the context. Strictly speaking, shape itself is scale invariant since for example, a square is a square regardless of whether the sides are 1 meter long or 0.5 meters long and under these conditions the scale invariant Jaccard would recognize that properly [17]. However, this work does not assume scale invariance for polygons so, just as the same side rotated is considered different, so too is the same shape but larger or smaller, leading to the decision to use the weighted Jaccard definition from [7] as given in Definition 4.

B. Locality Sensitive Hashes

The locality sensitive hashes are computed using the sketches described in the previous section rather than the polygons directly. Under these circumstances, the most important property of the sketches is that when operating under the weighted Jaccard similarity defined in Definition 4 is that the Jaccard distance of the original polygons is persevered.

Theorem 1 (Jaccard Similarity Preservation)**.** *Given two polygons A and B, along with their sketches s_A and s_B respectively, then*

$$\mathcal{J}_s(A, B) = c \cdot \mathcal{J}_w(s_A, s_B) \tag{5}$$

For some $c \geq 1$. Furthermore, as the resolution of the grid which produced the sketches increases and the area of each cell corresponding decreases, c approaches one.

Proof. Consider the contribution of one corresponding pair of elements in a pair of sketch vectors from polygons A and B, which is also two polygon fragments within the same grid cell. Let that grid cell have index i, denoting the fragments within cell i and A_i and B_i respectively. The Jaccard similarity between A and B is

$$\mathcal{J}_s(A, B) = \frac{\sum_i |A_i \cap B_i|}{\sum_i |A_i \cup B_i|} \tag{6}$$

or the sum of the area of intersection within each cell over the sum of the area of union within each cell. Each grid cell i contributes some fraction of area to both the intersection of the two polygons and the union of the two polygons according to the cases below.

1) Neither A nor B existing within that cell. Obviously the contribution of this cell is zero for both the intersection and union.
2) One of A or B existing within that cell. It is not possible for this cell to contribute to the intersection, but the area of whichever polygon exists within this cell is contributed to the union.
3) Both A and B exist within that cell. Examining Fig 2 reveals the minimal and maximal cases for the contributions.
    a) If A and B both exist within the cell but do not overlap, then zero is contributed to the area of intersection and $|A_i \cup B_i|$ is contributed to the area of union.
    b) Conversely, without lose of generality letting A be the red polygon, $|A_i|$ is added to the area of intersection while $|B_i|$ is added to the area of union.

Any other possible combination of intersections between A and B in cell i must fall between cases 3.a and 3.b. Case 3.b operates on the assumption that A is the smaller polygon, but that is not always the case, it could be B. Thus the more robust way to express that case is that $\min(|A_i|, |B_i|)$ is contributed towards the area of intersection and $\max(|A_i|, |B_i|)$ towards the area of union. Since those area are accurately recorded in the sketches of the polygons, this is analogous to $\min(s_{Ai}, s_{Bi})$ for the area of intersection and $\max(s_{Ai}, s_{Bi})$ for the area of union.

Substituted back into (6), if the assumption is made that every cell which falls into case 3 also falls into case 3.b, it becomes

$$\mathcal{J}_s = \frac{\sum_i \min(s_{Ai}, s_{Bi})}{\sum_i \max(s_{Ai}, s_{Bi})} \tag{7}$$

which matches the definition of weighted Jaccard similarity given in (3). By always maximizing the numerator and minimizing the denominator when both A and B exist within a cell, the contribution of each cell i is as close to 1 (the maximum value of $\mathcal{J}_s$) as possible, thus (7) acts as a maximizing approximation of $\mathcal{J}_s(A, B)$ which exceeds it by some factor $c \geq 1$.

The size of each cell i controls the value of c. By making the assumption that all cells follow case 3.b, error is introduced into the estimate for that cell. However, by subdividing that cell into more cells, (and assuming that the polygon is not fractal, which holds for all practical applications for this work), the error is reduced as a more accurate representation of actual relationship of the polygons is encoded into an increasingly long sketch. As the number of cells approaches infinity, the number of points within each cell decreases. Should the number of cells reach infinity, then each cell would contain a single point. Under the continuing notion of polygons as a set of points, then each point would individually be evaluated as a

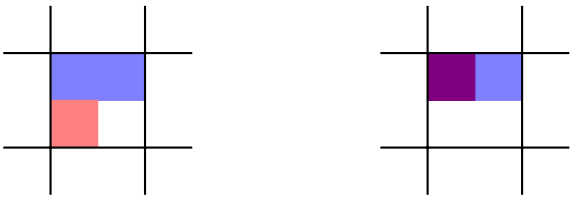

Fig. 2. Minimal and maximal possible contributions to the Jaccard similarity of two polygons within a single grid cell. Note that for the right image, the purple area shows the red fragment moving to overlay the unchanged blue area.

member of A, B, both or neither. This is the exact definition of the set based Jaccard similarity and in that case the weighted Jaccard modelled in (7) would have to equal the value given by (1). □

The weighted Jaccard similarity hashing process is adapted from [7] and produces hashes so that

$$Pr[h(s_A) = h(s_B)] \propto \mathcal{J}_w(s_A, s_B) \tag{8}$$

With the proportion depending on the length of the hash. Suppose that each sketch has D elements in it. Take the interval $[0, D]$ and divide it into D regions, one each of length one for each element of the sketches. In [7] they vary the width of each region according to the ceiling of the maximum value of that element across all sketches. Since each element of the sketch represents a fill ratio from Definition 6, the maximum possible value is 1 and the ceiling of any element which does not equal 1 will also be one. By exploiting this property, is it possible to eliminate two auxiliary hash maps from the hashing process.

Then, for some sketch s, each of the cells is filled in the range $[i, i + s_i]$ for all values of i, creating a series of shaded and unshaded regions such as Fig 3. Using a uniform random number generator, generate numbers across the range $[0, D]$ until one number falls within the shaded region. The number of attempts for that to happen in a weighted min-hash (recall the discussion of min-hashes from Section II) which respects the weighted Jaccard similarity between sketches [7]. The pseudo-code for this is given again below, but for a complete analysis of this process, refer to [7]. The seed array is used to maintain consistency across the polygons. In order for the LSH hashes to hold (8) each ith min-hash has to be generated with the same sequence of random numbers [7].

The difference between the pseudo-code here and that of [7] is that they require a separate function called $ISGREEN$ to determine if the generated number is in the shaded region. Since our data has additional properties, namely that the maximum value of each element of the sketch is 1, that is not required. The generated number r is in the shaded region if its decimal component is less than the decimal component of element $\lfloor r \rfloor$ leading to the **if** statement condition on line 6 of Algorithm 1. Without this property the only way to determine which element's range r fell into would be to use a hash map like in [7] with hash maps $CompToM[\,]$ and $IntToComp[\,]$.

Algorithm 1 Weighted min-hash, adopted from algorithm 3 of [7]

Input: Vector s, k, $seed[\,]$
Output: LSH hash of length k respecting weighted Jaccard similarity

1: Initialize $Hashes$ to all 0s
2: **for** $i \leftarrow 1$ **to** k **do**
3: $\quad randomseed \leftarrow seed[i]$
4: $\quad$ **while** true **do**
5: $\quad\quad r \leftarrow \text{Uniform}(0, D)$
6: $\quad\quad$ **if** $r < \lfloor r \rfloor + s[i]$ **then** ▷ r is in shaded region.
7: $\quad\quad\quad$ **break**
8: $\quad\quad$ **end if**
9: $\quad\quad Hashes[i] \leftarrow Hashes[i] + 1$
10: $\quad$ **end while**
11: **end for**
12: **return** $Hashes$

Fig. 3. Example min-hash generation regions

C. Hash Storage and Access

Once the hashes are generated, they must be stored to facilitate easy access during the query process. Unlike the original LSH paper [9] which used ring-cover trees, this work follows must closer to [6]. The data structure which is best for this task is itself a hash map which uses LSH hashes as key and the set of polygons which generated that hash as values. As explained in [6]:

> Thus, we use two levels of hashing: the LSH function maps a point p to bucket $g_j(p)$, and a standard hash function maps the contains of these buckets into a hash table of size M.

Which solves several practical issues with trying to use the LSH hashes directly as a hash value in the hash maps regarding collisions between hashes. There are two prevalent methods to deal with collisions, chaining and probing. However neither provide good solutions for use directly with the LSH hashes due to increased insertion time for probing and risk of losing the distinction between hashes in the bucket for chaining.

By using the LSH hashes in the role of keys, it is possible to use chaining without the problem of being unable to preserve the relationship between the hashes and polygons since the hash is part of the data, not merely a tool to operate the map. It is also advantageous given that the distribution of LSH hashes is unknown. If many LSH hashes are concentrated in a cluster (which is possible if the underlying data has similar similarities), the addition of a regular hash function designed to create as uniform a distribution as possible will mitigate the chances that the data in the hash map is wildly unbalanced.

Adapting the algorithms from [6] produces the hash map construction process given in Algorithm 2.

Algorithm 2 Hash map construction, adapted form Fig 1 of [6]

Input: A set of polygons P and l, the number of hash maps.
Output: A set of hash maps populated T.

```
for i ← 1 to l do
    Initialize table T_i with a random hash function.
end for
for i ← 1 to |P| do
    for j ← 1 to l do
        hp_i ← Hash(p_i)                ▷ Take the LSH hash
        T_j.insert(hp_i, p_i)
    end for
end for
return Tables T
```

Once the hash maps have been created and populated, the query process can begin. Querying LSH tables is typically given as taking the union of each bucket in which the query object would be placed, [6], [10], without inserting it since that would technically be a data leak. However, taking the union is an expensive operation which interacts with neighbors of every distance so long as they share a hash. It is likely that there are many more neighbors in the union of all the buckets than the query is searches for while only duplicates among that set of returned neighbors is a problem.

As an alternative to taking the union of all the buckets, it is possible to use a max-heap to maintain only the set of nearest known neighbors during the search. This would also facilitate returning not just a set of approximate nearest neighbors, but a sorted list of approximate nearest neighbors. The max-heap allows for constant time peaking at the top of the heap were neighbors are stored as neighbor pairs using their Jaccard distance as a key and the polygon itself as a value. Any neighbor pair which has a distance greater than the distance at the top of the heap can be immediately disregarded.

However, if the distance for the current neighbor pair is less than that of the maximum, it is not assured that this is a new neighbor pair to be added to the heap. Recall that the higher the Jaccard similarity, the more likely the LSH hashes are to match [7] so the closest neighbors are much more likely to be in the same bucket as the query shape multiple times. So if the distance for the current neighbor during the query process is less than the top of the heap it is merely a possible new neighbor and more extensive checks are required to verify this before adding it to the heap. First the heap is scanned for any neighbors with the same distance from the query as the potentially new neighbor. Since the distance from the neighbor to the query does not change, if there are no matches in the Jaccard distance of the current members of the heap, the neighbor is considered new and added. Even if an exact match in the Jaccard distances *is* found, that does not guarantee that the current member of the heap and the potential member of the heap are the same. It is possible for two distinct polygons to have the same Jaccard distance from the query, so the potential new near neighbor is added to the heap if and only if its Jaccard distance from the match is also non-zero. This process is summarized in Fig 4 and a pseudo-code algorithm is given in Algorithm 3.

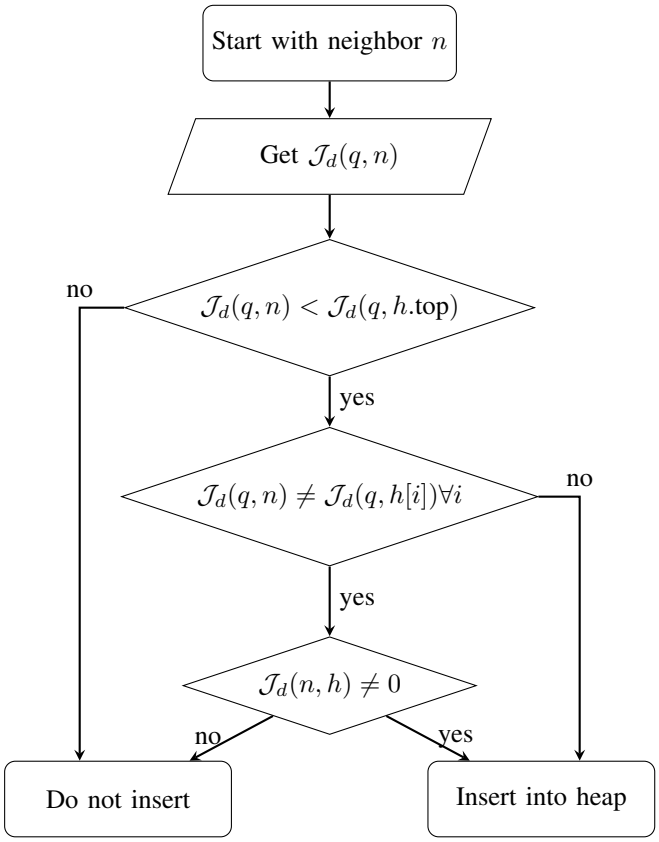

Fig. 4. The process of determining if a neighbor is a new near neighbor.

V. Experiments

A. Implementation

These algorithms were implemented in C++20 using the GEOS [18] library. GEOS offers a comprehensive set of features such as the ability to read and write one of the standard formats for encoded polygons, Well Known Text (WKT) which is what the data set [15] was encoded in. In addition, GEOS has a C++ API and a C API [18]. This work used the re-entrant C API to create a multi-threaded implementation which unfortunately had some instabilities which prevented comprehensive testing (see Section V-C) despite working well most of the time.

Work was planned to be divided into three types of threads,

1) *Input Threads*: Read the WKT encoded polygons, filter the list to exclusively contain polygons, and center the polygons and track a local MBR which is then combined with the results of the other input threads to produce the global MBR.
2) *Construction Threads*: These were the workhorse threads of the project. The Construction threads created the sketches of the polygons. Then, to enable infrequently used cells from the grid created over the

Algorithm 3 LSH query, adapted from Fig 2 of [6]

Input: A query point q, $seed[\ ]$, K, the number of approximate nearest neighbors
Output: K or less approximate nearest neighbors

```
S ← heap(∅)
for i ← 1 to l do
    h_q ← Hash(q)
    current ← T_i[h_q].head
    while current.next ≠ ∅ do
        dist ← J_d(q, current.shape)
        if S.size < K then
            S.insert(dist, current.shape)
            continue
        end if
        if dist < S.peak().key then
            newNeighbor ← true
            for neighborPair in S do
                if neighborPair.dist = dist then
                    if J_d = 0 then
                        newNeighbor ← false
                        break
                    end if
                end if
            end for
            if newNeighbor = true then
                S.pop()
                S.insert(dist, current.shape)
            end if
        end if
    end while
end for
return S.toSortedList()
```

global MBR to be removed, the threads wait at a barrier until all sketches have been generated. Upon arrival of the last thread to the barrier, the main thread triggers and cleans the grid by removing cells which do not meet a set minimum threshold. Next the construction threads have to revise the sketches to remove the cells which were purged. Finally they loop over the trimmed sketches, LSH hash them and insert into the hash maps as described in Algorithm 2.

3) *Query Threads*: Once the hash maps have been populated, the query threads read in a set of query files, center the polygons and sketch them. No revision is needed here since the grid cells have already been pruned. Once the sketches have been generated, Algorithm 3 is used to query the set of hash maps for the nearest neighbors.

Due to time restrictions during the REU, the query portion was not threaded although the code was structured so that this would be easily implemented by conforming to the threading interface used for the input and construction threads.

A brute-force all-pairs search was also implemented and used as a ground truth for the tests. Fig 5 gives a visual overview of the threads and how they interact.

B. Data Sets

The data set used to test the C++ implementation is all bodies of water from OpenStreeMap [15]. The entire data set is 9.1 Gigabytes and houses approximately 8 million records. Well known text encompasses many types of geometry, including polygons but also line strings, multi-polygons and geometry collections which are nested structures comprised of the other WKT entities. Our implementation only operates on the bodies of water defined as a single polygon, however the true number of operable shapes in the data set is unknown since the entire set was never read into memory.

Using the UNIX `split` command in the GNU core utilities, the data set was first broken into 1000 smaller sections in a round-robin manner to prevent all of the bodies of water in any given split to be concentrated in one geographic area. These files were manageable for the fully concurrent mode of operation, but still too large for the single threaded execution so one of those files was further `split` into 20 sections (thus approximately one five hundred thousandth of the whole data set). Once read and filtered down to just the polygons, this file contained 373 lakes from all over the world. For a test set, a similar process was followed which resulted in a file containing 29 polygons. It was important so that the test would complete within a reasonable amount of time that the query set be small.

An example of a query is given in Fig 6 along with the ground truth for that body of water found using the brute-force method.

C. Performance

A small grid search over four hyperparameters was conducted using the small data slice described in the previous section. The four parameters are described below.

- *Grid Size*: The number of cells per side of the global MBR. As explored in Section IV-B, specifically Theorem 1, higher grid resolution will cause the sketches to be a better approximation of the Jaccard similarity between the original polygons. However, it is suspected that if the resolution is too large, the effect of the grid threshold parameter will prune away too much of the grid. This is backed by seeing that the accuracy peaks at the middle grid size option.
- *Grid Threshold*: The number of times a cell must be at least partially filled to avoid being pruned. This parameter is a necessary evil. Any value greater than zero will reduce the accuracy of the LSH search, but it significantly reduces the run time of the search and keeps the values in the hash smaller which is important since overflow has been observed while inserted into the hash maps.
- *Number of Hash Maps*: The number of hash maps, also the number of times each polygon is hashed. Strictly speaking, a larger number of hash maps should always lead to better performance but it also results in the query process having to scan more the of total data set since

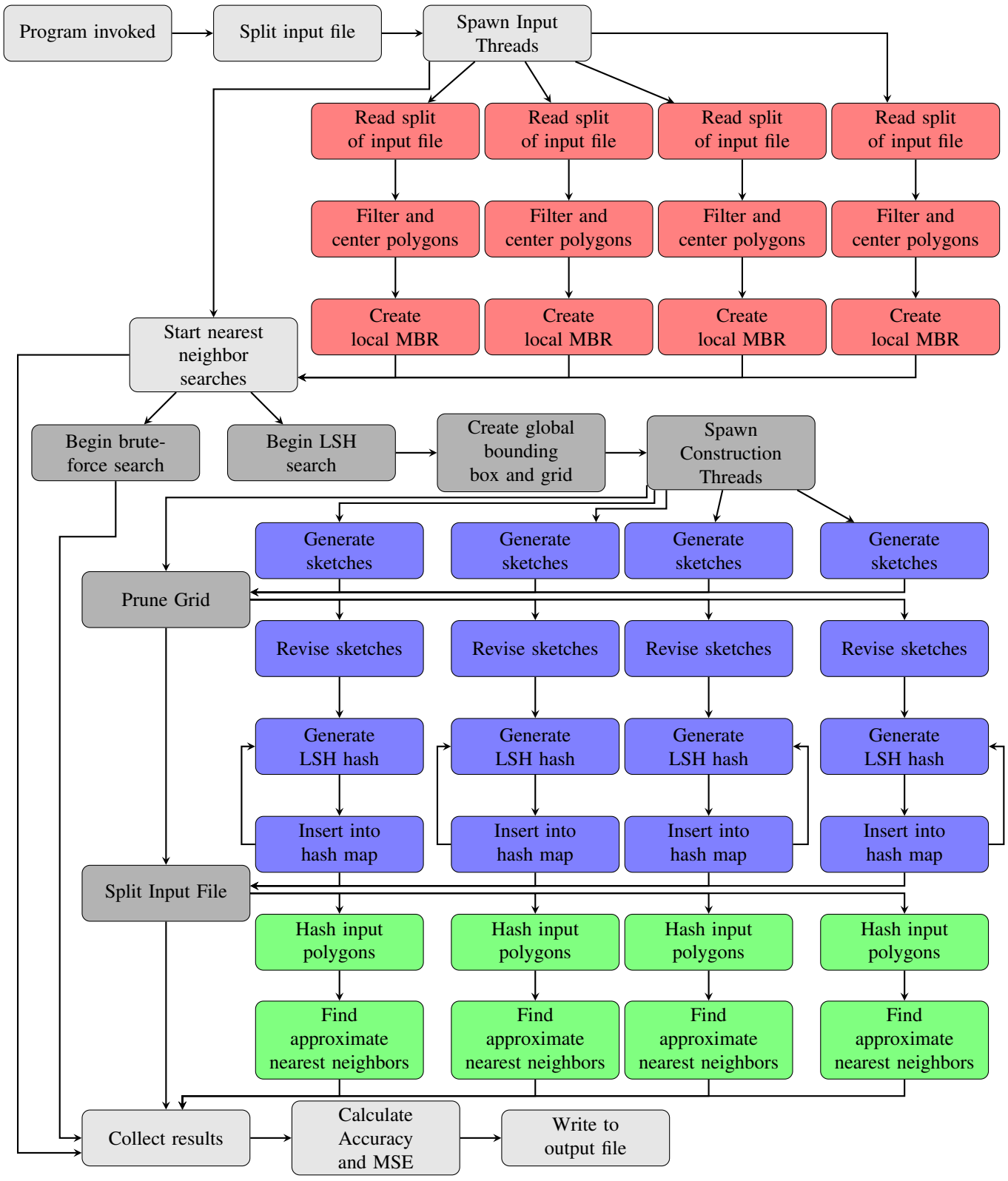

Fig. 5. Overview of the threading scheme used in the threaded implementation of this work. Input threads in red, Construction threads in red and Query threads in green.

Fig. 6. Example LSH query results compared to the brute-force results.

more hashes is more opportunities for distant shapes to produce the same hash as the query by chance.

- *LSH Hash Length*: The number of min-hashes used to create a single LSH hash. This hyperparameter directly influences the level of sensitivity to high and low Jaccard similarities. Longer hashes suppress the probability that shapes with low similarity produce the same hash by chance, which means that if it is too high it is possible that the system becomes only sensitive to neighbors closer than the actual set of nearest neighbors and suppress valid and correct output. Thus it is also data set dependent.

Results from the grid search are below. Each hyperparameter set is given as well as the accuracy and mean squared error (MSE) when compared to the ground truth as found by the brute-force algorithm. Accuracy is the proportion of approximate nearest neighbors returned by the LSH search which are actually part of the set of true nearest neighbors. Possible values range from 0 to 1 with higher values meaning better performance.

TABLE I
HYPERPARAMETER RANGES FOR SEARCH 1.

	Grid Size	Thres.	# Maps	Hash Len.
Min	20×20	1	10	2
Max	30×30	3	15	8
Step	$+5 \times +5$	2	5	3

TABLE II
BEST RESULTS FROM SEARCH 1

Grid Size	Thres.	# Maps	Hash Len.	Accuracy	MSE
25×25	3	15	5	0.218391	0.041997

Mean squared error is a measure of how approximate the approximate nearest neighbors are. It is calculated as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{9}$$

Since the tests asked for the three closes neighbors, the MSE for each query was processed so that the closest approximate nearest neighbor was only compared to the closest neighbor, the second approximate nearest neighbor the second nearest neighbor, etc. The minimum value is zero which indicates no error between the approximate nearest neighbors and the nearest ones with no upper bound.

The hyperparameter ranges are given in Table I and the best results from that search in Table II. While the accuracy is not high, the MSE is also rather low. This would seem to suggest that while the LSH search is not finding a lot of the actual nearest neighbors, the polygons that it *is* finding are relatively close to the true nearest neighbors. It is likely that this is a feature of the lakes data set [15] which is anecdotally supported by Fig 6, but again due to time restrictions, that hypothesis has not been formally tested.

The overall trend in the results from the first search was that a grid size of 25×25 was the more performant, so second search was conducted which fixed the grid size and explored the other parameters more thoroughly, given in Table III. As seen in Table IV, increasing the number of hash maps and thus the number of times each polygon is LSH hashed was able to raise the accuracy and lower the MSE which is an improvement on both accounts. This is suggestive that much more extensive testing is required to truly capture the capabilities of this system.

TABLE III
HYPERPARAMETER RANGES FOR SEARCH 2.

	Grid Size	Thres.	# Maps	Hash Len.
Min	25×25	1	10	2
Max	25×25	7	25	11
Step	$+0 \times +0$	2	5	3

TABLE IV
BEST RESULTS FROM SEARCH 2

Grid Size	Thres.	# Maps	Hash Len.	Accuracy	MSE
25×25	3	25	8	0.229885	0.037041

VI. CONCLUSIONS

This work has presented a theoretical framework and rudimentary implementation of an approximate nearest neighbor search for similar polygons under Jaccard distance using locality sensitive hashing. To the best of the authors current knowledge, our Jaccard Similarity Preservation theorem is novel and the heart of this method for leveraging existing Jaccard based LSH methods such as [7]. Our implementation proved to be sensitive to the Jaccard distance between polygons in the OpenStreeMap water data set [15] but performance was lackluster compared to results seen in other pieces of literature [8]. This is likely at least in part due to an inability to keep the multi-threaded implementation stable enough to perform a massive grid search over the hyperparameters described in Section V-C, but until the instabilities are removed it is impossible to verify.

Ultimately this is important foundational work, but it is far from over. There are several theoretical and implementation specific improvements which have the potential to increase model performance. Perhaps the largest improvement would be fixing the multi-threaded implementation which would would allow for much larger hyperparameter values to be used and searched over without inordinate execution times. One of the standard performance target is a accuracy of 0.9 or better which this work is still far from. The use of an R-tree during sketch creation would speed up that area of the implementation, was would implementing the multi-threaded query system which was designed. Converting from CPU programming to GPU programming also offers an exciting avenue to explore as well and would yield the largest performance gains, however that was not practical this summer.

On the theoretical side everything, the investigation of expanding the min-max LSH [12] and multi-probe LSH [11] techniques to be operable within the context of the proposed polygon LSH system would be significant gains. A final exciting way to continue the project is the use of an adaptive grid which is sensitive of the distribution of the polygons. As showing in Theorem 1, the resolution of the grid increases $\mathcal{J}_w$ becomes a better approximation of $\mathcal{J}_d$ at the cost of introducing more zero elements in the sketch for regions where the polygon does not extend to. An adaptive grid would selectively increase the resolution in areas where many polygons exist while the regions towards the edges which are not used would consume less space within the sketch, thus improving sketch accuracy without increasing the length of the LSH hashing process. This could be used to either replace or supplement the threshold system which is currently in place.

We were able to develop a system which is sensitive to the Jaccard distance between polygons as well as the theoretical work to support it. Much work remains but the progress from this summer REU will serve as a critical basis for future work.

REFERENCES

[1] C. Fu and D. Cai, "EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph," Dec. 2016. [Online]. Available: http://arxiv.org/abs/1609.07228

[2] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, pp. 461–474, Jan. 2019. [Online]. Available: https://dl.acm.org/doi/10.14778/3303753.3303754

[3] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[4] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization for Approximate Nearest Neighbor Search," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2013, pp. 2946–2953.

[5] Y. Kalantidis and Y. Avrithis, "Locally Optimized Product Quantization for Approximate Nearest Neighbor Search," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 2329–2336.

[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. Morgan Kaufmann Publishers Inc., Sep. 1999, pp. 518–529.

[7] A. Shrivastava, "Simple and Efficient Weighted Minwise Hashing," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/c2626d850c80ea07e7511bbae4c76f4b-Abstract.html

[8] M. Astefanoaei, P. Cesaretti, P. Katsikouli, M. Goswami, and R. Sarkar, "Multi-resolution sketches and locality sensitive hashing for fast trajectory processing," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, Nov. 2018, pp. 279–288. [Online]. Available: https://dl.acm.org/doi/10.1145/3274895.3274943

[9] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC '98*. ACM Press, 1998, pp. 604–613. [Online]. Available: http://portal.acm.org/citation.cfm?doid=276698.276876

[10] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, 3rd ed., Jul. 2019. [Online]. Available: http://infolab.stanford.edu/~ullman/mmds/book0n.pdf

[11] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07. VLDB Endowment, Sep. 2007, pp. 950–961.

[12] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang, "Min-Max Hash for Jaccard Similarity," in *2013 IEEE 13th International Conference on Data Mining*, Dec. 2013, pp. 301–309.

[13] C. Cai and D. Lin, "Find Another Me Across the World – Large-scale Semantic Trajectory Analysis Using Spark," Apr. 2022. [Online]. Available: http://arxiv.org/abs/2204.00878

[14] L. G. Azevedo, G. Zimbrão, J. M. de Souza, and R. H. Güting, "Estimating the Overlapping Area of Polygon Join," in *Advances in Spatial and Temporal Databases*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 91–108.

[15] A. Eldawy and M. F. Mokbel, "All water areas in the world from OpenStreetMap 2015," 2019. [Online]. Available: https://doi.org/10.6086/N1668B70#mbr=00bm,zzpg

[16] S. Ioffe, "Improved Consistent Sampling, Weighted Minhash and L1 Sketching," in *2010 IEEE International Conference on Data Mining*, Dec. 2010, pp. 246–255.

[17] R. Moulton and Y. Jiang, "Maximally Consistent Sampling and the Jaccard Index of Probability Distributions," in *2018 IEEE International Conference on Data Mining (ICDM)*, Nov. 2018, pp. 347–356. [Online]. Available: http://arxiv.org/abs/1809.04052

[18] GEOS contributors. (2022, July) GEOS computional geometry library. Open Source Geospatial Foundation. [Online]. Available: https://libgeos.org/