Authenticated Multi-Step Nearest Neighbor Search

Stavros Papadopoulos, Lixing Wang, Yin Yang, Dimitris Papadias, Panagiotis Karras

Abstract — Multi-step processing is commonly used for nearest neighbor (NN) and similarity search in applications involving high-dimensional data and/or costly distance computations. Today, many such applications require a proof of result correctness. In this setting, clients issue NN queries to a server that maintains a database signed by a trusted authority. The server returns the NN set along with supplementary information that permits result verification using the dataset signature. An adaptation of the multi-step NN algorithm incurs prohibitive network overhead due to the transmission of *false hits*, i.e., records that are not in the NN set, but are nevertheless necessary for its verification. In order to alleviate this problem, we present a novel technique that reduces the size of each false hit. Moreover, we generalize our solution for a distributed setting, where the database is horizontally partitioned over several servers. Finally, we demonstrate the effectiveness of the proposed solutions with real datasets of various dimensionalities.

Index Terms — Query Authentication, Multi-step Nearest Neighbors, Similarity Search.

1 INTRODUCTION

Let DB be a D-dimensional dataset. Each record $P \in DB$ can be thought of as a point in the space defined by the D attribute domains, and in the sequel we use the term record and point interchangeably. Given a point Q, a nearest neighbor (NN) query retrieves the record $\{P \in DB: DST(Q, P) \leq DST(Q, P') \ \forall \ P' \in DB\}$, where $DST(Q, P)$ denotes the distance between Q and P. Likewise, a kNN query returns the k closest points to Q. NN and kNN queries are common in *similarity retrieval*. Specifically, since similarity between records is inversely proportional to their distance, a kNN query returns the k most similar records to Q. The *multi-step* framework [11], [19] has been proposed for NN and similarity retrieval in domains that entail high-dimensional data (e.g., in Time Series, Medical, Image, Biological and Document Databases), expensive distance functions (e.g., Road Network Distance, Dynamic Time Warping), or a combination of both factors.

In this paper, we focus on authenticated multi-step NN search for applications that require a *proof of result correctness*. For instance, [3] argues that the most cost-effective way for medical facilities to maintain radiology images is to *outsource* all image management tasks to specialized commercial providers. Clients issue similarity queries to a provider. The latter returns the result set and additional verification information, based on which the client establishes that the result is indeed correct; i.e., it contains exactly the records of DB that satisfy the query conditions, and that these records indeed originate from their legitimate data source (i.e., the corresponding medical facility). A similar situation occurs for *data replication*, i.e., when a data owner stores DB at several servers. Clients issue their queries to the closest (in terms of network latency) server, but they wish to be assured that the result is the same as if the queries were sent to the original source of DB. In other cases, correctness is guaranteed by a trusted third party. For instance, *notarization services* [20] have been proposed to safeguard against tampering in document databases (the motivating example being Enron). Authenticated query processing ensures the client that the received result complies with the validated DB.

Initially, we study the problem assuming that the entire DB resides at a single server. Our first contribution is AMN, an adaptation of a multi-step algorithm that is optimal in terms of DST computations. AMN requires transmissions of *false hits*, i.e., records that are not in the result, but are nevertheless necessary for its verification. In addition to the network overhead, false hits impose a significant burden to the client, which has to verify them. The second contribution, C-AMN, alleviates this problem through an elaborate scheme that reduces the size false hits. Finally, we consider a distributed setting, where the database is horizontally partitioned over several servers. Our third contribution, ID-AMN, incrementally retrieves data, gradually eliminating servers that cannot contribute results.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 presents the indexing scheme and the query algorithms of AMN. Section 4 focuses on C-AMN and minimization of the false hits. Section 5 deals with distributed servers and ID-AMN. Section 6 contains an extensive experimental evaluation with real datasets, and Section 7 concludes the paper.

2 BACKGROUND

Section 2.1 describes multi-step query processing. Section 2.2 overviews similarity search for high-dimensional data. Section 2.3 surveys background on authenticated queries. Section 2.4 focuses on the MR-Tree, which is used by the proposed techniques.

- *S. Papadopoulos, L. Wang, Y. Yang and D. Papadias are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. Email: {stavros, lxwang, yini, dimitris}@cse.ust.hk*
- *P. Karras is with the School of Computing, National University of Singapore. Email: karras@comp.nus.edu.sg*

2.1 Multi-Step NN Framework

The *multi-step NN* framework is motivated by applications that entail expensive distance computations. Specifically, let $DST(Q, P)$ be the actual distance between a query Q and a data point $P \in DB$. The applicability of the multi-step framework rests on the existence of a *filter* distance metric *dst*, which is cheap to evaluate and satisfies the *lower bounding property*, i.e., for every possible Q and P: $dst(Q, P) \leq DST(Q, P)$. Multi-step NN search was introduced in [11]. Subsequently, Seidl and Kriegel [19] proposed the algorithm of Figure 1, which is optimal in terms of *DST* computations. In order to provide a concrete context, our explanation focuses on road networks [18], where *DST* and *dst* refer to the network and Euclidean distance, respectively. Compared to Euclidean distance (*dst*), network distance (*DST*) computations are significantly more expensive because they entail shortest path algorithms in large graphs. Moreover, the Euclidean *k*NNs can be efficiently retrieved using conventional NN search on a spatial index.

Assuming that *DB* is indexed by an R*-tree [1], the multi-step *k*NN algorithm first retrieves the *k* Euclidean NNs of *Q* using an incremental algorithm (e.g., [7]). These points are inserted into a result set *RS*, and their network (*DST*) distances are computed. Let DST_{max} be the network distance[1] between Q and its current k^{th} NN P_k. The next Euclidean NN P is then retrieved. As long as $dst(Q, P) < DST_{max}$, the algorithm computes $DST(Q, P)$ and compares it against DST_{max}. If $DST(Q, P) < DST_{max}$, P is inserted into *RS*, the previous P_k is expunged, and DST_{max} is updated. The loop of Lines 5-9 terminates when $dst(Q, P) \geq DST_{max}$; because of the lower bounding property of the Euclidean distance, any point lying further in the Euclidean space cannot be closer than DST_{max} in the network.

Algorithm *MultistepNN*(*Q*, *k*)

1. Retrieve the k NNs $\{P_1, ..., P_k\}$ of Q according to *dst*
2. $RS = \{P_1, ..., P_k\}$, sorted according to *DST*
3. $DST_{max} = DST(Q, P_k)$ // the current k^{th} NN *DST*
4. P = next NN of Q according to *dst*
5. While $dst(Q, P) < DST_{max}$
6.   If $DST(Q, P) < DST_{max}$
7.   Insert P into *RS* and remove previous k^{th} NN
8.   Update DST_{max} over *RS*
9. P = next NN of Q according to *dst*

Fig. 1 Optimal multi-step *k*NN processing

Independently of the application domain, the algorithm of Figure 1 performs the minimum number of *DST* computations. Specifically, in addition to *RS*, the *DST* distances are computed *only* for *false hits*, i.e., the set of points $FH = \{P \in DB\text{-}RS\text{: } dst(Q, P) \leq DST(Q, P_k)\}$, where P_k is the final k^{th} nearest neighbor. The rest of the records are not accessed at all (if they reside in pruned nodes of the R*-tree), or they are eliminated using their *dst* to *Q*.

[1] To avoid tedious details, in our discussion we assume that all data distances to the query point are different.

2.2 High-Dimensional Similarity Search using Multi-Step NN

Several applications including Image, Medical, Time Series and Document Databases involve high-dimensional data. Similarity retrieval in these applications based on low-dimensional indexes, such as the R*-Tree [1], is very expensive due to the *dimensionality curse* [2]. Specifically, even for moderate dimensionality (i.e., $D = 20$) a sequential scan that computes $DST(Q, P)$ for every $P \in DB$ is usually cheaper than conventional NN algorithms using the index. Consequently, numerous specialized structures have been proposed for exact [8] and approximate [22] *k*NN search in high dimensions.

The GEMINI framework [6], [11] follows a different approach, combining *multi-step search* with a *dimensionality reduction* technique that exhibits the lower bounding property. Specifically, each record $P \in DB$ is mapped to a low-dimensional representation p in d dimensions ($d \ll D$). The resulting d-dimensional dataset *db* is indexed by an R*-tree, or any low-dimensional index. The query Q is also transformed to a d-dimensional point q and processed using a multi-step method. For instance, in the algorithm of Figure 1, *DST* (resp. *dst*) computations involve high (low) dimensional points. The index prunes most nodes and records using the cheap, filter (*dst*) distances[2], whereas the expensive *DST* computations are necessary only for the points in result *RS* and false hit set *FH*.

GEMINI is the most common approach for performing similarity search over high-dimensional data, and especially time series. Numerous dimensionality reduction methods have been used extensively including *Discrete Fourier Transform* (DFT), *Singular Value Decomposition* (SVD), *Discrete Wavelet Transform* (DWT), *Piecewise Linear Approximation* (PLA), *Piecewise Aggregate Approximation* (PAA), *Adaptive Piecewise Constant Approximation* (APCA), and *Chebyshev Polynomials* (CP). Their effectiveness is measured by the number of records that they can prune using only the low dimensional representations (i.e., it is inversely proportional to the cardinality of *FH*). Ding et al. [5] experimentally compare various techniques, concluding that their effectiveness depends on the data characteristics.

2.3 Authenticated Query Processing

In authenticated query processing, a server maintains a dataset *DB* signed by a trusted authority (e.g., the data owner, a notarization service). The signature *sig* is usually based on a *public-key cryptosystem* (e.g., RSA [16]). The server receives and processes queries from clients. Each query returns a result set $RS \subseteq DB$ that satisfies certain predicates. Moreover, the client must be able to establish that *RS* is *correct*, i.e., that it contains all records of *DB* that satisfy the query conditions, and that these records have not been modified by the server or another entity. Since *sig* captures the entire *DB* (including records not in the query

[2] Note that in GEMINI *DST* and *dst* may be based on the same definition (e.g., they may both be Euclidean distances). In this case, *dst* is cheaper because of the lower dimensionality.

result), in addition to RS, the server returns a *verification object* (VO). Given the VO, the client can verify RS based on *sig* and the signer's public key.

VO generation at the server is usually performed using an *authenticated data structure* (ADS). The most influential ADS is the *Merkle Hash Tree* (MH-Tree) [15], a main-memory binary tree, originally proposed for single record authentication. Each leaf in the MH-Tree stores the digest of a record, calculated using a one-way, collision-resistant hash function $h(\cdot)$, such as SHA-1 [16]. An inner node stores a digest computed on the concatenation of the digests in its children. The trusted authority signs the root digest. Figure 2 illustrates an MH-Tree over 16 records. Assume that a client requests record P_6. When traversing the tree to locate P_6, the server produces a VO that contains the digests (shown in grey) of the siblings of the visited nodes: VO = [[h_{25} [[h_5 P_6] h_{20}]] h_{30}]. Tokens '[' and ']' signify the scope of a node. VO and *sig* are transmitted to the client, which subsequently simulates a reverse tree traversal. Specifically, from h_5 and P_6 it computes h_{19}, then h_{26} (using h_{19} and h_{20}), h_{29} (using h_{25} and h_{26}), and finally h_{31} (using h_{29} and h_{30}). Due to the collision resistance of the hash function, if P_6 is modified, then the digest h_{31} re-constructed by the client will not match *sig*; hence, the verification will fail.

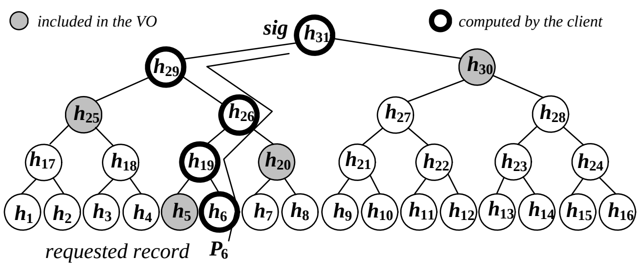


Fig. 2 MH-Tree example

The MH-Tree constitutes the basis of a large number of subsequent ADSs that support: (i) range queries on a single attribute [13], (ii) multi-dimensional ranges [23], (iii) continuous queries on data streams [24], (iv) search in generalized directed acyclic graphs [14], (v) search over peer-to-peer networks [21], (vi) XML queries [12], and (vii) similarity-based document retrieval [17]. In the following, we focus on the MR-Tree [23], the state-of-the-art multi-dimensional ADS, which is utilized by our methods.

2.4 THE MR-TREE

The MR-Tree [23] combines the concepts of the MH-Tree and the R*-Tree [1]. A leaf node contains entries e_{lf} of the form (pg_P, P), where P is an indexed point, and pg_P is a pointer to the page accommodating the record of P. An internal node N stores entries e_{in} of the form (pg_{N_c}, MBR_{N_c}, h_{N_c}), where pg_{N_c} points to e_{in}'s child node N_c. If N is at level 1 (the leafs being at level 0), MBR_{N_c} is the minimum bounding rectangle (MBR) of the points in N_c, and h_{N_c} is a digest computed on the concatenation of the binary representation of the points in N_c. If N lies at higher levels, MBR_{N_c} is the MBR of all the MBR values in N_c, and h_{N_c} is the digest of the concatenation of all pairs (MBR_i, h_i) in N_c. Figure 3 illustrates a 2-dimensional MR-Tree assuming a maximum node capacity of 3 entries per node. A signature *sig* is produced on the root digest $h_{root} = h(MBR_{N2} \mid h_{N2} \mid MBR_{N3} \mid h_{N3})$.

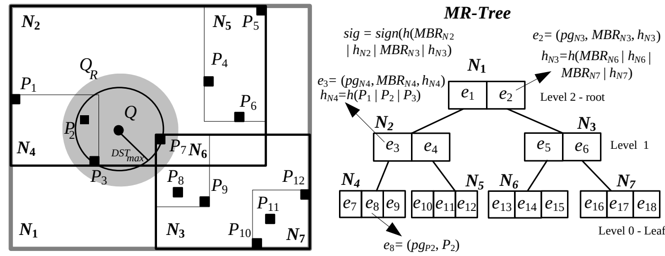


Fig. 3 MR-Tree example

Upon receiving a range query Q_R, the server performs a depth-first traversal of the MR-Tree, using the algorithm of Figure 4, to retrieve the set RS of points in Q_R. Furthermore, it generates a VO_R that contains: (i) all the points outside Q_R that reside in a leaf MBR overlapping Q_R, and (ii) a pair (MBR_N, h_N), for every node N pruned during query processing. In the example of Figure 2, given the shaded range Q_R, we have RS = {P_2, P_3, P_7}, and VO_R = [[[P_1, *result*, *result*] (MBR_{N5}, h_{N5})] [[*result*, P_8, P_9] (MBR_{N7}, h_{N7})]]. The token *result* signifies an object in RS according to the order of appearance in RS. For instance, [P_1, *result*, *result*] corresponds to node N_4; the first occurrence of *result* refers to P_2, and the second one to P_3. In order to distinguish the type of each element in the VO, *MR_Range* includes a *header* prior to each token, digest, and point in the VO. This header consumes 3 bits, which suffice to represent 8 different element types. For simplicity, we omit the headers in our presentation since the element type is implied by its name.

Algorithm *MR_Range* (Q_R, N)

1. Append [to VO
2. For each entry e in N // entries must be enumerated in original order
3. If *N is leaf*
4. If e falls in Q_R
5. Insert e into RS and append a *result* token to VO
6. Else // e falls out of Q_R
7. Append $e.P$ to VO
8. Else // *N is internal node*
9. If *$e.MBR_{Nc}$ overlaps Q*, *MR_Range*(Q, $e.pg_{Nc}$)
10. Else append $e.MBR_{Nc}$, $e.h_{Nc}$ to VO // a pruned child node
11. Append] to VO

Fig. 4 Range query processing with the MR-tree

The server sends RS, VO and *sig* to the client, which first verifies that all points of RS indeed fall in Q_R. Then, it scans VO_R and re-constructs h_{root} bottom-up (using a process similar to the MH-Tree verification). In our example, the client initially substitutes the first two result tokens with P_2 and P_3 from RS, and uses P_1, P_2, and P_3 to compute digest $h_{N4} = h(P_1 \mid P_2 \mid P_3)$ and MBR_{N4}. Then, it utilizes MBR_{N4} and MBR_{N5} to evaluate $h_{N2} = h(MBR_{N4} \mid h_{N4} \mid MBR_{N5} \mid h_{N5})$ and MBR_{N2}. Likewise, it subsequently calculates h_{N3} and MBR_{N3} from the partial VO [[*result*, P_8, P_9] (MBR_{N7}, h_{N7})]. Eventually, it computes $h_{root} = h(MBR_{N2} \mid h_{N2} \mid MBR_{N3} \mid h_{N3})$. If h_{root} matches *sig*, the client is assured that the points in RS are sound. Furthermore, if all MBR values and points in VO lie outside Q_R, then RS is complete.

The MR-Tree can also process NN queries [23]. Assume, for instance, a query asking for the three NNs of point Q in Figure 3. The server: (i) retrieves the *k*NNs of Q (i.e., P_2, P_3 and P_7) using any conventional algorithm; (ii) computes the distance DST_{max} of the k^{th} NN (i.e., $DST_{max} = DST(Q, P_7)$), and (iii) it finally executes *MR_Range* treating (Q, DST_{max}) as the range. For this example, the VO is identical to that of Q_R in Figure 3 since both the resulting points and accessed nodes are the same. The verification process of the client is also identical to the one performed for range queries. However, as an adaptation of the R*-tree, the MR-Tree also suffers from the dimensionality curse [2]. Therefore, the application of the afore-mentioned method on high dimensional data has very limited pruning power. Specifically, for numerous dimensions, nearly all leaf nodes must be visited (leading to high server cost); consequently, the majority of points are inserted in the VO (leading to high communication overhead); finally, the client has to verify almost the entire dataset.

3 AUTHENTICATED MULTI-STEP NN

Our work adopts the GEMINI framework because (i) it has been proven effective in non-authenticated similarity retrieval, especially for numerous (i.e., $D > 100$) dimensions, where even high-dimensional indexes fail[3]; (ii) it can be extended to authenticated query processing based on a low dimensional ADS, i.e., the MR-Tree, whereas, currently there are no authenticated high-dimensional structures; (iii) it is general, i.e., it can also be applied when the expensive distance computations are due to the nature of the distance definition (e.g., network distance), rather than the data dimensionality (in which case $D = d$).

We assume a client-server architecture, where the server maintains data signed by a trusted authority. There are two versions of the signed dataset: a D-dimensional DB and a d-dimensional db ($d \ll D$), produced from DB using any dimensionality reduction technique that satisfies the lower bounding property. For instance, DB may be a set of high-dimensional time series and db their low dimensional representations obtained by DFT. There is a single signature *sig*, generated by a public key cryptosystem (e.g., RSA), that captures both DB and db. DST (dst) refers to the distance metric used in the D(d)-dimensional space. For ease of illustration, we use Euclidean distance for both the DST and dst metrics. Nevertheless, the proposed techniques are independent of these metrics, as well as of the underlying dimensionality reduction technique.

[3] High-dimensional indexes for exact NN retrieval, such as the state-of-the-art *i*-Distance [8], are designed for up to about 50 dimensions, whereas we perform experiments with D up to 1024.

TABLE I
Summary of symbols

Symbol	Description
DB (db)	Dataset in the original (reduced) space
D (d)	Dimensionality (reduced dimensionality)
P (p)	Original data point (reduced representation)
Q (q)	Query (reduced representation)
RS (FH)	Set of the actual k NNs (false hits) of Q
VO_R (VO_P)	VO that authenticates range R (point P)
N	MR-Tree node
h_P / h_p / h_N	Digest of P / p / N
DST (dst)	Distance metric in the original (reduced) space

The proposed *Authenticated Multi-step* NN (AMN) extends the multi-step nearest neighbor algorithm of [19] to our setting. As opposed to optimizing the processing cost at the server, the major objective of AMN (and any query authentication technique, in general) is to minimize (i) the network overhead due to the transmission of the verification object (VO), and (ii) the verification cost at the client (which is assumed to have limited resources compared to the server). Section 3.1 describes the indexing scheme of AMN, while Section 3.2 presents the query processing and verification algorithms. Table 1 summarizes the most important symbols used throughout the paper.

3.1 INDEXING SCHEME

The server indexes db using an MR-Tree. Since authentication information should capture both low- and high- dimensional representations, AMN necessitates the following modifications on the structure of the MR-Tree. Each leaf (level 0) entry e_{lf} has the form (pg_P, p, h_P), where $p \in db$ is the reduced representation of $P \in DB$, and h_P is the digest of the binary representation of P. Pointer pg_P points to the disk page(s) storing P. An intermediate MR-Tree node entry e_{in} has the form (pg_{Nc}, h_{Nc}, MBR_{Nc}), where pg_{Nc} is a pointer to a child node (let N_c), and MBR_{Nc} is the MBR of the points in N_c. The value h_{Nc} depends on the level. For level 1, $h_{Nc} = h(h_{p1} \mid h_{P1} \mid h_{p2} \mid h_{P2} \mid \ldots \mid h_{pf} \mid h_{Pf})$, where h_{pi} (h_{Pi}) denotes the digest of p (resp. P) in the i^{th} entry in N_c. At higher levels, h_{Nc} is computed as in the traditional MR-Tree. Observe that the digests of *both* reduced and original points in the tree are incorporated into the root digest h_{root}. The trusted authority generates a signature *sig* by signing h_{root}. The server maintains DB, the MR-tree and *sig*. Figure 5 outlines the indexing scheme of AMN, assuming the data points and node structure of Figure 3. Note that the proposed techniques are independent of the underlying index. For instance, an authenticated high-dimensional index (if such an ADS existed), would permit higher values of d (compared to the MR-Tree).

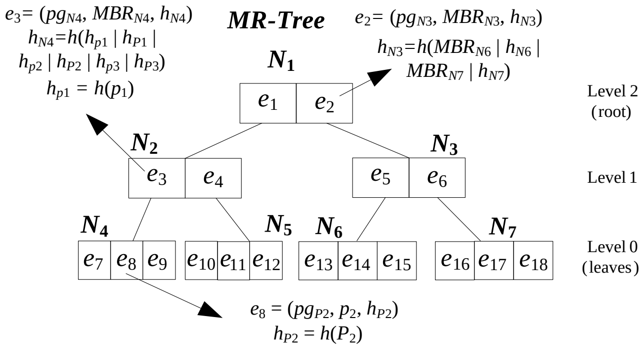


Fig. 5 Indexing scheme in AMN

3.2 Query Processing and Verification

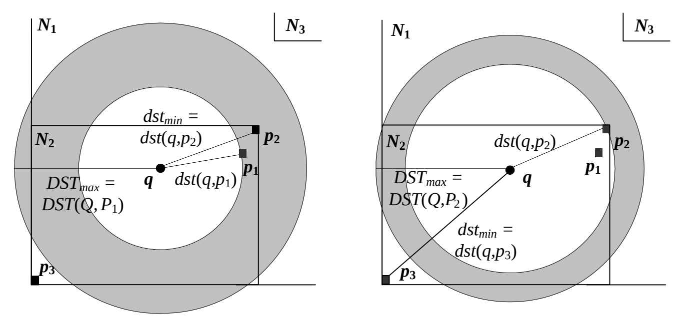
Let Q be a D-dimensional query point received at the server. The goal of AMN is to return to the corresponding client the kNNs of Q, in a verifiable manner. The server starts processing Q by reducing it to a d-dimensional point q, using the same dimensionality reduction technique as for DB. Figure 6 provides the pseudo code for AMN at the server for arbitrary values of k. Initially (Lines 1-4), the algorithm (i) extracts the k closest points $\{p_1, \ldots, p_k\}$ to q using an incremental NN algorithm (e.g., [7]) on the MR-Tree; (ii) it retrieves their high-dimensional representations $P_1, \ldots, P_k$ from the disk, following the pointers from the leaf entries in the MR-Tree, (iii) it initializes a false hit set $FH = \varnothing$ and a result set $RS = \{P_1, \ldots, P_k\}$, and computes DST_{max}, i.e., the DST of the current k^{th} NN P_k; (iv) it obtains the next NN (i.e., the $(k+1)^{th}$ NN) point p of q from the tree.

Algorithm ***AMN_server*(*Q*, *k*)**
// Q is the query and q its reduced representation
1. Retrieve the k NNs $\{p_1, \ldots, p_k\}$ of q in d-dimensional space
2. Retrieve the respective $\{P_1, \ldots, P_k\}$ in D-dimensional space
3. Set $FH = \varnothing$, $RS = \{P_1, \ldots, P_k\}$, and compute DST_{max} over RS
4. p = next NN of q in d-dimensional space
5. While $(dst_{min} = dst(q, p)) < DST_{max}$
6.   Retrieve P // D-dimensional representation of p
7.   If $DST(Q, P) > DST_{max}$, Insert P into FH
8.   Else
9.   Insert P into RS
10.   Remove P_k from RS and insert it into FH
11.   Update DST_{max} over RS
12. p = next NN of q in d-dimensional space
13. $VO_R = MR\_Range((q, DST_{max}), root)$
14. Send RS, FH, VO_R and sig to the client

Fig. 6 Authenticated kNN processing at the server

The procedure then enters the loop of Lines 5-12, where it computes distance $dst_{min} = dst(q, p)$. Observe that dst_{min} is the minimum distance between q and any point p, whose high-dimensional representation has not yet been retrieved. If $dst_{min} > DST_{max}$, the algorithm terminates. Otherwise, it retrieves the true representation P of p from the disk. If $DST(Q, P) > DST_{max}$, then P is a false hit and appended to FH; else, P is a candidate result and is inserted into RS. This causes the deletion of the current k^{th} NN P_k from RS, and its insertion into FH. The algorithm updates DST_{max} over the new RS, and proceeds to retrieve the next NN of q in the tree. After the NN loop terminates, Line 13 performs the authenticated range query $q_R = (q, DST_{max})$ using the most updated value of DST_{max}. This produces a VO_R that contains (i) a pair (h_N, MBR_N) for every pruned node N, (ii) a pair (h_P, p) for every point p in a leaf node that intersects q_R, but whose P representation is not in $RS \cup FH$, (iii) a *result* (*false_hit*) token for every index point whose true point is in RS (FH).

We illustrate AMN using Figure 7 and assuming that k=1 and d=2. Lines 1-3 (in Figure 6) set $FH = \varnothing$, $RS = \{P_1\}$ and $DST_{max} = DST(Q, P_1)$. Let SR (search region) be the area within distance $(dst(q, p_1), DST_{max})$ from q, i.e., the shaded area in Figure 7a. Only points in SR are candidate results. The server proceeds by retrieving the next NN p_2 of q in db, and its high-dimensional representation P_2. Assuming that $DST(Q, P_2) < DST(Q, P_1)$ (= DST_{max}), P_2 becomes the new NN of Q and it is inserted into RS. Moreover, P_1 becomes a false hit, and is moved from RS to FH. The server then updates DST_{max} to $DST(Q, P_2)$, which leads to the shrinking of the SR as shown in Figure 7b. The next NN p_3 of q falls out of SR, and NN retrieval terminates with $RS = \{P_2\}$ and $FH = \{P_1\}$. Next, the server performs an authenticated range query $q_R = (q, DST_{max})$, where $DST_{max} = DST(Q, P_2)$. The result of q_R contains the low-dimensional representations of p_1 and p_2. Considering that the MR-Tree in Figure 7 consists of root node N_1, and its two children N_2 and N_3, the VO of q_R is $VO_R = [[result, false\_hit, (p_3, h_{P3})] (h_{N3}, MBR_{N3})]$, i.e., it is computed as in the traditional MR-Tree, with two differences: (i) p_3 is inserted along with its associated digest h_{P3} (since this is necessary for computing h_{root}), and (ii) two tokens are used as placeholders, one corresponding to the reduced representation of an actual result (*result*), and one of a false hit (*false_hit*). Note that it is redundant to provide p_1 and p_2 because the client can derive them from P_1 and P_2 included in $RS \cup FH$. Signature sig, RS, FH and VO_R are transmitted to the client.



(a) 1st NN candidate (b) 2nd NN candidate

Fig. 7 Finding the 1NN

Having RS, FH and VO_R, the client can re-construct the signed digest h_{root} using the verification algorithm of the MR-Tree with two alterations: (i) it derives the reduced point p and digest h_P of every point P in RS (FH), and substitutes its corresponding *result* (*false_hit*) token in VO_R with (p, h_P); (ii) it computes the level-1 digests as in Figure 5.

Figure 8 provides the pseudo code for the verification algorithm. After re-constructing and matching h_{root} against *sig*, the procedure computes DST_{max} over *RS*. It then establishes that the points in *FH* are indeed false hits. Finally, it confirms that all points and MBRs in VO_R (other than the *result*/*false_hit* tokens) lie further than DST_{max} from *q*. In the running example, the client first computes p_1 and p_2 from P_1 and P_2, and MBR_{N2} using p_1, p_2 and p_3. Then, it generates $h_{N2} = h(h_{p1} \mid h_{P1} \mid h_{p2} \mid h_{P2} \mid h_{p3} \mid h_{P3})$, and subsequently $h_{root} = (h_{N2} \mid MBR_{N2} \mid h_{N3} \mid MBR_{N3})$. If h_{root} matches *sig*, the client evaluates $DST_{max} = DST(Q, P_2)$ and ascertains that $DST(Q, P_1) > DST_{max}$. Eventually, it establishes that the distance of p_3 from *q*, as well as the minimum distance between MBR_{N3} and *q*, is indeed larger than DST_{max}.

Algorithm *AMN_client* (*R*, VO_R)
1. *MR_verify* (*RS*, *FH*, VO_R)
2. Compute DST_{max} over *RS*
3. For each point P_i in *FH*, Verify $DST(Q, P_i) > DST_{max}$
4. For each point p_i (MBR_i) in VO_R,
5. Verify $dst(q, p_i) > DST_{max}$ ($mindist(q, MBR_i) > DST_{max}$)

Fig. 8 *k*NN verification at the client

Proof of Correctness. We distinguish two cases: (i) the server returns to the client a set *RS′*, which is derived from the correct *RS* after substituting a point *P* with another *P′* that does not exist in *DB* (i.e., *P′* is bogus). Since the signature does not incorporate authentication information about *P′*, the re-constructed h_{root} does not match *sig* due to the collision-resistance of the hash function and, thus, the client is alarmed. (ii) The server returns to the client a set *RS′*, which is derived from *RS* after substituting a point *P* with another *P′* that belongs to *DB* (i.e., *P′* is legitimate). Let DST_{max} (DST'_{max}) denote the distance between *Q* and its k^{th} NN in *RS* (*RS′*). Since *P′* is not a true result, $DST'_{max} > DST_{max}$. The server also generates a VO_R' which verifies *RS′*, i.e., it authenticates range $q_R' = (q, DST'_{max})$. Point *p* lies in range q_R' because $dst(q, p) \leq DST(Q, P) \leq DST_{max} < DST'_{max}$. Given that *P* is not in *RS′*, it must be included in *FH* (otherwise, the verification of VO_R' will fail). Thus, Line 3 of algorithm *AMN_client* detects the violation and the client is alarmed. □

Since AMN follows the optimal framework of Figure 1, it is also optimal in terms of *DST* distance computations. A more important question in our setting regards its performance in terms of the communication overhead and the verification cost at the client. Recall that, along with the result, the client needs a representation for each $P \in FH$ in order to verify that indeed *P* is a false hit. For typical data series applications, *D* can be up to 1-3 orders of magnitude larger than *d*. Therefore, *FH* emerges as the most important performance factor in authenticated NN search, especially for very high-dimensional data. However, as we show next, *FH* does not have to include the complete representations of false hits.

4 Communication-efficient AMN

Depending on the dimensionality reduction technique, the values *D*, *d*, and *k*, and the dataset characteristics, there may be numerous false hits in *FH*, each containing hundreds or thousands (i.e., *D*) values. Next, we propose *communication-efficient* AMN (C-AMN), which decreases the size of the false hits, significantly reducing the transmission and verification cost without compromising the security of AMN. Section 4.1 explains the main concepts of C-AMN, whereas Section 4.2 presents the concrete algorithm for false hit reduction.

4.1 General Framework

We illustrate the main idea of C-AMN through Figure 9, where (i) *Q*, P_1, P_2 are 16-dimensional time series (i.e., D = 16), (ii) *q*, p_1, p_2 are their 2-dimensional representations obtained by taking the first two coefficients of the DFT decomposition (i.e., $d = 2$), and (iii) *DST* and *dst* correspond to Euclidean distances. The coefficients and distances are computed using the real values of *Q*, P_1, and P_2. Since $DST(Q, P_2) < DST(Q, P_1)$, P_2 is the 1NN of *Q*. Furthermore, $dst(q, p_2) > dst(q, p_1)$, which signifies that P_1 is a false hit; thus, all its 16 values are included in *FH* by AMN.

Full Representations																
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
***Q*:**	5	6	10	3	7	40	2	2	5	20	25	10	5	3	3	7
P_1:	25	6	10	4	7	20	2	5	5	20	25	10	5	3	13	7
P_2:	24	15	11	3	22	40	1	0	4	18	20	20	6	5	4	7

Reduced Representations		
***q*:**	9.5625	-2.1908
p_1:	10.4375	-0.1698
p_2:	12.5	-0.0279

Distances	
$DST(Q, P_1)$ = 30. 1662	**$dst(q, p_1)$** = 2.2023
$DST(Q, P_2)$ = 28.443 (DST_{max})	**$dst(q, p_2)$** = 3.6479

Fig. 9 Representations of *Q*, P_1, P_2, *q*, p_1, p_2 (of Figure 6)

Let $P[i]$ ($1 \leq i \leq D$) be the i^{th} value of *P* (e.g., $P_1[2] = 6$ in Figure 8), and *SP* be an ordered subset of *P* values, e.g., $SP_1 = (P_1[1], P_1[6], P_1[15]) = (25, 20, 13)$. SQ_1 contains the values of *Q* in the same positions as SP_1, e.g., $SQ_1 = (Q[1], Q[6], Q[15]) = (5, 40, 3)$. Then, $DST(SQ_1, SP_1) = 28.4605 > DST(Q, P_2)$ (= 28.443). Thus, instead of the complete P_1, it suffices for the server to include into *FH* only SP_1. The client computes SQ_1 from *Q*, establishes that $DST(Q, P_2) < DST(SQ_1, SP_1) \leq DST(Q, P_1)$, and confirms P_1 as a false hit. Assuming that each value has size S_v = 8 bytes, SP_1 consumes 24 bytes, as opposed to 128 bytes for P_1. By sending SP_1 to the client, the server reduces the communication cost significantly (this is the largest possible reduction for the current example). However, it must also prove that SP_1 is not falsified.

We next demonstrate a solution to this problem, assuming, for simplicity, that *D* is a power of 2. The server maintains an MH-Tree over P_1 as shown in Figure 10. Recall that in AMN, the digest h_{P1} included in the MR-Tree leaf accommodating p_1 is computed on the concatenation of the binary representation of the values of P_1, i.e., $h_{P1} = h(P_1[1] \mid P_1[2] \mid \dots \mid P_1[16])$. Instead, in C-AMN, h_{P1} is the digest h_{31} in the root of the MH-Tree, which summarizes authentication information about the entire P_1. The rest of the MR-Tree structure remains the same. Returning to our running example, after producing SP_1, the server constructs a *VO* (denoted as VO_{SP1}) that can verify SP_1. VO_{SP1} contains the necessary components for re-constructing h_{P1}. These components are highlighted in dark grey in Figure 10: VO_{SP1} = [[[[25 h_2] h_{18}] [[h_5 20] h_{20}]] [h_{27} [h_{23} [13 h_{16}]]]]. Note that the positions[4] of SP_1 values can be computed based on the structure of VO_{SP1}. For example, prior to 20 in VO_{SP1}, there is one value (25), two level-0 digests (h_2, h_5) and a level-1 digest (h_{18}). This implies that 20 is the 6[th] value in P_1.

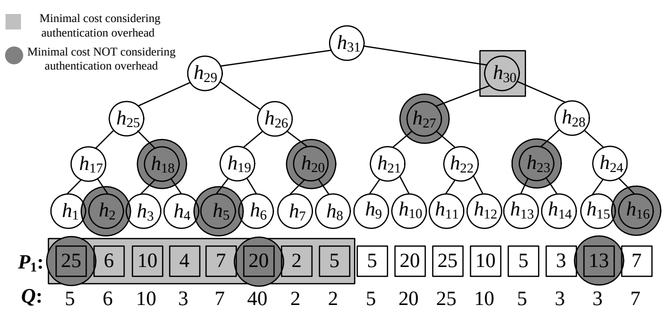


Fig. 10 MH-Tree over P_1

Given VO_{SP1}, the client produces h_{P1} by performing successive hash operations on the contents of corresponding '[' and ']' tokens as explained in Section 2.3. Finally, recall that the digest of p_1, h_{p1}, is needed for computing the level-1 digest of N_2, $h_{N2} = h(h_{p1} \mid h_{P1} \mid h_{p2} \mid h_{P2} \mid h_{p3} \mid h_{P3})$. In AMN, the client could compute h_{p1} by first reducing P_1 (fully included in *FH*) to p_1 and then calculating $h_{p1} = h(p_1)$. In C-AMN, however, the client does not have the entire P_1, and it cannot generate p_1 from SP_1. Therefore, the server must send h_{p1} to the client as well. The additional authentication information increases the communication overhead. Continuing the example, and assuming that a digest has size S_h = 20 bytes (a typical value), VO_{SP1} and h_{p1} consume a total of 184 bytes, which is larger than P_1 (128 bytes). This is due to the fact that the values of SP_1 are dispersed in the MH-Tree, causing the insertion of numerous digests in VO_{SP1}. Consider now that we modify SP_1 to include values $P_1[1]$ - $P_1[8]$. In this case, $DST(SQ_1, SP_1)$ = 28.46 > DST_{max}, and VO_{SP1} = [[[[25 6][10 4]][[7 20][2 5]]] h_{30}] contains a single digest, h_{30} (light grey in Figure 9). The total cost is now 106 bytes, which is actually the lowest possible in this example. Note that for simplicity, we omit the headers of the *VO* elements (see Section 2.4) in the size computations because their space consumption is negligible.

Summarizing, C-AMN aims at replacing each false hit *P* with a verifiable representation *SP* that consumes less space. C-AMN necessitates some modifications over the AMN indexing scheme: (i) for every $P \in DB$, the server maintains a MH-Tree, and (ii) each digest h_P at the leaf level of the MR-Tree is set to the root digest of the MH-Tree of *P*. Query processing at the server proceeds as in Figure 6, but after computing *RS*, *FH* and VO_R, the server calls a function *ReduceFH* to replace every false hit *P* with a pair (VO_{SP}, h_p), where VO_{SP} contains the subset *SP* of *P* that proves that *P* is a false hit, along with verification information, and h_p is the digest of *P*'s indexed point *p*.

Verification at the client is similar to *AMN_client* (Figure 8), with the following alterations: (i) *MR_verify* computes, for every pair (VO_{SP}, h_p) $\in$ *FH*, the digest h_P of the corresponding false hit *P*, simply simulating the initial calculation of h_{root} in the MH-Tree of *P*. (ii) For every VO_{SP} in *FH*, Line 3 extracts *SP* from VO_{SP}, computes the respective *SQ* of *Q*, and verifies that $DST(SQ, SP) > DST_{max}$. The proof of correctness of C-AMN is identical to that in AMN, given that there is no way for the server to falsify *SP* (otherwise the re-constructed h_{root} would not match *sig* due to the collision-resistance property of the hash function).

4.2 False Hit Reduction Algorithm

Ideally, for each false hit *P*, *ReduceFH* should derive the subset *SP* with the minimum length. Intuitively, this task is at least as difficult as the *Knapsack Problem*; we need to select a subset of items (*SP* of *P* values), each assigned a cost (communication overhead) and a weight (distance $DST(SQ, SP)$), such that the sum of costs is minimized and the sum of weights exceeds DST_{max}. An additional complication is that, when we select one item, the cost of the rest changes (i.e., unlike knapsack, where the cost is fixed).

Theorem 1. *Determining the optimal subset SP of P, which leads to the minimal VO_{SP} under the constraint that $DST(SQ, SP) > DST_{max}$, is NP-hard.*

Proof. This can be proven by a straightforward polynomial reduction from the NP-Hard *Precedence Constraint Knapsack Problem* [9]. □

We next propose a *greedy* version of *ReduceFH*. Let $dist_i = (P[i]-Q[i])^2$ be the contribution of $P[i]$ to $DST(SQ,SP)$, and $comm_i$ the additional cost if $P[i]$ is inserted in the current VO_{SP}. To assist the iterative decisions of our greedy algorithm, we assign to each unselected value $P[i]$ a *benefit* $B[i] = dist_i / comm_i$ that captures the potential gain if $P[i]$ is included in *SP*. For example, in Figure 9, suppose that $SP_1 = (P_1[6])$; hence, VO_{SP1} = [[h_{25} [[h_5 20] h_{20}]] h_{30}]. If we insert $P_1[1]$ in SP_1, the new VO_{SP1} verifying $SP_1 = (P_1[1], P_1[6])$ becomes [[[[20 h_2] h_{18}] [[h_5 20] h_{20}]] h_{30}]. The increase of VO_{SP1} due to the insertion of $P_1[1]$ is $comm_1 = S_h + S_v = 28$ ([[20 h_2] h_{18}] substitutes h_{25} in VO_{SP1}). At each step, the unselected $P[i]$ with the *highest* benefit $B[i]$ is greedily inserted in *SP*. Intuitively, we prefer values with large *dist* (to satisfy $DST(SQ, SP) > DST_{max}$ with as few values as possible), and small *comm* (to keep VO_{SP} as small as possible). After

[4] The positions of the P_1 values used in SP_1 are necessary so that the client can compute SQ_1 from *Q*.

inserting $P[i]$ in SP, we set $B[i] = 0$ (so that it cannot be selected again). We also update (if necessary) the benefits of the yet unselected P values, since *comm* depends on the VO_{SP} verifying the *current* SP. The algorithm terminates when the condition $DST(SQ, SP) > DST_{max}$ is satisfied.

The most expensive operation is the update of the benefits at each step. A naive algorithm would take $O(D^2)$ time in overall; upon each selection, it would scan *all* unselected P values and update their benefits as necessary. We next present an implementation with complexity $O(D \cdot \log D)$. We demonstrate the process on P_1 of our running example using Figure 11. The algorithm initially sets $SP_1 = \varnothing$, and $VO_{SP1} = \varnothing$. Moreover, it assigns $comm_i = 4 \cdot S_h + S_v = 84$ for all $P_1[i]$, since inserting any $P_1[i]$ in VO_{SP1} would cause its increase by four digests and one value. Subsequently, it calculates the benefit vector B of P_1, and constructs an auxiliary binary tree on B where: (i) each leaf node n_i stores $B[i]$, and (ii) inner nodes are created bottom-up, storing the *maximum* value in their sub tree. In the first step, *ReduceFH* performs a root to leaf traversal of the tree, visiting always the node with the maximum value. In Figure 11a, the leaves with the maximum benefits are n_1 and n_6 because they have the largest distances to $Q_1[1]$ and $Q_1[6]$, respectively. Suppose that the algorithm (randomly) visits n_6; it inserts $P_1[6] = 20$ into SP_1, and updates the value of n_6 to 0 (so that n_6 will not be selected in the future). VO_{SP1} becomes [[h_{25} [[h_5 20] h_{20}] h_{30}], and $DST(SQ, SP_1) = 20 < DST_{max}$. Therefore, the algorithm continues.

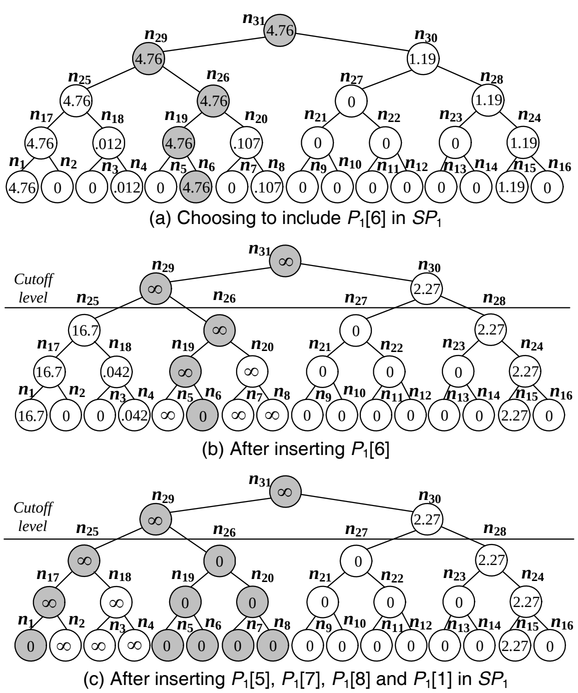


Fig. 11 Using the auxiliary tree of *ReduceFH*

The next action regards the updating of the benefits in the rest of the tree. Observe that, if after the inclusion of $P_1[6]$ in SP_1, we also insert $P_1[5]$, digest h_5 in VO_{SP1} will be replaced by a value. Thus, the size of VO_{SP1} will *decrease* because S_h (20 bytes) is larger than S_v (8 bytes). In general, depending on the relative size of S_h and S_v, there is a *cutoff level cl*, such that, if a value $P[i]$ is inserted in SP, all the benefits in the same sub-tree as n_i rooted at cl will become negative because of the reduction of the communication cost. To include the corresponding values in SP, we set their respective benefits to ∞. In our example $cl = 2$, and the selection of $P_1[6]$ will assign ∞ to the benefits of nodes n_5, n_7, n_8, n_{19}, n_{20}, n_{26}, n_{29}, n_{31}. The benefits of the remaining nodes are updated as shown in Figure 11b.

In the three next steps, *ReduceFH* follows the nodes with ∞ benefit and inserts $P_1[5]$, $P_1[7]$, and $P_1[8]$ into SP_1, updating their corresponding benefits (as well as the values of their ancestors) to 0. Observe that these insertions do not influence the benefits of the nodes in the sub trees rooted at n_{25} and n_{30}. The general rule is that $comm_i$ changes, only if the insertion of a $P[j]$ into SP causes the substitution (in VO_{SP}) of a digest that was taken into account in $comm_i$. Note that each such digest corresponds to a sub-tree of the auxiliary tree in *ReduceFH*, since the latter has a one-to-one correspondence with the MH-Tree of P. For example, in Figure 11b (i.e., after $P_1[6]$ has been inserted in SP_1), $comm_9$ considers digests h_{10}, h_{22} and h_{28}, corresponding to nodes n_{10}, n_{22} and n_{28}, respectively. After the insertion of $P_1[5]$, $comm_9$ still involves the above digests and, thus, does not change. *ReduceFH* marks every visited node of the tree during the selection of the next P_1 value.

The marked nodes in Figure 11 are highlighted in grey color. A marked node signifies that there is at least one leaf n_i in its sub-tree such that $P_1[i]$ is in SP_1. In each new traversal, if a visited node n is not marked, then the benefits in the sub-tree of n's sibling n_s must be updated. For instance, in Figure 11b, $comm_9$ takes into account h_{28} corresponding to n_{28}, which is not marked. If n_{28} is visited, the benefit of n_9 (i.e., of $P_1[9]$), and of all leaves in the sub-tree rooted at n_{27}, will change. Returning to our example, at this point, $SP_1 = (P_1[6], P_1[5], P_1[7], P_1[8]) = (20, 7, 2, 5)$, and $DST(SQ, SP_1) = 20.224 < DST_{max}$. In the next step, the procedure inserts $P_1[1] = 25$ into SP_1 (it has the largest benefit), and updates the benefits of n_2, n_3, n_4, n_{18}, n_{25}, n_{29}, n_{31} to ∞, as shown in Figure 11c. Subsequently, $P_1[2]$, $P_1[3]$, $P_1[4]$ are inserted because their benefit is ∞. The resulting $SP_1 = (P_1[6], P_1[5], P_1[7], P_1[8], P_1[1], P_1[2], P_1[3], P_1[4]) = (20, 7, 2, 5, 25, 6, 10, 4)$ satisfies the constraint $DST(SQ, SP_1) = 28.461 > DST_{max}$. Its corresponding VO is equal to VO_{SP1} = [[[[25 6][10 4]][[7 20][2 5]]] h_{30}]. Observe that SP_1 and VO_{SP1} are the same as the ones we constructed for Figure 9, and are optimal for this example.

Figure 12 provides the generalized pseudo code of *ReduceFH*. Function *ConstructTree* in Line 2 computes the initial benefit vector B of P and builds a binary tree over it. The loop in Lines 3-5 first calls function *FindNextValue*, which retrieves the next value *P_v* with the maximum benefit. It also updates the benefits of the affected nodes. As soon as the condition in Line 3 is satisfied, Lines 6-8 include the values with benefit ∞ into SP as well. *ReduceFH* eventually returns SP.

Algorithm *ReduceFH*(P, Q, DST_{max})

1. Set $SP = \varnothing$
2. $n_{root} = ConstructTree(P)$
3. While $DST(SQ, SP) \leq DST_{max}$
4. $\quad P\_v = FindNextValue(n_{root})$
5. $\quad$ Insert P_v in SP
6. While $n_{root}.B = \infty$
7. $\quad P\_v = FindNextValue(n_{root})$
8. $\quad$ Insert P_v in SP
9. Return SP

Fig. 12 Algorithm *ReduceFH*

Complexity Analysis. The construction of the tree takes O(D) time. Furthermore, the loops in Lines 3-5 and 6-8 involve $O(D)$ steps. *FindNextValue* involves two operations: (i) finding the value with the maximum benefit, which takes $O(\log D)$ time, and updating the benefits of the affected nodes. Instead of calculating the latter cost in each step of *FindNextValue*, we will compute it collectively in the entire execution of *ReduceFH*. We mentioned that the benefit of a node n_i is updated, only when the sub-tree corresponding to a digest involved in $comm_i$ is marked for the first time. Since the maximum number of digests entailed in $comm_i$ is log D, the total time required for updating all values is $O(D \cdot \log D)$, which is also the complexity of *ReduceFH* for each false hit.

5 AMN IN DISTRIBUTED SERVERS

In this setting we assume that the database is horizontally partitioned and distributed over m (>1) servers. Specifically, each server S^i stores a subset DB^i such that: $DB^1 \cup .. \cup DB^m = DB$ and $DB^i \cap DB^j = \varnothing$, $\forall\ 1 \leq i, j \leq m$. In addition, S^i maintains an MR-Tree on the corresponding reduced data set db^i, which is signed by a signature sig^i. A query result comprises the kNNs over all servers. Minimization of transmissions (of the high-dimensional data) is particularly important for this setting, especially for large values of m. Section 5.1 presents SD-AMN (short for *simple distributed* AMN), used as a benchmark in our experimental evaluation. Section 5.2 proposes ID-AMN (short for *incremental distributed AMN*), a more elaborate method, which quickly eliminates servers that cannot contribute results.

5.1 SIMPLE DISTRIBUTED AMN

In SD-AMN, a client sends its kNN query Q to all servers. Each server S^i retrieves the partial result RS^i on the local DB^i using the conventional multi-step algorithm of Figure 1, and generates a vector $\mathbf{kDST}^i$ with the *distance values* of the kNN set RS^i in S^i. The client collects the vectors from the servers and determines the global k^{th} nearest distance DST_{max} over all $m \cdot k$ collected distances. Then, it transmits a range $q_R = (q, DST_{max})$. Each server S^i executes q_R using its MR-Tree and returns VO_R^i, RS^i and FH^i. VO_R^i has the same meaning as in centralized processing, i.e., it is the VO of q_R. RS^i (resp. FH^i) is a set of results (resp. false hits), i.e., points of db^i that fall in q_R and whose high-dimensional representations have distance from Q smaller (resp. larger) than DST_{max}. The size of FH can be reduced through C-AMN.

We demonstrate SD-AMN using the example of Figure 13a, assuming a 3NN query Q. There are four servers, each containing four points. For ease of illustration, we sort these points on their distance to Q and display their DST, e.g., P_1^2 is the first point in S^2 and $DST(Q, P_1^2) = 3$. The diagram also includes the distances in the reduced space, e.g., $dst(q, p_1^2) = 2$. Given $\mathbf{kDST}^1 = (1, 2, 5)$, $\mathbf{kDST}^2 = (3, 6, 12)$, $\mathbf{kDST}^3 = (5, 7, 10)$ and $\mathbf{kDST}^4 = (7, 8, 9)$, the client computes $DST_{max} = 3$ (the first two NNs are in S^1, and the third one in S^3) and transmits the range $q_R = (q, 3)$. S^1 returns VO_R^1, $RS^1 = \{P_1^1, P_2^1\}$ and $FH^1 = \{P_3^1, P_4^1\}$. P_3^1 and P_4^1 are necessary, in order for the client to establish that they are indeed false hits. At the other extreme, S^4 returns VO_R^4, $RS^4 = FH^4 = \varnothing$, as it does not contain results or false hits (for all points in S^4, their dst from q exceeds 3). If each VO_R^i is verified successfully, and for every point P in each FH^i it holds $DST(Q, P) \geq DST_{max}$, then the client is assured that RS is correct.

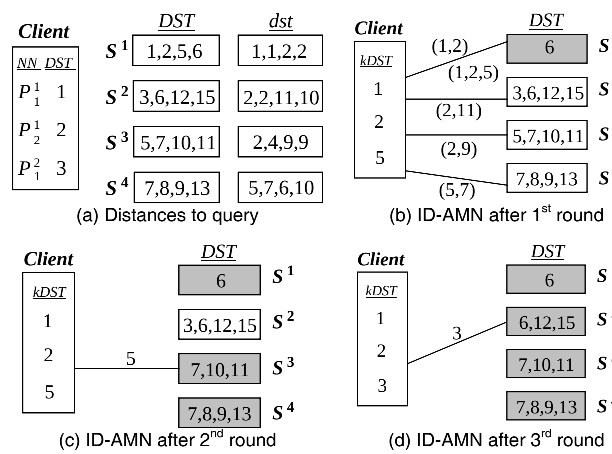


Fig. 13 Distributed authenticated kNN processing

Proof of Correctness. In SD-AMN, the client computes a DST_{max} (e.g., 3 in Figure 13a) such that the range (Q, DST_{max}) contains exactly k high dimensional points in $DB^1 \cup .. \cup DB^m$. At the last step of SD-AMN all servers perform a verifiable range $q_R = (q, DST_{max})$, during which they cannot cheat. Thus, they can only misreport the distance vectors $\mathbf{kDST}^i$, leading the client to compute a DST'_{max} that is different from the real DST_{max}. We distinguish two cases with respect to the relationship between DST'_{max} and the DST_{max}. If $DST'_{max} > DST_{max}$, the client will obtain all results (possibly, in addition to more points) through the verifiable range (q, DST'_{max}), and will detect the discrepancy between DST'_{max} and DST_{max}. If $DST'_{max} < DST_{max}$, the client receives fewer than k objects in $RS^1 \cup .. \cup RS^m$ and is alarmed. Therefore, it can always establish result correctness. □

As an example, suppose that in Figure 13a S^2 misreports $DST(Q, P_1^2)$ as 4 (instead of 3). Consequently, the client computes the distance to its 3rd NN as $DST'_{max} = 4 > DST_{max}$, and receives a VO for range (q, 4) from every server. S^2 will then have to return P_1^2 as a result, and the difference between the reported and actual $DST(Q, P_1^2)$ will be revealed. For the second case, consider that that server S^2 misreports $DST(Q, P_1^2)$ as 2, leading the client to compute $DST'_{\max} = 2 < DST_{max}$. Upon receiving the *VO*s for range (q, 2) from all servers, it discovers that there are only 2 points (P_1^1,P_2^1) in $RS^1 \cup .. \cup RS^4$. Note that servers can misreport distances of points in *DB-RS-FH* without being caught, provided that the false *DST* reported are larger than DST_{max}. In the example of Figure 13a, there are no low dimensional objects of S^4 within (q, 3). Therefore, S^4 can send to the client any vector that contains distances larger than 3 because the verification of (q, 3) does not involve the transmission of any point ($RS^4 = FH^4 = \varnothing$). Clearly, this type of false values does not affect the result.

5.2 INCREMENTAL DISTRIBUTED AMN

SD-AMN is optimal in terms of high-dimensional point transmissions because the client receives D-dimensional representations only for points in q_R. All these points (results and false hits) are necessary to establish correctness anyway. However, it must transmit Q to all servers. Moreover, each server S^i has to compute RS^i although none of the points of RS^i may participate in the global result (e.g., S^4 in Figure 12). ID-AMN avoids these problems by gradually eliminating servers that cannot contribute results. Specifically, ID-AMN incrementally retrieves distance values from servers to compute the final DST_{max}, postponing local NN computations at the servers until they are required. We present the pseudo code of ID-AMN (at the client) in Figure 14, and explain its functionality by continuing the example of Figure 13 ($k = 3$).

Initially, the client marks each server S^i as a candidate, and transmits q to S^i, which responds with two values: the dst^i ($kdst^i$) of its 1st (kth) NN in the low dimensional space. For instance, in Figure 13, the client receives (1, 2), (2, 11), (2, 9), (5, 7) from S^1, S^2, S^3, S^4, respectively. Intuitively, a low $kdst^i$ implies a promising server that may contain k good results. DST^i and dst^i are used for server pruning and selection, to be discussed shortly. Let S^j, be the server (e.g., S^1) with the minimum $kdst^j$. The client directs Q to S^j and obtains a vector ***kDST*** with the distance values of the kNN set RS^j in S^j. DST_{max} is set to the kth distance in ***kDST*** and S^j ceases to be a candidate. Continuing the example, ***kDST*** = (1, 2, 5), DST_{max} = 5. Figure 13b illustrates the server-to-client transmissions during these steps.

The while loop (Lines 8-17) starts by eliminating each server such that $DST^i \geq DST_{max}$ (initially $DST^i = dst^i$). For instance, $DST^4 = 7 \geq DST_{max} = 5$, and the client discards S^4 without sending Q. Since the subsequent verification of S^4 does not require Q either, there is no transmission of high-dimensional data (query, or points) between the client and S^4. Line 11 selects the candidate server S^i with the minimum DST^i, and asks for the distance DST_{new} of the next NN in S^i. If $DST_{new} \geq DST_{max}$, S^i is purged. Assuming that the selected server is S^3 ($DST^3 = DST^2 = 2$), then $DST(Q, P_1^3) = 5 \geq DST_{max} = 5$, causing the elimination of S^3 without changing ***kDST***. Figure 13c shows the pruned servers S^1, S^3, S^4 in grey. The next iteration of the loop selects the last candidate S^2, and retrieves $DST_{new} = 3$. Since $3 < DST_{max}$, DST_{new} is inserted into ***kDST***, and DST_{max} changes to 3. The loop terminates because all servers have been eliminated (Figure 13d). Lines 18-20 simply verify the range $q_R = (q, DST_{max})$ in each server. All the result points (RS^i), as well as false hits (FH^i) are transmitted during this step. The client generates the final result RS locally from the union of all RS^i. C-AMN can be applied to reduce the size of false hits. Note that Line 12 may call *get_next_ smallest_DST* multiple times on the same server S^i. In this case, the client needs to transmit the full query Q only the first time; for subsequent requests, it suffices to send the query ID.

Algorithm *ID-AMN_client* (*Q*, *k*)
1. For each server S^i
2.   Set *Candidate*[i]=1;
3.   (dst^i, $kdst^i$,)= *get_ smallest_dist*(q, S^i)
4. $DST^i = dst^i$
5. Let S^j be the server with the minimum $kdst^j$
6. Set vector ***kDST*** = *get_k_smallest_DSTs*(Q, S^j)
7. Set DST_{max} = maximum value in ***kDST***; Set *Candidate*[j]=0
8. While there are candidate servers
9.   For each server S^i
10.   If $DST^i \geq DST_{max}$, Set *Candidate*[i]=0
11.   Select candidate server S^i with minimum DST^i
12.   Set DST_{new} = *get_next_ smallest_DST*(Q, S^i) from server S^i
13.   If $DST_{new} \geq DST_{max}$, Set *Candidate* [i]=0
14.   Else // $DST_{new} < DST_{max}$
15.   Insert DST_{new} into ***kDST***
16.   Set DST_{max} = maximum value in ***kDST*** ;
17. $DST^i = DST_{new}$
18. For each server S^i
19.   (VO_R^i, RS^i, FH^i) = *MR_Range*((q, DST_{max}), $root^i$)
20.   *Verify*(VO_R^i) and incorporate RS^i into RS

Fig. 14 Incremental distributed AMN (client)

Proof of Correctness. The client obtains all results and false hits at the end through the verifiable range (Lines 18-20). As shown in the proof SD-AMN, any *DST* misreporting that leads to the computation of a $DST'_{max} \neq DST_{max}$ can be detected by the client. Let us now consider that some server S^i sends false dst^i and $kdst^i$. The value of $kdst^i$ is only used as an estimator for the selection of the initial server (Line 5), and it only affects the efficiency (but not the correctness) of the algorithm. For instance, if S^3 reports $kdst^3 = 1$ (instead of 9), it will become the initial server, increasing the communication overhead (S^4 cannot be immediately eliminated), without however altering the result. Moreover, as discussed in Section 5.1, any false distance smaller than DST_{max} will be caught by the verification. Similarly, dst^i is used as a lower bound for DST^i. If S^i sends a value of dst^i lower than the actual one, it can only be selected earlier than necessary during the while loop without affecting correctness. On the other hand, if the transmitted dst^i exceeds the actual one, (i) S^i is selected later during the loop, or (ii) eliminated altogether if the reported dst^i exceeds DST_{max}. Case (i) only affects the efficiency, whereas case (ii) is detected during the verification because S^i has to send objects within the range $q_R = (q, DST_{max})$. □

Similar to SD-AMN, ID-AMN is optimal in terms of data point transmissions. Now let us consider query transmissions. Ideally, an optimal algorithm would send the complete query Q only to servers that contribute nearest neighbours (S^1, S^2 in Figure 13) because the distances of these NNs are necessary for obtaining the final value of DST_{max}. Additionally, ID-AMN sends Q to some *false candidates* (e.g., S^3) that cannot be eliminated. Specifically, let DST_1^i be the distance of the 1st NN in S^i. S^i is a false candidate, if $dst^i < DST_{max} < DST_1^i$. Regarding the CPU cost, in ID-AMN some eliminated servers (e.g., S^4) do not perform any high-dimensional distance computations. False candidates return a single DST_1^i; thus, they need to retrieve the first local NN (which may involve multiple DST computations, if there are false hits). The rest of the servers must retrieve either k^i or (k^i+1) NNs, where k^i ($\leq k$) is the contribution of S^i to the final kNN set. The distance of the additional (+1) NN is sometimes required to eliminate S^i. In comparison, SD-AMN, transmits Q to all servers, and each server performs the necessary computations to obtain the k local NNs.

6 Experimental Evaluation

We use four real datasets that capture different combinations of dimensionality D, cardinality N, and application domain: (i) *Corel* (D = 64, N = 68040), (ii) *Chlorine* (D = 128, N = 4310), (iii) *Stock* (D = 512, N = 10000), and (iv) *Mallat* (D = 1024, N = 2400). *Corel* [5] can be downloaded from *archive.ics.uci.edu/ml/*, while the rest are available at: *www.cs.ucr.edu/~eamonn/time_series_data/*. We decrease the dimensionality of each dataset using Chebyshev polynomials [4]. The value of d is a parameter with range [2, 16] and default value 8. Each reduced dataset is indexed by an MR-Tree using a page size of 4KB. Every digest is created by SHA-1 [16]. We assume that both DST and dst are based on the Euclidean distance. Section 6.1 compares AMN and C-AMN considering a single server. Section 6.2 evaluates SD-AMN and ID-AMN assuming multiple servers.

6.1 Single Server

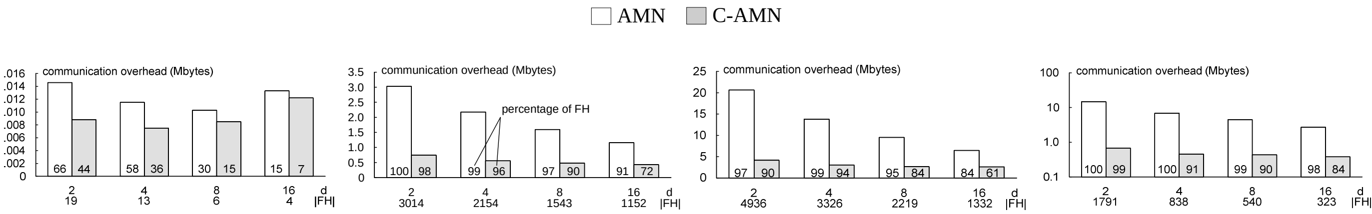
The measures of interest are the communication overhead, and the CPU cost at the server and the client. We assess the communication overhead based on the verification information sent to the client. The transmission of the query and the result is omitted because it is necessary in any method. The CPU cost is measured in terms of the *elementary distance computations*. Specifically, D elementary computations are required to derive the Euclidean distance of two D-dimensional points. We exclude the I/O cost at the server because it is identical for both AMN and C-AMN (and similar to that of the conventional multi-step algorithm) since in any case, we have to retrieve the low dimensional NNs using the MR-Tree. For each experiment we select a random data point as the query, and report the average results over 10 queries.

Figure 15 fixes the number k of NNs to 8, and investigates the effect of d on the communication overhead of the verification information. Specifically, the overhead is measured in *Mbytes*, assuming that each value consumes S_v=8 bytes (a double precision number) and each digest is S_h=20 bytes (typical size for SHA-1). We indicate the number $|FH|$ of false hits below the x-axis. As d increases, $|FH|$ drops because the reduced representation captures more information about the corresponding point. In all cases, C-AMN leads to a significant decline of the overhead. The savings grow with D, and exceed an order of magnitude for *Mallat*, because long series provide more optimizations opportunities. On the other hand, the gains decrease as d grows due to the smaller FH. In order to demonstrate the effect of the false hits, we include inside each column of the diagrams, the contribution of FH as a percentage of the total overhead. For high D and low d, FH constitutes the dominant factor, especially for AMN (e.g., at least 98% in *Mallat*), corroborating the importance of C-AMN.

The absolute overhead is lower (in both AMN and C-AMN) for high values of d due to the decrease of $|FH|$. The exception is *Corel*, where the communication cost actually grows when d exceeds a threshold (8 for AMN, 4 for C-AMN). This is explained as follows. A typical record (i.e., image) in *Corel* has very low values (<0.005) on most (>60) dimensions, and relatively large values (>0.1) on the rest. Furthermore, the large values of different records usually concentrate on different dimensions. Dimensionality reduction using Chebyshev polynomials [4] captures effectively those important dimensions even for low d. Consequently, there is a small number of false hits (for d=2, $|FH| \approx 0.28\%$ of N, whereas in the other datasets $|FH|$ is 50-75% of N). As d grows, $|FH|$ does not drop significantly; on the other hand, the verification information transmitted to the client contains more boundary records and node MBRs, increasing the *VO* size.

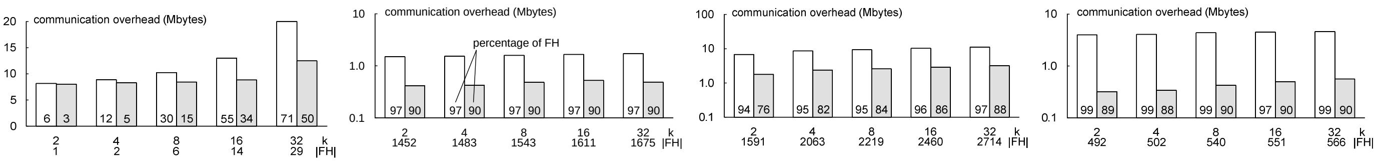
Figure 16 illustrates the communication overhead as a function of the nearest neighbors k to be retrieved (d=8). Note that the minimum value of k is 2 because the query is also a data point, i.e., its first NN is itself. The number of false hits increases with k, leading to higher transmission cost. However, compared to Figure 15, the difference is rather small because k has a lower impact on $|FH|$ than d. For the same reason, the absolute performance of both methods is rather insensitive to k. The benefits of C-AMN are again evident, especially if $D \geq 128$.

[5] *Corel* has four attribute sets. We use the first two sets (Color Histogram and Color Histogram Layout, of 32 attributes each) to derive the 64D dataset used in the experiments.



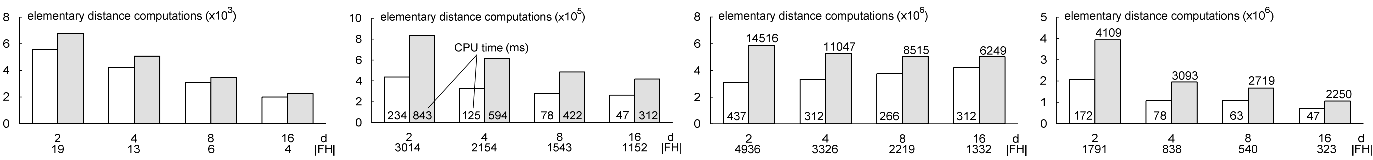
(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 15 Communication overhead vs. d (k=8)



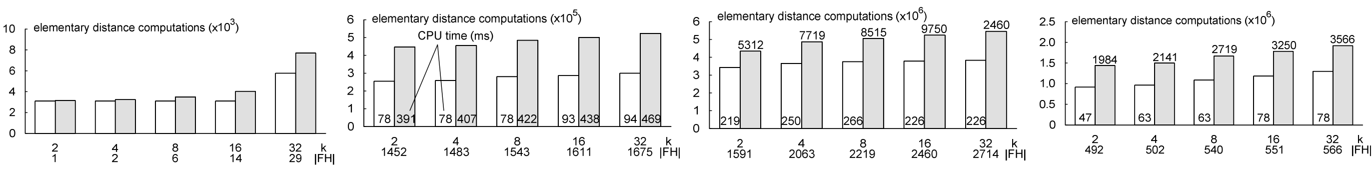
(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 16 Communication overhead vs. k (d=8)



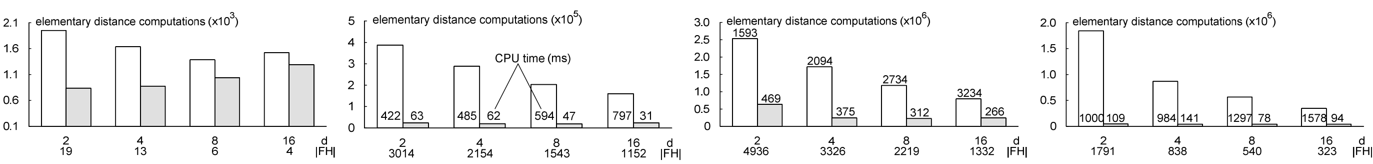
(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 17 Server computations vs. d (k=8)



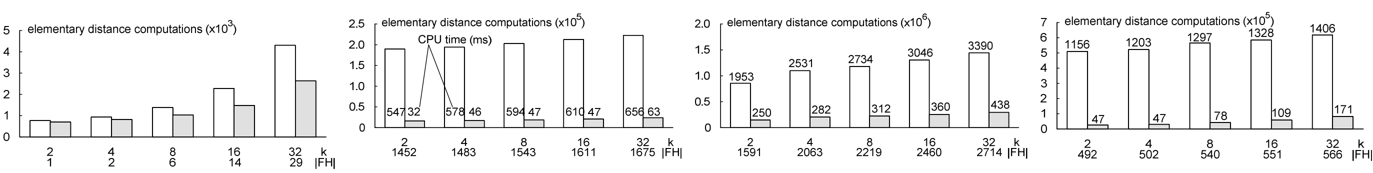
(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 18 Server computations vs. k (d=8)



(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 19 Client computations vs. d (k=8)

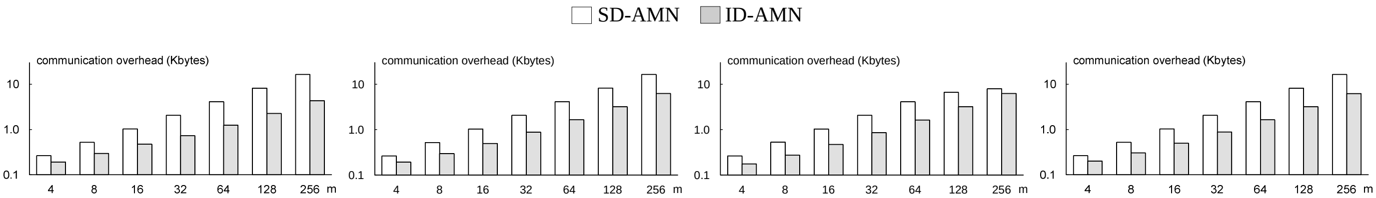


(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 20 Client computations vs. k (d=8)

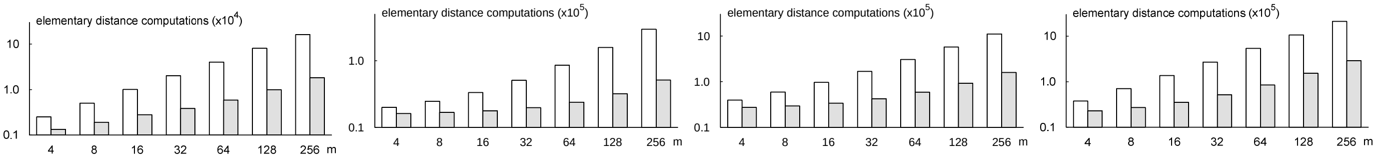
Figure 17 investigates the elementary distance computations at the server as a function of d. C-AMN is always more expensive than AMN due to the additional cost of *ReduceFH*. This cost drops with increasing d because there are fewer false hits to be reduced. The numbers inside each column denote the CPU time (in milliseconds) using a Pentium Intel Core 2 Duo 2.33GHz processor. For *Corel*, the CPU time is too low to be accurately measured. However, for *Chlorine*, *Stock* and *Mallat* it may reach several seconds due to the large values of D, and $|FH|$.

Figure 18 shows the server computations versus the number of required neighbors. The cost increases slightly with k, but similar to Figure 16, the effect is not as pronounced as that of d. Note that the diagrams do not



(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 21 Communication overhead vs. m (d=8, k=8)



(a) *Corel* (D=64, N=68040) (b) *Chlorine* (D=128, N=4310) (c) *Stock* (D=512, N=10000) (d) *Mallat* (D=1024, N=2400)

Fig. 22 Server computations vs. m (d=8, k=8)

include the I/O cost, which is identical to both methods. I/O operations normally dominate the processing overhead (since large records must be retrieved from the disk) and the performance difference of the two methods in terms of the overall cost diminishes. Moreover, the difference of C-AMN and AMN would decrease further (in Figures 17-18), if *DST* were based on a more expensive distance function than *dst* (e.g., DTW vs. Euclidean distance as in [10]) and applied the optimization of Section 4.3. This is because *ReduceFH* would entail only cheap *dst* computations, which would be dominated by the more expensive *DST* calculations, common in both methods.

Figure 19 illustrates the number of elementary distance computations at the client as a function of d. C-AMN leads to significant gains, sometimes dropping the processing cost by more than an order of magnitude. Since this cost is proportional to the amount of data received by the client, the diagrams are co-related to those in Figure 15; accordingly, the benefits of C-AMN are more significant for settings that involve large values of D, and $|FH|$. Figure 20 investigates the effect of k on the client. Similar to Figures 16 and 18, the CPU cost increases with k, but the impact of k is rather small.

Summarizing, compared to AMN, C-AMN imposes additional CPU cost for the server, in order to reduce the communication overhead and the verification effort at the client. This is a desirable trade-off in client-server architectures because (i) data transmissions constitute the main bottleneck in most applications, especially those involving wireless networks, and (ii) clients are assumed to have limited resources, whereas servers are powerful. Finally, note that the transmission overhead can also be reduced by conventional compression techniques. We do not include experiments with this approach since it benefits both AMN and C-AMN. Moreover, it increases the computational burden of the client, which has to decompress the data before their verification.

6.2 Distributed Servers

Next, we compare SD-AMN and ID-AMN considering that the database is horizontally partitioned over m servers. Recall that the methods first collect distance information, based on which they determine the range that contains the result. The NNs and the false hits are obtained during the verification of this range, which is identical in SD-AMN and ID-AMN. Thus, when measuring the communication cost, we focus on their differences, which regard the transmission of query points and the distance information. The CPU overhead is based again on elementary distance computations. Finally, due to the identical verification process, the client cost is similar, and the corresponding experiments are omitted.

Figure 21 shows the communication cost as a function of the number m of servers. Since we do not count the common data transmissions, the dominant factor is the number of high-dimensional query (Q) transmissions. SD-AMN sends Q to all servers yielding an overhead of $D \cdot m$ values. On the other hand, ID-AMN transmits Q only to candidate servers. In the best case, all results may be found in a single server, and the rest are eliminated using the *dst* bound; in the worst case, Q must be sent to all servers, if they all constitute false candidates. In general, the number of eliminated servers increases with their total number, leading to the savings of ID-AMN.

Figure 22 compares the two methods on elementary distance computations at the server versus m. The retrieval of a kNN set involves a number of computations linear to $(k+|FH|)\cdot(d+D)$ because the distances of all results and false hits must be evaluated in both low and high-dimensional spaces. In SD-AMN, each of the m servers must retrieve the k NNs; thus, the total cost increases linearly with both m and k. In ID-AMN a server has to perform a number of computations that is proportional to its contribution k^i ($\leq k$) in the result set. The value of m affects the number of computations only indirectly, by increasing the false candidates. In general, ID-AMN clearly outperforms SD-AMN in all settings.

7 CONCLUSIONS

The importance of authenticated query processing increases with the amount of information available at sources that are untrustworthy, unreliable, or simply unfamiliar. This is the first work addressing authenticated similarity retrieval from such sources using the multi-step *k*NN framework. We show that a direct integration of optimal NN search with an authenticated data structure incurs excessive communication overhead. Thus, we develop C-AMN, a technique that addresses the communication-specific aspects of NN, and minimizes the transmission overhead and verification effort of the clients. Furthermore, we propose ID-AMN, which retrieves distance information from distributed servers, eliminating those that cannot contribute results.

ACKNOWLEDGEMENTS

This work was supported by grant HKUST 618108 from Hong Kong RGC.

REFERENCES

[1] Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. *SIGMOD*, 1990.

[2] Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U. When is "Nearest Neighbor" Meaningful? *ICDT*, 1999.

[3] Bryan R. The Digital rEvolution: the Millennial Change in Medical Imaging. *Radiology*, November 2003.

[4] Cai, Y., Ng, R. Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials. *SIGMOD*, 2004.

[5] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *PVLDB*, 2008.

[6] Faloutsos, C., Ranganathan, M., Manolopoulos, Y. Fast Subsequence Matching in Time-Series databases. *SIGMOD*, 1994.

[7] Hjaltason, G., Samet, H. Distance Browsing in Spatial Databases. *ACM TODS*, 24(2), pp.265-318, 1999.

[8] Jagadish, H., Ooi, B., Tan, K., Yu, C., Zhang, R. i-Distance: an Adaptive B+-tree Based Indexing Method Nearest Neighbor Search. *ACM TODS*, 30(2), pp.364-397, 2005.

[9] Kellerer, H., Pferschy, U., Pisinger, D. *Knapsack Problems*. Springer, 2004.

[10] Keogh, E., J., Ratanamahatana, C., A. Exact Indexing of Dynamic Time Warping. *Knowl. Inf. Syst.*, 7(3), 2005.

[11] Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z. Fast Nearest Neighbor Search in Medical Image Databases. *VLDB*, 1996.

[12] Kundu, A., Bertino, E. Structural Signatures for Tree Data Structures. *PVLDB*, 2008.

[13] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. Dynamic Authenticated Index Structures for Outsourced Databases. *SIGMOD*, 2006.

[14] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S. A General Model for Authenticated Data Structures. *Algorithmica*, 39(1): 21-41, 2004.

[15] Merkle, R. A Certified Digital Signature. *CRYPTO*, 1989.

[16] Menezes, A., van Oorschot, P., Vanstone, S. *Handbook of Applied Cryptography*. CRC Press, 1996.

[17] Pang, H., Mouratidis, K. Authenticating the Query Results of Text Search Engines. *PVLDB*, 2008.

[18] Papadias, D., Zhang, J., Mamoulis, N., Tao, Y. Query Processing in Spatial Network Databases. *VLDB*, 2003.

[19] Seidl, T., Kriegel, H.-P. Optimal Multi-step *k*-Nearest Neighbor Search. *SIGMOD*, 1998.

[20] Snodgrass, R., T., Yao, S., S., Collberg, C. Tamper Detection in Audit Logs. *VLDB*, 2004.

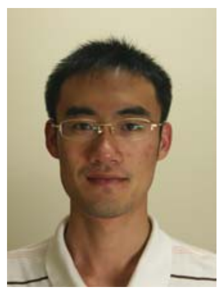
[21] Tamassia, R., Triandopoulos, N. Efficient Content Authentication over Distributed Hash Tables. *ACNS*, 2007.

[22] Tao, Y., Yi, K., Sheng, C., Kalnis, P. Quality and Efficiency in High Dimensional Nearest Neighbor Search. *SIGMOD*, 2009.

[23] Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G. Spatial Outsourcing for Location-based Services. *ICDE*, 2008.

[24] Yi, K., Li, F., Hadjieleftheriou, M., Kollios, G., Srivastava, D. Randomized Synopses for Query Assurance on Data Streams. *ICDE*, 2008.

Stavros Papadopoulos received his BSc degree in Computer Science from the Aristotle University of Thessaloniki, Greece. Currently, he is a PhD candidate at the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include authenticated query processing, private information retrieval, spatio-temporal databases and cloud computing.



Lixing Wang received a BE degree in software engineering and a BS degree in computer science from Fudan University, China and the University College Dublin, Ireland, respectively. Currently, he is a PhD student at the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include spatio-temporal databases and query processing in wireless sensor networks.

Yin Yang received a BE degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, and a PhD in Computer Science from the Hong Kong University of Science and Technology. He is currently a Visiting Scholar at the Department of Computer Science and Engineering, HKUST. His research interests include database outsourcing, query processing in data stream management systems, keyword search, and spatio-temporal databases.

Dimitris Papadias is a Professor of Computer Science and Engineering, Hong Kong University of Science and Technology. Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of

Athens, Queen's University (Canada), and University of Patras (Greece). He serves or has served in the editorial boards of the VLDB Journal, IEEE Transactions on Knowledge and Data Engineering, and Information Systems.

Panagiotis Karras is a Lee Kuan Yew Postdoctoral Fellow at the National University of Singapore. He earned a Ph.D. in Computer Science from the University of Hong Kong and an M.Eng. in Electrical and Computer Engineering from the National Technical University of Athens. In 2008, he received the Hong Kong Young Scientist Award. He has also held positions at the University of Zurich and the Technical University of Denmark. His research interests are in data mining, algorithms, spatial data management, anonymization, indexing, and similarity search. His work has been published in major database and data mining conferences and journals.