Protecting Databases from Malicious Discovery through Automated Similarity Queries

Fatih Altiparmak[1], Ali Şaman Tosun[2], Hakan Ferhatosmanoglu[1]
[1]Department of Computer Science and Engineering, The Ohio State University
[2]Department of Computer Science, The University of Texas at San Antonio
emails : {altiparm,hakan}@cse.ohio-state.edu, tosun@cs.utsa.edu

Abstract

Companies, hospitals, and research laboratories in certain domains have developed extensive databases, such as clinical databases, as part of their research or daily activities. The entities that have developed these databases may wish to lease or allow use parts of the database by external users. Due to the significant time and monetary investment in the development of the databases, and the proprietary or the private nature of the data itself, they may not want to sell or allow access to the entire database. However we show that such databases are vulnerable to reverse engineering using popularly employed similarity-based queries. We identify some important security issues related to k-NN search and investigate their vulnerabilities against users who try to copy the database by sending automated queries. We analyze two models for similarity search, namely reply model and score model. Reply model responds with the k tuples that most closely match the query according to some metric, and score model responds with only the score of similarity search which provides more power in preserving the privacy. For these models we analyze possible attack methodologies and develop strategies that can be used to detect the potential attacks. We state the limits of protection provided by each query response model, and also provide techniques to guard the database against malicious discovery.

I. Introduction

Collecting data and maintaining large databases can require a great expenditure of both time and monetary resources. Owners of these databases may wish to lease or allow use of their database without exposing the entire database to discovery. As such, the owners of such databases want to know the vulnerabilities in their systems. Bioinformatics databases provide an example of this limited user database access paradigm, because the database owner can benefit economically or through symbiotic research relationships by provided limited access to the database but does not want to provide access to the entire database because of privacy concerns, the intellectual proprietary nature of the data involved or resources expended in its development.

In this paper, we first discuss how the contents of a database can be copied using intelligent automated queries. We focus on similarity queries which is the common type of query in many modern data repositories and web-based search engines. We analyze the vulnerabilities of databases against automated similarity queries using two different models and discuss possible approaches to detect such attacks. These models are:

- *Reply Model.* Client sends vector x and database responds with the closest k vectors y_i $(i = 1...k)$. This is the model used in most of the current applications.
- *Score Model.* Client sends vector x and database responds with similarity score $\parallel x - y \parallel$, where y is the closest vector in the database to the x. This model is suitable for proprietary databases but requires development of distance functions which give the user a clear idea of how similar the vector is to query vector.

Both models are appropriate for medical and biological databases where gathering data is a costly process, and preserving privacy or intellectual proprietary is crucial. For example, consider a database

This material was prepared with the support of the U.S. Department of Energy (DOE) Award No. DE-FG02-03ER25573, of the National Science Foundation (NSF) Award No. CNS-0403342 and of the Pfizer Inc. Ali Şaman Tosun is supported in part by CIAS at the University of Texas at San Antonio through a grant provided by US Air Force. [*]An earlier version of this paper appeared in [49].

consisting of DNA sequence of people with cancer. A user can search this database to see if his or her DNA sequence is similar to the ones in the database. Based on the score, user can decide to see a doctor to investigate it further. Many common diseases are hereditary so such a system using the score model [22] would be invaluable. Also consider a clinical data warehouse, generated by a hospital or a clinical trials study, consisting of patients' blood and urine measurements. Such data contain not only private but also proprietary information, such as data about responses to a newly studied drug. For a set consisting of blood measurements where all records belong to a patient with a specific disease, such as Arthritis, using the score model database owners can let users search information without revealing the proprietary measurements. In this way, a user could determine if any person with the disease in the database is within some similarity threshold to their own measurements. Such information could be useful in determining the possibility of having the disease. Now, consider a data set that contains the records of all patients, both healthy and non-healthy, with any adverse conditions for each patient identified. Using the reply model, a user could get a consensus of the type of diseases the nearest k-patients have in terms of blood and urine measurement similarity and gain information about their own potential health state.

Using the two models, we investigate the following general issues in this paper :

- What is the best strategy a potential attacker can use to copy the whole database or to learn contents of the database?
- What can be done to prevent such attacks on the database?

A. Related Work

Privacy-preserving Data Management Over the last few years there has been considerable amount of interest on privacy-preserving database operations [5], [10], [2], [25], [6], [24] including access control, indexing, and mining. A somewhat related problem is *Private Information Retrieval* (PIR). In PIR [16], the server has an n bit information vector *x*, the client requests a bit x_i and the server returns the bit with the constraint that server do not learn the index of the bit. It has been shown that the solution to the PIR problem has a communication cost equal to transmitting the entire database. Recent approaches achieve polylogarithmic communication using different intractability assumptions [15]. However, proposed solutions are impractical for most large datasets. Most of the solutions to these problems make use of random vectors. Also, *all* the approaches presented in the literature for these problems make use of trusted third party anonymizers.

Secure-multi party computation is a closely related field to privacy-preserving database operations. It has been theoretically proven that secure-multi-party computation problem can be solved using circuit evaluation protocol [31], [53]. In circuit evaluation protocol, each functionality is represented by a boolean circuit and the parties run a protocol for every gate in the circuit. This approach is general but communication complexity is high. Using solutions based on circuit evaluation for special cases of multi-party computation is impractical and special solutions needs to be developed for efficiency reasons. During the last couple of years special solutions are developed for secure multi-party computational geometry [9] and privacy preserving statistical analysis [23].

Most of the privacy research focuses on privacy-preserving data mining which aims to enable mining without direct access to specific private information. Watermarking research [3], [4], [44], [43] embeds information into the database that can later be used to identify the owner. Cryptographic primitives are used for distribution of XML documents to clients that allows clients to locally process them without violating access controls [39]. Automated trust negotiation [48] establishes trust between server and client through the exchange of digital credentials. There is a growing need to address security issues in modern database applications that require extensive research to develop or involve intellectual proprietary data.

Inference Control Even if an access control policy is used, illegal data accesses via inference channels may occur and inference channels must be eliminated. In relational databases, integrity constraints including functional, multivalued, and join dependencies are important for inference control. There are two main approaches to detect and remove inference channels. The first approach removes inference channels by modifying the database design and by increasing classification level of some items [32], [38], [47]. The second approach eliminates inference channels during query processing [19], [21], [14]. For more information on inference control readers are referred to [27], [1], [34]. Above approaches are tailored for relational databases and are not suitable for emerging database applications such as high dimensional databases and nearest neighbor queries. Logic based representation and formal method based inference control can not be used for high dimensional databases since dependencies do not exist. In this paper, we use a geometry based approach to protect the databases from malicious discovery through similarity queries.

Similarity Queries Due to the nature of the data and the large quantity of information, traditional database queries such as exact match queries are largely being replaced by similarity-based queries. The degree of similarity between objects in a database is often quantified by a distance measure, e.g., Euclidean distance, operating on the multi-dimensional data objects or the feature vectors extracted from the data objects. For example, a user may pose a query over a medical database asking for X-rays that are similar to a given X-ray in terms of Euclidean distance of multi-dimensional texture feature vectors [36], [35]. 3D Shape histograms of proteins are used to identify their similarities [7]. Similarity query is usually implemented by finding the closest feature vector(s) to the feature vector of the query data. This type of query is known as nearest neighbor (NN) query [42] and it has been extensively studied in the past [30], [33], [51], [8], [11], [28], [12], [20]. A closely related query is the ϵ-range query where all feature vectors that are within ϵ neighborhood of the query point q are retrieved.

Bloom Filters Bloom filters are used in many applications in databases and networking including query processing [41], [18], [40], IP traceback [45], [46], per-flow measurements [17], [37], web caching [50], [26] and loop detection [52]. A survey of Bloom filter applications is given in [13]. A Bloom filter computes k distinct independent uniform hash functions. Each hash function returns and n-bit result and this result is used as index into a 2^n sized bit array. The array is initially set to zeros and bits are set as data items are inserted. Insertion of a data is accomplished by computing the k hash function results and setting the corresponding bits to 1 in the bloom filter. Retrieval can be done by computing the k digests on the data in question and checking the indicated bit positions. If any of them is zero, the data is not stored in the table (since storing the data would set the bits). If all the bits are set, the data is stored in the table with high probability. It is possible to have all the bits set by other insertions. This is called a *false positive*, bloom filter returns a result indicating the data is in the filter but actually it is not. However, bloom filters do not cause *false negatives*. It is not possible to return a result that says data not in filter whereas data is stored in the filter.

The rest of the paper is organized as follows. In Section 2, we analyze vulnerabilities associated with the reply model and discuss possible attack detection techniques. In Section 3, vulnerabilities associated with the score model and possible detection techniques are considered. We compare these two models in Section 4. We discuss attack detection in replicated databases in Section 5. We conclude and provide directions for future work in Section 6.

II. Vulnerabilities in the Reply Model

The reply model is the natural and the common way of implementing a similarity query. In the reply model upon receiving a query, the database responds with the closest k tuple(s) according to the Euclidean distance as the distance metric. In this section, we discuss some of the possible attacks that can be used to learn the contents of the database using the reply model for similarity query response. Then, we present approaches to detect such attacks.

A. General Scheme

We will first explain the attack on 1 dimension and then generalize it to higher dimensions. Assume a 1-dimensional continuous attribute with lower bound l_1 and upper bound u_1. The minimum distance between a point in the database and its k^{th} closest point in the database, c_k, is formalized as follows.

Definition: 2.1: $c_k = \min_x\{dknn(k,x)\}$ where $dknn(k,x)$ is the distance between x and k^{th} nearest neighbor of x.

```
Discovered_Points = $\emptyset$
$\alpha$ = $\lfloor \frac{u_1-l_1}{c_k} \rfloor + 1$
for $i$= 0 to $\alpha - 1$ do
    probe = $l_1$+$\frac{c_k}{2}$+$ic_k$
    $q$ = sim_search(probe)
    for $j = 1$ to $k$ do
        if $|probe - q[j]| \leq c_k/2$
            Discovered_Points.insert($q[j]$)
```

Fig. 1. General Probe algorithm for 1 dimension

The algorithm in Figure 1 can be used to replicate the contents of the database by using $\lfloor \frac{u_1-l_1}{c_k} \rfloor + 1$ equally spaced queries. The additional probe $(+1)$ is added to capture any potential data points lying on the upper bound of the attribute. Sample execution of the algorithm for k equal to 1 is given in Figure 2. Arcs denote the result of similarity search and x's on arcs denote that the most similar point found is not inserted into the data structure that contains the discovered points for the given probe. Triangles on the bottom depict the boundaries of the region being covered by the current query.

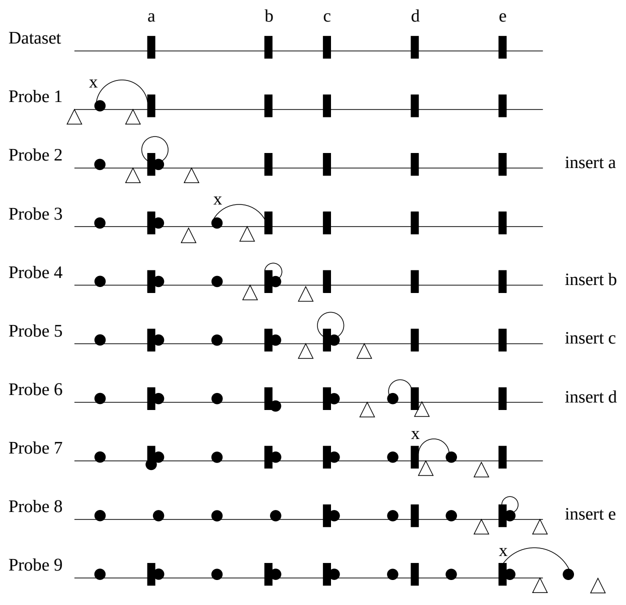


Fig. 2. Sample execution of probe algorithm, k=1

All nodes in this case are discovered using 8 probes (and guaranteed to be discovered in 9 probes). Sample execution for k equal to 2 is given in Figure 3. Not only the interval between probes but also number of probes is different from the $k = 1$ case. As the definition of 2-NN implies, the 2^{nd} closest neighbor is also returned together with the 1^{st} one. All nodes are discovered using 5 probes. As k increases, the interval between probes (c$_k$) will increase and this will lead a decrease in total number of probes.

We can optimize this algorithm and skip some probes if the returned k^{th} neighbor is multiple probe points away. Following lemma is based on the algorithm given in Figure 1.

Lemma 2.1: Probe algorithm discovers the whole database.

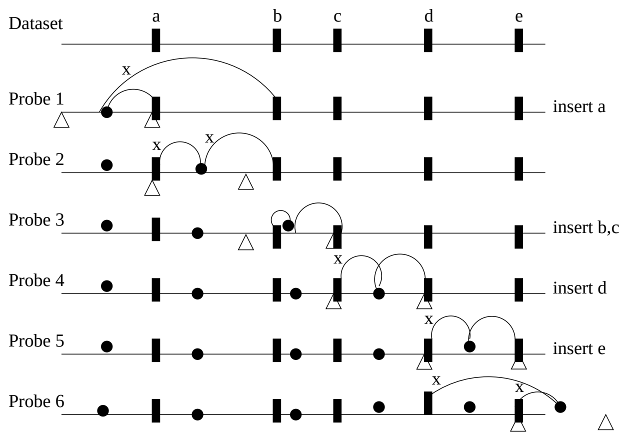


Fig. 3. Sample execution of probe algorithm, k=2

Proof: Consider the voronoi diagram of probe points. The voronoi region of a probe point p is the region which is closer to p than any other region. Every point is in some voronoi region since voronoi regions cover the whole space. Each voronoi region has at most k points since the maximum distance in a voronoi region is c_k. Therefore every point is returned as a result of one of the similarity searches. Hence the whole database is discovered. ■

The value of c_k in this database may not be known to the attacker. The attacker can pick some distance d and use the algorithm based on d assuming that the minimum distance to the k^{th} neighbor is d. Some points may not be discovered using the probe algorithm in this case. However we will still have an idea of what we might have missed in the database.

Lemma 2.2: Using probe algorithm with distance d, missed points are less than d distance away from some discovered point.

Proof: Consider voronoi regions. Voronoi regions cover the whole space. Points are missed if more than k points fall into one voronoi region. In that case, the k closest points to probe point is returned. Since the maximum distance in a voronoi region is d, missed points are less than d away from some discovered point. ■

B. Progressive Scheme

For some data sets c_k can be very small. This causes a very high number of probes to be required to learn the whole database. In this section we show a progressive probe algorithm which discovers the data in finer detail as the number of probes increases.

```
Discovered_Points = $\emptyset$
$\alpha$ = $\lfloor \frac{u_1 - l_1}{c_k} \rfloor + 1$
$levelno$ = $\lceil \log \alpha \rceil$
for $i = 1$ to $levelno$ do
    for $j$ = 1 to total number_of_nodes_in_level_$i$
        $q$ = sim_search($levelNode(i, j)$)
        if $q$ not in Discovered_Points
            Discovered_Points.insert($q$)
```

Fig. 4. Progressive probe algorithm for 1 dimension

The algorithm for progressive probing is given in Figure 4. The function $levelNode$ returns the j^{th} node in level i. Progressive probe uses the same probe points but in different order, therefore it also discovers the whole database. Total number of levels ($levelno$) is calculated based on the total number of probes.

For the sake of reader clarity, it may be appropriate to remind that total number of probes decreases as k increases. The probes at a given level are determined by the mod operation. Indices of the probes start from 0. In each level, we have probes whose indices are divisible by $2^{levelno-levelnumber}$ but not by higher powers of 2. As an example, for $k = 1$ we have 9 probes, hence $levelno$ is 4. In the first level we have points whose indices are divisible by 2^{4-1}. In the second level we have points whose indices are divisible by 2^{4-2} but not by 8. In the third level we have points whose indices are divisible by 2^{4-3} but not by higher powers of 2 and so on. The strategy used to compute the level of a single node for progressive scheme is summarized in Figure 5.

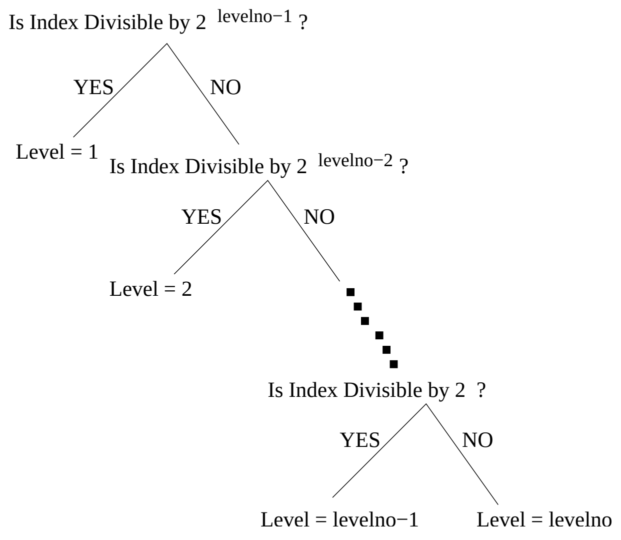

Is Index Divisible by 2 levelno−1 ?
YES
NO
Level = 1
Is Index Divisible by 2 levelno−2 ?
YES
NO
Level = 2
Is Index Divisible by 2 ?
YES
NO
Level = levelno−1
Level = levelno

Fig. 5. Computing the level of a single node for Progressive Scheme

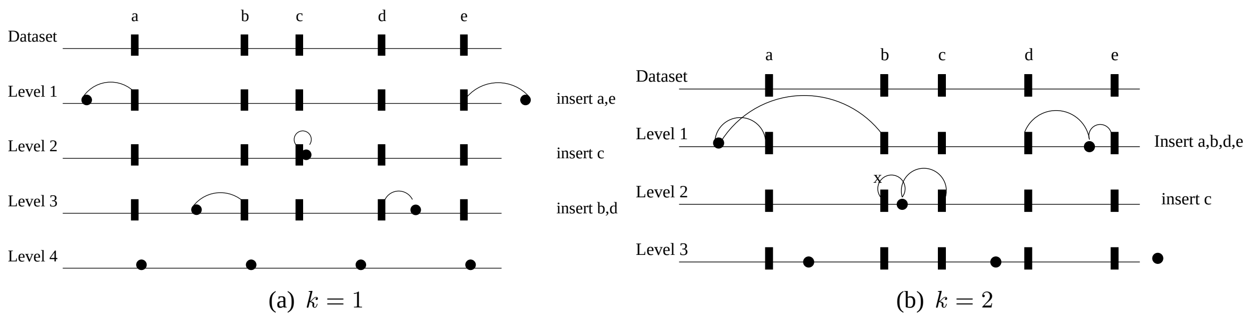


Fig. 6. Sample execution of progressive probe algorithm

Sample executions of the algorithm for $k = 1$ and 2 are given in Figure 6. The arcs denote results of the similarity search and x's on arcs denote that the point is already discovered. The progressive probe scheme distributes probes in the data space in a way that probes are more likely to find previously undiscovered tuples. Note that, in these example executions, the last levels do not introduce any new data and total number of probes to discover all nodes is 5 for $k = 1$ and 3 for $k = 2$. As a result, for equal number of queries to a non-skewed database, the progressive scheme will gather more information about the database than the general one.

The performance of the progressive scheme will degrade on highly skewed data. We propose an adaptive progressive scheme which will adaptively change itself according to the distribution of the data points. The general idea of adaptive scheme is to dig deeper into denser regions which are more likely to contain more points and various heuristics can be used for this purpose. For n dimensional region, 2^n corners and the center point are identified and k-NN query for each of these points is executed. The sum of the

```
$n$: dimension of database
$PQ$: priority queue
$e$,$e_2$,$e_3$: element (structure with $2^n + 1$ points)
$e$: initial element corresponding to whole region
$PQ$.push($e$)
while $PQ$ not empty
        $e_2$ = PQ.pop()
        for i = 1 to $2^n$ do
            $e_3$ = develop_region(i)
            if $e_3$.distsum() $<$ threshold
               PQ.push($e_3$)
```

Fig. 7. Adaptive progressive algorithm for 2 dimensions

distances from these points to their k^{th} nearest neighbor is computed. We use a priority queue and at each iteration, we find the region with smallest sum, divide it into 2^n subregions, find sum of distances for each subregion and push them into the priority queue. We use a pruning strategy to eliminate the regions that potentially contain no points. The algorithm for the adaptive progressive scheme is given in Figure 7.

C. Random Scheme

In the random scheme, random points in the space are picked and sent to the database as probe points. Although this scheme performs well initially, probabilistically a huge number of queries is needed to discover the whole database. The random scheme is ideal for uniformly distributed data but real data sets are usually highly correlated and clustered. Data clustered in dense chunks will be very difficult to discover using the random scheme.

D. Distributed Scheme

A powerful way of probing is to employ a distributed scheme. A number of sites can send independent probes, collect data and then construct the database by combining individually constructed databases. The probes can be clustered or de-clustered and each cluster or de-cluster can be assigned to a site.

E. Using Query Histories

To discover every point in the database, the space is divided into equally spaced voronoi regions. The edge of a voronoi region is selected such that the query sent from the center of the region will discover at least all the points in the region. This will result in some of the data points returned for more than one probe. It is possible to eliminate unnecessary probes by using the results of the earlier ones.

In Figure 8 squares and circles represent data points and probes respectively. Let r_i(i from 1 to k) be the result of probe p, then all the probes whose voronoi cells lie completely within the circle can be eliminated. This optimization can be formally stated as follows.

Lemma 2.3: Let p and q be two probe points. Assume that the result of probe p returns k points and $dknn(k, p) = s$. If the region of probe q lies completely in the circle with center p and radius s, then probe q is unnecessary.

Proof: By selection of probe points there are at most k points in the region of a probe point. This guarantees that all the points in the region are returned as a result of the query with the probe point. If probe p returns k points with $dknn(k, p) = s$ and the region of probe q lies completely in the circle with center p and radius s, then there is no data point in the region of probe q. Therefore, elimination of probe q does not cause any missed data points and probe q is unnecessary. ■

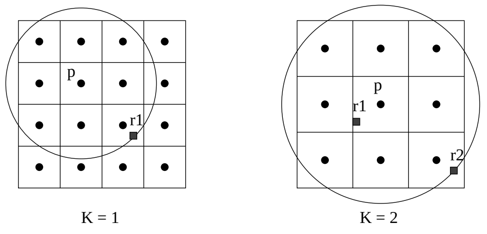


Fig. 8. Elimination of Unnecessary Probes

The above lemma can be applied by verifying that all four corners of the voronoi cell are in the circle with radius s and center p. An easier way is to use the bounding circle of the voronoi cell. Voronoi cells are constructed to make sure that at most k data points lie in the cell. Therefore, the bounding circle of voronoi cell has radius $c_k/2$ and we can use the following to test if voronoi cell lies completely within the circle.

Lemma 2.4: Let p and q be two probe points. Assume that the result of probe p returns k points and $dknn(k,p) = s$. If $d(p,q) < s - c_k/2$ then the region of probe q lies completely in the circle with center p and radius s

Proof: Let x be a corner of the region. By construction of bounding circle we have $d(q,x) = c_k/2$. By triangle inequality we have $d(p,x) <= d(p,q) + d(q,x) <= s - c_k/2 + c_k/2 <= s$. Therefore x is in the circle with center p and radius s. ■

F. Extensions to Higher Dimensions

Both the general scheme and the progressive scheme can be extended to higher dimensions. For example in 2 dimensions, the general scheme can send probes in order from left to right in each row and when a row is completed it can move on to next row. The number of probes to discover the whole database is given below, where c_k denotes the minimum distance to kth neighbor, u_i denotes upper bound of dimension i and l_i denotes lower bound of dimension i. Recall that the euclidian distance is used for the reply model.

Theorem 2.1: Probe algorithm requires $\left(\lfloor\frac{u_1-l_1}{c_k/\sqrt{2}}\rfloor + 1\right)\left(\lfloor\frac{u_2-l_2}{c_k/\sqrt{2}}\rfloor + 1\right)$ probes to discover all elements of a 2 dimensional database using the general scheme.

Proof: The range of the database is divided into cells of size $c_k/\sqrt{2}$ by $c_k/\sqrt{2}$. At centers of the cells we have probe points. Voronoi regions of probe points are also squares of size $c_k/\sqrt{2}$. Voronoi regions cover the whole range and we can have at most 1 point in a voronoi region. All the points in the database will be in some voronoi region therefore the whole database will be discovered. ■

Similarly probe algorithm can be extended to higher dimensions as follows.

Corollary 2.1: Probe algorithm in n dimensions requires $\prod_{i=1}^{n}\lfloor\frac{u_i-l_i}{c_k\prime}\rfloor + 1$ probes to discover all elements of a n dimensional database using general scheme where $c_k\prime$ is a linear function of c_k to guarantee that only one point will be in each voronoi region. The value of $c_k\prime$ is $\frac{c_k}{\sqrt{n}}$ in n dimensions.

The progressive scheme is better at discovering more of the database with earlier queries since it spreads the probe points. For higher dimensions, the level of a probe is calculated similar to the one dimensional case. The only difference is in the number of indices, which is equal to the number of dimensions, for a probe point . The total of the indices is used and the level of a probe is calculated by the modula operation. For example, for two dimensions modula of the sum of row number and column number is used for calculating the level of a probe. Note that probe indices for each dimension start from 0. As an example, $levelno$ for a two dimensional space divided into 5 columns and 5 rows is 4 ($\lceil$max row index + 1 + max column index + 1$\rceil$). The probe pattern of progressive discovery for this space is given in Figure 9.

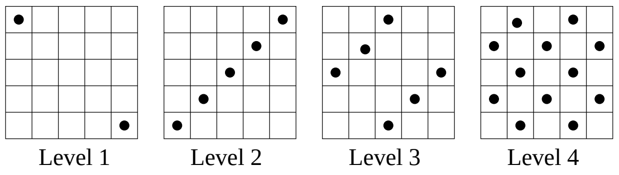


Fig. 9. Levels of Progressive Probing in 2 dimensions

G. Experimental Results

We implemented random probing, random probing with query histories, progressive probing with query histories and adaptive progressive probing. k-NN queries (for different values of $k\{1, 2, 5, 10\}$) were performed on two synthetic and two real-life data sets. There are three 2-dimensional datasets (shown in Figure 10) each of which represents a different type of distribution. The first data set is latitude and longtitude of road crossings in Maryland. This data set is a good example for uniformly distributed data. The second one is correlated data typically seen in time series data such as stock price movements [29]. The third data set has points mostly clustered in one region. For each of these three data sets, if the data set contained more than 1000 points, we used a subsampled version with 1000 points. The fourth data set is a clinical data obtained from a pharmaceutical company. The data contains measurements of 4 blood ingredients for 244 patients. Half of them are healthy patients and the rest are suffering from Arthritis. Since the minimum and maximum value each blood ingredient can take is known finding the l_i and u_i for each dimension is a trivial task.

The number of queries sent and the number of points discovered are given in Figure 11. For all strategies, as k increases, the total number of queries needed to gather whole database decreases. An interesting result is that for the same data set, shape of the curves remains same as k increases. The only difference is the total number of queries that needs to be sent to gather same amount of data points.

The probing algorithms that take advantage of the information provided by early query results perform well for any k. Adaptive probing considers the results of the queries corresponding to the corners and the center of the region while digging into the regions, however, it has to send these queries without making any check if they are necessary or not. Hence, it requires more probes to discover the whole database. For two dimensional data sets, random probing performs initially better than the adaptive progressive one, especially on data sets with nearly uniform distribution of points. Adaptive progressive probing performs similar to random probing initially but it discovers points with fewer probes as the number of queries increases.

Since the four dimensional data set is sparse, random probing with query histories performs better than progressive probing with query histories. Random probing performs better than the progressive probing with query histories for lower number of queries but the latter needs less probes to discover whole data set. The performance of Adaptive progressive probing is similar to random scheme. As the number of queries

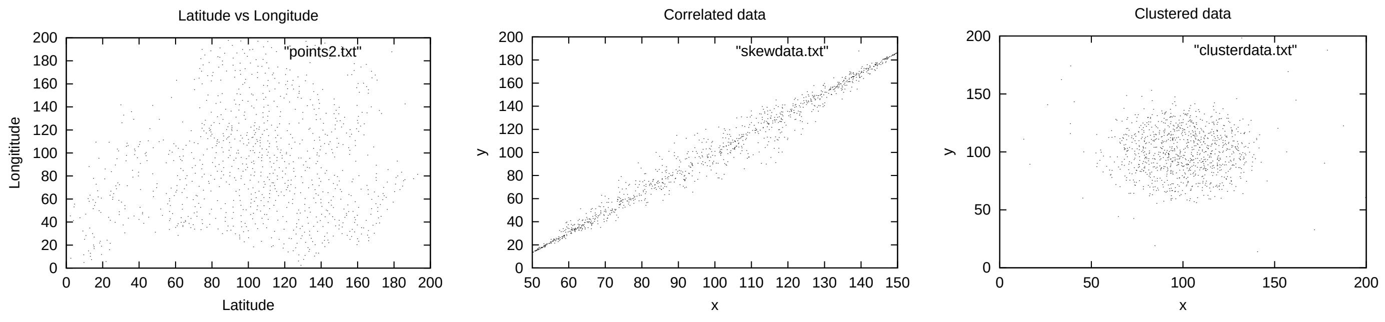
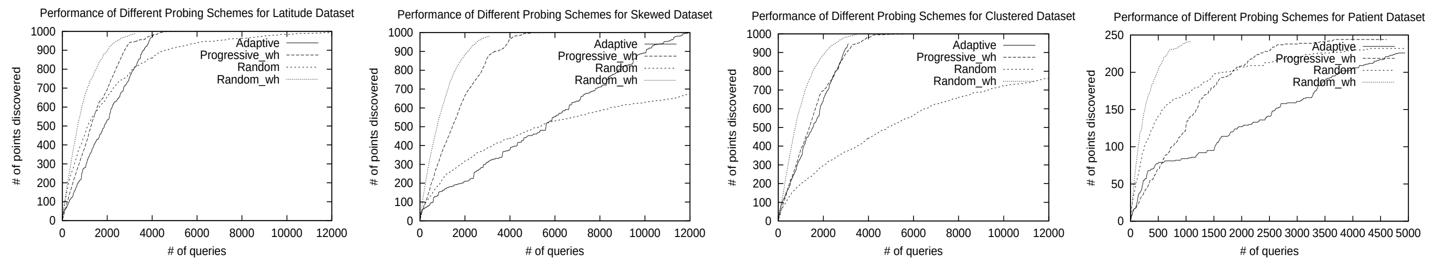


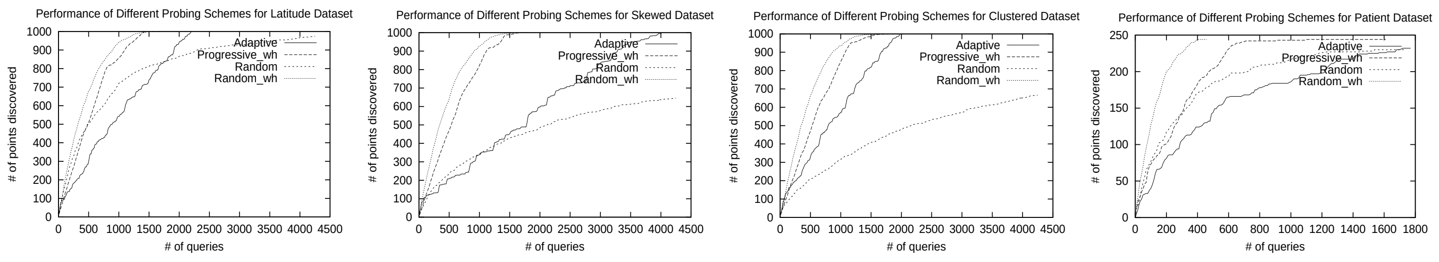
Fig. 10. Plot of 2 Dimensional Data sets

increases, more points are discovered by both schemes. The maximum number of points discovered by both schemes uses same number of queries.

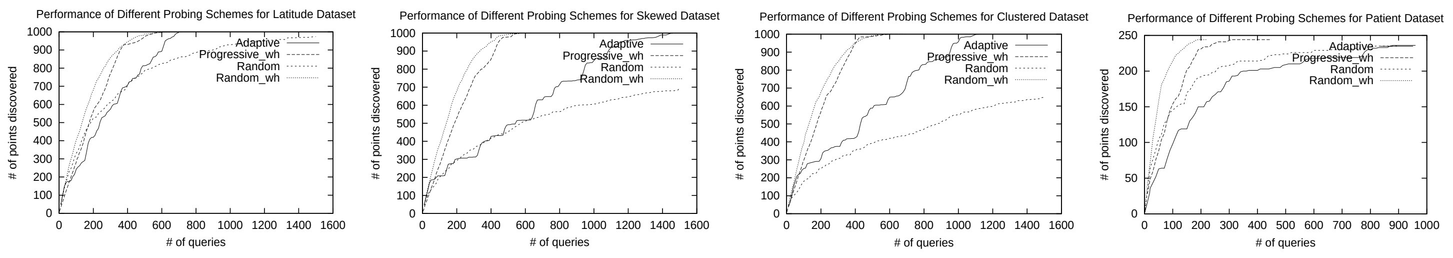
These results show that similarity search based systems are highly vulnerable under the reply model and protection mechanisms must be designed to protect them.



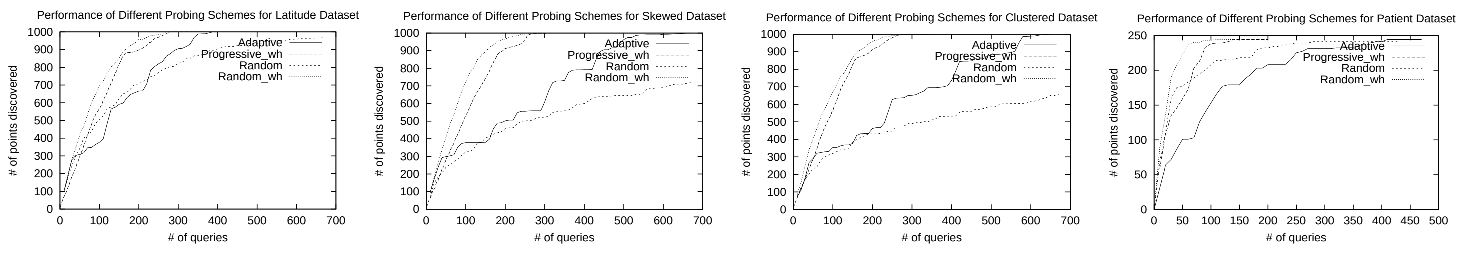
(a) k=1



(b) k=2



(c) k=5



(d) k=10

Fig. 11. Performance of 4 probing algorithms for reply model

H. When to Stop Sending Queries

Without using the results of the earlier queries, for general and progressive models user should send all possible probes to be sure that the whole database is discovered. The adaptive scheme has its own stopping criteria which is the emptiness of the priority queue. For random scheme, we need to know when all the points in the database are discovered. For example in 2 dimensions with $k = 1$, if the circles (center is probe point, NN is on the perimeter) cover the domain then all the points are discovered. We can formally state this for k-NN and high dimensions as follows:

Theorem 2.2: Consider a d-dimensional system where N probes for k-NN are sent and k^{th} nearest neighbor of probe p_i=(a_i, b_i) is r_i=(α_i, β_i) where $d(p_i, r_i)$=s_i. If $\cup_{m=1}^{N}\ Hypersphere(p_m, s_m)$ covers the domain $[l_1, u_1]$ x $[l_2, u_2]$ x...x $[l_d, u_d]$ then all the points in the database are discovered where Hypersphere(p_m, s_m) denotes hypersphere with center p_m and radius s_m.

Proof: Assume that Hyperspheres cover the domain $[l_1, u_1]$ x $[l_2, u_2]$x...x $[l_d, u_d]$ and a point q is not discovered by the probes. Since hyperspheres cover the domain q has to be in at least one hypersphere. Let $Hypersphere(p_j, s_j)$ be a hypersphere in which q lies. Then, q satisfies $d(p_j, q) <$ s$_j$. This contradicts the fact that Probe with center p_j has returned k-NN with k^{th} nearest neighbor being $r_j = (\alpha_j, \beta_j)$ where $d(p_j, r_i) = s_j$. Probe p_j must have returned q. ■

Using the results of earlier queries prevent the user from sending unnecessary queries. Results have shown that for progressive model, which has equal number of probes to that of the general scheme, by using query histories total number of queries sent is about 1% of all possible probes. For random scheme the total number of probes is determined according to adaptive probing which sends more queries than the other two schemes. In random scheme with query histories, the stopping criteria is when all the space is covered by the queries sent. In our implementation, the stopping criteria is when no new data points are discovered by certain number of probes. This number depends on k.

I. Attack Detection

For the reply model database discovery techniques described, we can use a number of tools to attempt to detect such attacks. As the nature of the discovery method becomes more sophisticated, a more detailed attack detection method is necessary. Query information that can be useful in the detection of attacks includes the requesting user IP addresses, successive query patterns and differences, and query success in terms of avoiding unnecessary queries. Possible detection methods as well as the challenges associated with the probe techniques as they increase in sophistication are described below.

The general probe scheme is the easiest to detect. It sends probes in an ordered sequence. When a probe is sent to the database, the reply has to be received to learn the data in the database. Therefore the real IP address of each node should be sent to the database. This IP address helps in attack detection. To detect such an attack, one need only to detect a sequence of ordered probes from an IP address.

A level of complexity is added if the attacker sends the same sequence in some arbitrary order without sending a probe twice. For example if we have 8 probes numbered 0 to 7, we can send them in the order 1,6,3,0,5,2,7,4. This follows from the number theory fact that if number of probes np and skip value s are relatively prime (gcd(np,s)=1) then each number is repeated only once. Same argument applies to higher dimensions as well with condition that the skip value in each dimension should be relatively prime to np. Detecting such attacks is difficult and sophisticated mining algorithms may be needed.

This level of complexity is similar to that experienced with the progressive attacks. Progressive attacks are harder to detect since we need to look at the pattern the probes are sent. Probes at a given level can be reordered making it even more difficult to detect. Still more complexity can be added to the probing by distributing the attacks from different receiving IP addresses. Typically distributed attacks are the most difficult to detect. The ability to detect such attacks will depend on the number of hosts involved in the attack.

Most of the above attacks will require large number of queries being sent to the database. The database server can identify the clients based on their IP address. These attack strategies use c_k to determine the positions of the probes. In the case where c_k is unknown they use d. So, if a user is identified such that the corresponding queries uses a common d for all dimensions, the number of queries that can be sent by that user can be limited in some time interval. If he continues to send such queries, the time interval can be increased according to the total number of queries. As an example, if a user has sent 10 queries with a common d, then he is subsequently allowed to send only one query in 1 minute period. The key to the detection of these attacks is detecting the use of a common d in a sequence of queries. This method could be problematic if a user queries the database in such a way that it appears that a common d is being used (such as always querying with 2 significant figures).

As the experimental results highlighted, the probing algorithms using query histories perform well. An additional complication is that random probing with query histories does not use a common d. Therefore, detecting such attacks is more difficult than the organized ones. However, the strength of these strategies, using query histories, may also be a weakness. The queries sent after a query q should not have a probe point such that the corresponding voronoi region of that point completely lies in the region covered by q and its answer. By storing the top m queries with their results according to the spread of their coverage for a user, we can check if the user avoids sending unnecessary queries or not. According to the number of queries do not include any unnecessary ones, the user can also be limited as described for the previous case.

Detecting this class of attacks requires a more complicated structure. To store the information, we use need a space-efficient lightweight data structure. We use a bloom filter for this purpose. We insert the client IP address into the bloom filter. This address uniquely identifies the computer used to send the query. The insertion sets several bits in the bloom filter. Upon receiving a query, the filter is checked to see if the corresponding bits are set. If they are then the query is rejected. Otherwise, the corresponding bits are set. To reduce the cost of bloom filter operation, we can insert a query into the filter probabilistically. In this case our detection scheme will also be probabilistic. Assume that the query is inserted into the bloom filter with probability p. Let Z count the number of failures (queries not inserted into filter) prior to the first success (query inserted into the filter). Then, Z has a geometric distribution with parameter p. The probability mass function is $p_Z(k) = (1-p)^k p$ and $E[Z] = \frac{1-p}{p}$. So, given p expected number of failures prior to first success is $\frac{1-p}{p}$ and total number of trials is $\frac{1-p}{p} + 1$ We can choose p depending on how many queries we are willing to let from a given user. For example, if we choose $p = 0.1$ then, $\frac{1-0.1}{0.1} + 1 = 10$ queries from a single client will be accepted on the average. In order to have m queries to be accepted, p should be $\frac{1}{m}$. The probability that more than desired number of queries is accepted can be computed as follows

$$\sum_{k=m}^{\infty} p(1-p)^k = p\frac{(1-p)^m}{p} = (1-p)^m.$$

III. Vulnerabilities in Score Model

In the score model, upon receiving a similarity query x from user, the database responds with similarity score $\parallel x - y \parallel$ [22]. As only the distance from the query point to the closest data point in the database is returned, c will be used instead of c_1 in this section. The attack strategy will initially be explained for 2 dimensions using l_2 (Euclidian) distance and then will be generalized to n dimensions using l$_p$ norm as distance metric.

```
$n$ = dimensionality of vectors $x$ and $y$
$q_1 = sim\_search([0, 0, ...0])$
for $i$= 1 to $n$ do
    $q_2 = sim\_search(createVector(n, i, 1))$
    $y_i = (q_2^2 - q_1^2 - 1)/-2$
```

Fig. 12. Algorithm to discover y

We will first focus on a simple version where a database has only a *single* n-dimensional vector y. By using the algorithm given in Figure 12, the vector y can be discovered using $n+1$ queries. So, for 2 dimensions 3 queries are needed. The function $createVector(d, i, v)$ creates a d dimensional vector which has 0 in all dimensions but v in the i^{th} dimension.

Lemma 3.1: Algorithm in Figure 12 discovers y.

Proof: Consider i^{th} iteration of the loop. We have $q_1 = (\sum_{k=1}^{n}(y_i)^2)^{1/2}$ and $q_2 = ((y_i - 1)^2 + \sum_{k=1,k\neq i}^{n}(y_i)^2)^{1/2}$. By simple algebra we get $q_2^2 - q_1^2 = -2y_i + 1$. We can solve this equation to find y_i. ■

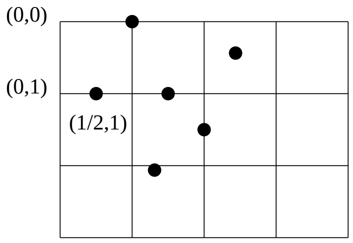


Fig. 13. Example on score model

Now let us consider a database with large number of tuples. In this case, the algorithm in Figure 12 can not be used since queries q_1 and q_2 can return scores based on different vectors in database. The example shown in Figure 13 returns a score of 1 for similarity search $(0,0)$ and returns a score of $1/2$ for similarity search $(0,1)$. These are not comparable since their distances are to different nodes.

To discover every point in the database, closely located queries need to be sent to make sure that all the queries return a score to the same data point. Assume that the whole space is divided into equally spaced squares as in the general scheme of reply model and that the length of an edge of the square, which is also the distance between two consecutive probes, is $c\prime$. In the best case, (n+1)=3 probes having the coordinates x,y,x,y+$c\prime$,x+$c\prime$,y are required to return score to the same data point to guarantee discovering it. Recall that in the score model we have the probes and the associated scores, but do not have any information identifying the data point for the returned score. Therefore, to discover data points when the databases use the score model, we must take advantage of all the information at hand: the returned scores and the distance between probes, $c\prime$. The task becomes finding a probe distance $c\prime$ such that putting a condition on the returned scores for the nearby probe points guarantees that the score returned by each of them is to the same data point. In the following lemma we put a condition, maximum score, on the returned distance for a probe to guarantee that each of its $c\prime$ distanced neighbors returns a score to the same data point.

Lemma 3.2: Let the score for a probe $p = (x,y)$ be δ where $\delta \leq \frac{c}{4}$ and the closest point to probe p be $q = (s,t)$, then the closest point to probes $p_2 = (x + \frac{c}{4}, y)$, $p_3 = (x - \frac{c}{4}, y)$, $p_4 = (x, y + \frac{c}{4})$, and $p_5 = (x, y - \frac{c}{4})$ is $q = (s,t)$.

Proof: Consider the distance $d(p_2, q)$. By triangle inequality we have $d(p_2,q) \leq d(p_2,p) + d(p,q) \leq \delta + \frac{c}{4}$. Therefore, $d(p_2,q) \leq \frac{c}{4} + \frac{c}{4} \leq \frac{c}{2}$. Since c is the smallest distance between pairs, closest point to probe $p_2 = (x + \frac{c}{4}, y)$ is $q = (s,t)$. Proofs for p_3, p_4, p_5 are similar. ■

The required distance, $c/4$, and $c\prime$, $\frac{c}{4}$, are selected such that sum of them is $\leq c/2$ and at least one of the corners of the square a data point lies in returns a score $\leq$ the required distance, $c/4$ to the point. Algorithm shown in Figure 14 utilizes lemma 3.2 to discover all points in a two dimensional space.

Theorem 3.1: Algorithm in Figure 14 discovers the whole database.

Proof: To make the proof complete we need to show for each data point that

(1) at least one probe exists such that distance between
the probe and the data point is $\leq \frac{c}{4}$ and
(2) algorithm guarantees to find the data point.

The proof for each item will be made separately.

(1) Since voronoi regions cover the whole region, each data point lies in a square having edge of $\frac{c}{4}$. Hence, the longest distance between a data point and the nearest corner of the square it lies into can be $c\sqrt{2}/8$ which is less than $c/4$. Thus, at least one of these probes will return a score $\leq \frac{c}{4}$ to the point.

```
$\alpha_1$ =$\lceil \frac{u_1-l_1}{c/4} \rceil + 1$
$\alpha_2$ =$\lceil \frac{u_2-l_2}{c/4} \rceil + 1$
for $i$= $0$ to $\alpha_1 - 1$ do
    for $j$= $0$ to $\alpha_2 - 1$ do
        $probe[1]$ = $l_1$+i$\frac{c}{4}$
        $probe[2]$ = $l_2$+j$\frac{c}{4}$
        $dist_{i,j}$ = dist_search($probe$)

for $i$= $0$ to $\alpha_1 - 2$ do
    for $j$= $0$ to $\alpha_2 - 2$ do
        if $dist_{i,j} \leq c/4$
          $y_1 = \frac{\frac{dist^2_{i+1,j}-dist^2_{i,j}}{c/4} - 2(l_1+i\frac{c}{4}) - c/4}{-2}$
          $y_2 = \frac{\frac{dist^2_{i,j+1}-dist^2_{i,j}}{c/4} - 2(l_2+i\frac{c}{4}) - c/4}{-2}$
          if $y$ not in database
            save $y$
```

Fig. 14. Solution for Score model with General Probe Algorithm

(2) As stated by Lemma 3.2, each of the probes with indices $\{i, j\}, \{i+1, j\}$ and $\{i, j+1\}$ returns the distance to the the same data point if the returned distance for $\{i, j\} \leq \frac{c}{4}$. Let us call this data point y. We have $dist_{i,j} = ((l_1+i\frac{c}{4}-y_1)^2+(l_2+j\frac{c}{4}-y_2)^2))^{1/2}$ and $dist_{i+1,j} = ((l_1+(i+1)\frac{c}{4}-y_1)^2+(l_2+j\frac{c}{4}-y_2)^2))^{1/2}$. By simple algebra we get $dist^2_{i+1,j} - dist^2_{i,j} = \frac{c}{4}(-2y_1 + 2(l_1 + i\frac{c}{4}) + \frac{c}{4})$. We can solve this equation to find y_1. We can find y_2 similarly by considering probe with indices $\{i, j+1\}$ instead of $\{i+1, j\}$. Since we have shown that such a probe exist for each data point, the algorithm will discover the whole database. ■

For 2 dimensional space, the probe points are placed at the center of the squares in the reply model, whereas in score model they are located at the corners. Therefore, instead of the floor operation, the ceiling operation is performed while the total number of divisions for each dimension is decided. The distance between two consecutive probes in reply model, $c_k/\sqrt{2}$, is selected such that at most 2 point(s) will be in each square. Hence, each point is returned as an answer for at least one query. In the score model, the distance $c/4$ is selected to guarantee that for each data point, at least one of the probes belonging to corners of the square that the data point lies in returns a score less than or equal to the required distance, $c/4$. Therefore, there is at least one probe within $c/4$ distance to each point and by using Lemma 3.2 the point can be discovered.

A. *Extension to n Dimensions and l_p norm*

We will extend the solution presented for 2 dimensions to n dimensions in three steps. As a first step, we will show that there is a solution for the simple case where the database has only one point. Then the function to calculate the total number of probes in n dimensions is provided. This function depends on the distance metric (l_p norm) and n. The last step is to generalize Lemma 3.2, on which solution was built, to the case where we have n dimensions and use the l_p norm as the metric.

Definition: 3.1: l_p norm between n dimensional vectors x and y is defined as $l_p(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$.

The solution shown in Figure 15 is an extended version of the solution in Figure 12. There are two differences between these solutions. The first difference is that we send a value of -1 instead of a value of 1 in the appropriate dimension when we are gathering information about a dimension. The other difference

```
$q_1 = sim\_search([0, 0, ...0])$
for $i$ = 1 to $n$ do
    $q_2 = sim\_search(createVector(n, i, -1))$
    Compute $y_i$
```

Fig. 15. Algorithm to discover y

is that while for 2 dimensions using l_2 norm we can find an exact solution for y_i, we can only show that a unique real solution exists for n dimensions using l_p norm.

Theorem 3.2: Algorithm in Figure 15 discovers y for every l_p.

Proof: Consider i^{th} iteration of the loop. Using the definition of l_p norm, we have

$$q_1 = (\sum_{j=1}^{n} |y_j|^p)^{1/p}$$

and

$$q_2 = (|y_i + 1|^p + \sum_{j=1, j\neq i}^{n} |y_j|^p)^{1/p}$$

By using above equation we have

$$q_2^p - q_1^p = |y_i + 1|^p - |y_i|^p$$

Left hand side is known since we know q_1, q_2 and p. As q_1 and q_2 are returned for the same data point, we know that this equation has at least one real solution, y_i. Therefore, we need to prove the uniqueness of the solution to discover y_i. We need to show that this is an increasing function in each of the possible 3 intervals, ≥ 0, $[-1, 0)$, and < -1, to prove the uniqueness of the solution.

◇ $y_i \geq 0$: The expression becomes $(y_i + 1)^p - (y_i)^p$. If we take the derivative with respect to y_i we get

$$p(y_i + 1)^{p-1} - p(y_i)^{p-1}$$

Since $p \geq 1$ above expression is always positive. Therefore, in this interval the function is increasing.

◇ $-1 \leq y_i < 0$: The expression becomes $(y_i + 1)^p - (-y_i)^p$. If we take the derivative with respect to y_i we get

$$p(y_i + 1)^{p-1} + p(-y_i)^{p-1}$$

Above expression is always positive therefore, in this interval the function is increasing.

◇ $y_i < -1$: The expression becomes $(-(y_i + 1))^p - (-y_i)^p$. If we take the derivative with respect to y_i we get

$$-p(-(y_i + 1))^{p-1} + p(-y_i)^{p-1}$$

If we reorganize the expression we get

$$p((-y_i)^{p-1} - (-(y_i + 1))^{p-1})$$

Since the first term in the parenthesis is always greater than the second one, the derivative is always positive in this interval. Hence, in this interval the function is increasing .

Since the function is increasing in all of the intervals, it is an increasing function. As a result, independent from p we have a unique real solution for y_i and it can be computed by Newton's method. ■

As shown for the reply model, the total number of probes for n dimensions is found by multiplying the number of probes needed for each dimension separately. The following theorem considers this fact and the gives the method to find the common distance between closest probes, $c\prime$.

Theorem 3.3: For score model, general scheme in n dimensions utilizing l_p norm as the distance metric requires $\prod_{i=1}^{n} \lceil \frac{u_i - l_i}{c\prime} \rceil + 1$ probes to discover all elements of an n dimensional database where $c\prime$ is equal to the minimum of $c/4$ and $\frac{c}{2(n)^{1/p}}$.

Proof: The general scheme divides the whole space into hypercubes having an edge of c' which is selected to guarantee that the returned score for at least one of the probes belonging to the corners of the region that a data point lies in is $\leq$ the required distance, $c/4$. To make this guarantee, $c/4$ should be the longest distance between a point and its closest probe. Hence, the diagonal of the hypercube should be $c/2$ and an edge of it, c' should be $\frac{c}{2(n)^{1/p}}$. However, we should also consider the requirement that c' should be $\leq$ c/4. So, c' is the minimum of c/4 and $\frac{c}{2(n)^{1/p}}$ for n dimensions using l_p norm. ■

Lemma 3.2 is at the heart of the solution for 2 dimensions using l_2 norm shown in Figure 14. The extended version of this Lemma should be used for the solution for n dimensions using l_p norm. We will not make the proof, instead we will verify that the sum of the required distance, $c/4$ and the c' is $\leq c/2$. The value of c' for n dimensions using l_p norm is given in Theorem 3.3 as the minimum of $\frac{c}{2(n)^{1/p}}$ and $c/4$. So, the sum will be less than or equal to $c/2$. As a result, if the distance returned for a probe is less than or equal to $c/4$, all probes which are in the c' distance to this probe will return a score to the same data point. In total, there are $2n$ such probes.

The proof outline for proving the existence of a unique real solution when the database contains a large number of data points, and the probe that has a score $\leq c/4$ is considered with its c' neighbors is to follow the same path as Theorem 3.2. It is trivial if the path of this proof is followed.

B. Experimental Results

In the score model, due to the nature of the solution, only the organized algorithms can be used. Therefore, we implemented general probing with query histories and progressive probing with query histories. Since we do not utilize any distance $> c/2$ while discovering a data point, we used already sent queries to eliminate probes which have scores more than $> c/2$. Results for the data sets described in Section II-G are given in Figure 16.

Because of the leveling strategy used in the progressive model, queries that are next to each other are not sent until the last level. Therefore, this strategy does not find any points until it starts to send the probes on this last level. On the other hand, the distribution of probes resulting from the progressive scheme eliminates more probes than the general scheme. This phenomenon is shown in the graphs in Figure 16. The progressive scheme does not begin to discover points until the general scheme has already discovered about half the database, but it completely discovers the database before the general scheme does. Depending upon when a potential attack detection method catches the attack, one of these two schemes can be preferred.

Results for both of these models show that similarity search based systems are vulnerable and protection mechanisms must be designed to protect them.

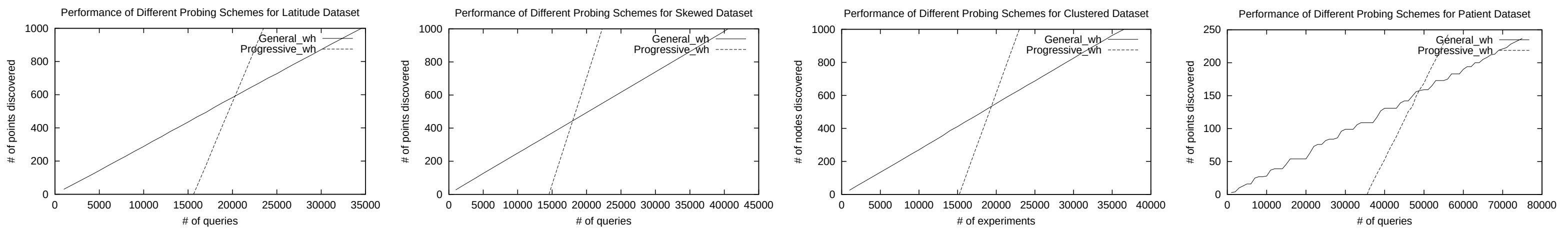


Fig. 16. Performance of General and Progressive probing using history

C. Attack Detection

Potential attack detection scenarios for both schemes were discussed in section II-I. In this section, a score model specific detection model will be introduced.

We know that two similarity queries with 1 dimension differing by $c\prime$ should be sent to reveal a specific dimension of a vector. If we sum all coordinates of query points, then the two queries used to attack a given tuple will differ by $c\prime$. Therefore, we can represent each query point by a single number. We will use a data structure to store query points. When a new query q_{new} is received, we first find the sum of all coordinates and then search the data structure to see if there is a node which differs from sum of coordinates of the query point q_{new} by $c\prime$. If there is such a node then the database server denies this query otherwise it responds to the query.

We can use hashing to deny similar queries by the same user by combining the identifier for user with the sum of attributes of the query vector. For example, for a user with IP address 192.80.25.6 and query [2 5 8.2] in 3 dimensions, we sum the attributes of vector to get 15.2 and we concatenate it with IP address to get 192080025006015.2. Note that we pad individual fields of IP addresses with 0's to get a number with same number of digits. Here we assume that score of similarity search will be less than 1000. We use a bloom filter to store this information since bloom filters are space-efficient and lightweight. We drop the fractional part and insert 192080025006015 into the bloom filter. Another query from the same machine with vector sum of 15.* will be denied.

Distributed attacks are harder to detect in any model, and the score model is no exception. Depending on how many nodes are potentially involved in the attack, we can use slightly different detection strategies. The difference will be the IP address used. There are 3 different types of IP addresses. IP addresses are in the form of A.B.C.D where A,B,C and D are 8 bit binary numbers. Some domains have at most 2^8 computers connected to them. Such domains will have the same A,B,C components in the IP address only D components will differ for machines in these domains. Some domains can accommodate 2^{16} computers and have the same A and B fields in addresses. The third type of domains can accommodate 2^{24} computers and have the same A fields in their addresses. Detection of distributed attacks will be based on how distributed the attack can be. For example, all computers in a lab will have same A,B and C fields in their IP address. To detect attacks launched used multiple computers in a lab, we can combine the A, B and C fields of IP address with sum of attributes of tuples and use this number in hashing. For example, for a user with IP address 192.80.25.6 and query [2 5 8.2] in 3 dimensions we insert 192080025015 into the filter. The only way to detect truly distributed attacks (multiple computers from different domains) is to use only the sum of attributes in hashing. This may result in some legitimate queries being denied.

As the number of queries increases, the data structure used to store the information will grow. The structure can be maintained for a certain amount of time and then a new data structure can be started. With this model, queries by the same user sent in the lifespan of the data structure will be denied. We can partition the database and use different data structures for each partition. This approach will alleviate the scalability problems caused by the data structure. Another approach for scalability is a probabilistic approach. We can insert a query into this data structure based on some probability. Using this approach the number of nodes in the data structure will be lower and queries are denied probabilistically.

IV. Comparison of Reply Model and Score Model

If no attack detection scheme is used both models are vulnerable. However as the results of experiments for score model and reply model with k=1 shows, the number of queries needed to obtain the contents of the database in score model is much larger than the number needed for reply model. In reply model at least N queries and in score model at least $(n+1)N$ queries needed to be sent to learn the contents of the database. However, due to the nature of the question in score model, selected $c\prime$ is $1/2\sqrt{2}$ (when equal to c/4) times that of $c_k\prime$ in reply model. Therefore, the number of queries needed by the common probing algorithms to discover whole data set varies according to the ratio between the edge of the hypercubes rather than (n+1). For example for 2 dimensional data set this ratio is about 1/8.

In score model attacks are limited, very close queries should be sent to the database to learn information and these can be detected without using much information. It is enough to store only the sum of attributes of the tuples. Close queries will have similar sum and these can be detected. In reply model based on common increment factor for all dimensions or number of necessary queries we limit the number of queries that can be sent by a single user in some fixed time interval. This can be implemented efficiently by using hash table and the attributes needed for a user such as timestamp, IP address, common increment factor and top m queries.

V. Attack Detection in Replicated Databases

We considered a single copy of the database. If the database is replicated then attackers can send similar queries to the different copies and avoid being detected. Therefore for replicated databases distributed attack detection schemes should be used. All the copies of the database should jointly decide whether the query sent to one of the copies should be denied or not.

For score model a detection scheme is as follows: Whenever a copy receives a query it sends the value to be hashed (can be IP address concatenated with sum of attributes or only sum of attributes depending on what is hashed) to other copies of the database. Upon receiving a check request, other copies use the bloom filter table to see if they had received a similar query in the past or not. If they have not received a similar query they send an accept message to database which originally received the request. If all other copies send accept messages then the database accepts the query and sends the score to user.

Communicating a single number serves two useful purposes.

- It preserves the privacy of the copy of the database which received the request.
- It is communication efficient since instead of 100s of attributes only a single number is sent.

The databases can combine the data structures into a single structure by using the union property of bloom filters. Let $B(i)$ be a bloom filter for database i that stores the queries sent to database i. Then, the bloom filter for all the queries can be computed as $\bigvee_{i=1}^{k} B(i)$ where $\bigvee$ denotes the bitwise OR operation on the filters.

In score model we can also use a centralized detection scheme by using a centralized bloom filter containing requests from all the copies. In this case, the database waits for permission from the central detection server to respond to the query.

In reply model detection can be performed by limiting the number of queries sent by a single user to any copy of the database. This can be done using a distributed algorithm or a centralized algorithm. However, number of items shared within the nodes is much larger than the ones for score model. So, our proposed solutions for detection can easily be extended to replicated databases.

VI. Conclusion

In this paper we showed the vulnerabilities of similarity search based systems against malicious discovery through automated similarity queries. We addressed two kinds of similarity searches, *reply model* and *score model*. In *reply model* the most similar k data points to a query are returned whereas the similarity score to the closest data point is returned in *score model*. We stated that both systems reveal data if automated queries are sent and measures must be taken to detect and deny such queries. We identified possible attacks in both models and develop strategies against them. Reply model is the natural choice for the majority of the applications, but it is clearly more vulnerable to attacks based on automated requests. Our experimental results for reply model showed that as k increases the percentage of the points learnt by sending m automated queries for a database of m points increases from 15$\%$ to 100$\%$. For example, for a data set with 1000 data points, only 200 queries were enough to identify the whole data set. When the score model was used, only 2$\%$ of the data was revealed in the same experiment. Score model is less vulnerable since it requires much larger number of queries and attacks are easier to detect. We proposed a detection scheme for both models that utilizes common divisor for all dimensions or necessary queries to determine the time interval in which only a single query is accepted from a user. In addition, a detection technique is developed based on hashing a value composed of IP address of users and a single number (sum of attributes of query) based on query sent for score model. We also extended proposed schemes to detect attacks to replicated databases.

Web-based search engines and many biomedical and clinical databases utilize similarity search as their major type of query. We proposed a number of techniques that protect the proprietary and private information in these databases while keeping them functional. More sophisticated data mining techniques can be utilized to detect malicious discoveries. Quantitative measures are needed to evaluate the detection schemes in terms of their power to minimize number of false denials (denied legitimate queries) and missed attacks (attack queries identified as legitimate).

VII. Acknowledgements

A special thanks to Michael Gibas for his valuable comments and suggestions.

References

[1] N. R. Adam and J. C. Wortman. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4), 1989.

[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[3] R. Agrawal, P. Haas, and J. Kiernan. Watermarking relational data: Framework, algorithms and analysis. In *The VLDB Journal*, volume 12, 2003.

[4] R. Agrawal and J. Kiernan. Watermarking relational databases. In 28^{th} *International Conference on Very Large Data Bases*, August 2002.

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *ACM SIGMOD International Conference on Management of Data*, 2004.

[6] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD International Conference on Management of Data*, May 2000.

[7] G. Ankerst, M.and Kastenmller, H. Kriegel, and T. Seidl. Nearest neighbor classification in 3d protein databases. In *Proc. 7th Int. Conf. on Intelligent Systems for Molecular Biology (ISMB'99)*.

[8] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[9] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *7th International Workshop on Algorithms and Data Structures*, August 2001.

[10] M. Bawa, R. Bayardo, and R. Agrawal. Privacy-preserving indexing of documents on the network. In 29^{th} *International Conference on Very Large Data Bases*, 2003.

[11] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, pages 78–86, Tuscon, Arizona, June 1997.

[12] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful. In *Int. Conf. on Database Theory*, pages 217–225, Jerusalem, Israel, Jan. 1999.

[13] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, pages 636–646, 2002.

[14] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE Transactions on Knowledge and Data Engineering*, 12(6), 2000.

[15] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Lecture Notes in Computer Science*, 1592:402–??, 1999.

[16] C.Chor, O.Goldreich, E.Kushilevitz, and M.Sudan. Private information retrieval. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 1995.

[17] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin. Enforcing fairness in routing as a space and time efficient method to keep track of which hosts are sending too much traffic. INFOCOMM 2001: Twientieth Annual Joint Conference of the IEEE Computer and Communications Societies, 2001.

[18] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. Compactly encoding a function with static support in order to support approximate evaluations queries. Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, January 2004.

[19] F. Chin. Security problems on inference control for sum, max and min queries. *Journal of the ACM*, 33(3):451–464, July 1986.

[20] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proc. Int. Conf. Data Engineering*, pages 244–255, San Diego, California, Mar. 2000.

[21] D. E. Denning. Commutative filters for reducing inference threats in multilevel database systems. In *IEEE Symposium on Security and Privacy*, pages 134–146, 1985.

[22] W. Du and M. Atallah. Protocols for secure remote database access with approximate matching. In *7th ACM Conference of Computer and Communications Security (ACMCSS 2000), The First Workshop on Security and Privacy in E-commerce*.

[23] W. Du and M. J. Atallah. Privacy-preserving cooperative statistical analysis. In *Annual Computer Security Applications Conference (ACSAC 2001)*, December 2001.

[24] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy-preserving data mining. In 22^{nd} *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2003.

[25] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy-preserving mining of association rules. In 8^{th} *ACM SIGKDD International Conference on Knowledge Discovery in Databases and Data Mining*, July 2002.

[26] L. Fan, P. Cao, J. Almeida, and A. Broder. Web cache sharing. Collaborating Web caches use bloom filter to represent local set of cached files to reduce the netwrok traffic. IEEE/ACM Transactions on Networking, 2000.

[27] C. Farkas and S. Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 4(2).

[28] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained nearest neighbor queries. In *Proc. of the 7th International Symposium on Spatial and Temporal Databases (SSTD)*, Los Angeles, CA, July 2001.

[29] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the 9th ACM Int. Conf. on Information and Knowledge Management*, pages 202–209, McLean, Virginia, Nov. 2000.

[30] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc of 17th IEEE Int. Conf. on Data Engineering (ICDE)*, pages 503–511, Heidelberg, Germany, Apr. 2001.

[31] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[32] T. H. Hinke. Inference aggregation detection in database management systems. In *IEEE Symposium on Security and Privacy*, pages 96–106, 1988.

[33] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM Symposium on Theory of Computing*, pages 604–613, Dallas, Texas, May 1998.

[34] T. F. Keefe, B. Thuraisimgham, and W. T. Tsai. Secure query-processing strategies. *IEEE Computer*, 22(3):63–70p;, 1989.

[35] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast and efficient retrieval of medical tumor shapes. *IEEE Transactions on Data Engineering (TKDE'98)*.

[36] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 215–226, Mumbai, India, 1996.

[37] A. Kumar, J. J. Xu, and J. W. Li Li. Measuring approximately yet reasonably accurate per-flow accounting without maintaining per-flow state. Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement, 2003 October.

[38] D. G. Marks. Inference in mls database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):46–55, 1996.

[39] G. Miklau and D. Suciu. Cryptographically enforced conditional access for xml. In *WebDB*, 2002.

[40] P. Mishra and M. H. Eich. Join processing in relational databases. In *ACM Computing Surveys (CSUR)*, March 1992.

[41] J. K. Mullin. Estimating the size of joins in distributed databases where communication cost must be maintained low. IEEE Transactions on Software Engineering, 1990.

[42] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, California, May 1995.

[43] R. Sion. Proving ownership over categorical data. In *IEEE International Conference on Data Engineering*, 2004.

[44] R. Sion, M. Atallah, and S. Prabhakar. Rights protection for relational data through watermarking. In *IEEE International Conference on Data Engineering*, 2004.

[45] A. C. Snoeren. Hash-based IP traceback. In *ACM SIGCOMM Computer Communication Review , Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[46] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet IP traceback. In *IEEE/ACM Transactions on Networking (TON)*, volume 10, December 2002.

[47] T.-A. Su and G. Ozsoyoglu. Controlling fd and mvd inferences in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(4), 1991.

[48] Y. Ting, M. Winslett, and K. Seamons. Automated trust negotiation over the internet. In *6th World Multiconference on Systemics, Cybernetics and Informatics*, July 2002.

[49] A. Tosun and H. Ferhatosmanoglu. Vulnerabilities in similarity search based systems. In *Proceedings of the eleventh international conference on Information and knowledge management*, November 2002.

[50] J. Wang. Caching proxy servers on the world wide web to improve performance and reduce traffic, October 1999.

[51] R. Weber and K. Bohm. Trading quality for time with nearest-neighbor search. In *Proc. Int. Conf. on Extending Database Technology*, pages 21–35, Konstanz, Germany, Mar. 2000.

[52] A. Whitaker and D. Wetherall. Detecting loops in small networks. 5th IEEE Conference on Open Architectures and Network Programming (OPENARCH), June 2002.

[53] A. C. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.