Close Dominance Graph: An Efficient Framework for Answering Continuous Top-k Dominating Queries

Bagus Jati Santoso and Ge-Ming Chiu, *Member, IEEE Computer Society*

Abstract—There are two preference-based queries commonly used in database systems: (1) *top-k query* and (2) *skyline query*. By combining the ranking rule used in top-k query and the notion of dominance relationships utilized in the skyline query, a top-k dominating query emerges, providing a new perspective on data processing. This query returns the k records with the highest domination scores from the dataset. However, the processing of the top-k dominating query is complex when the dataset operates under a streaming model. With new data being continuously generated while stale data being removed from the database, a continuous top-k dominating query (cTKDQ) requires that updated results can be returned to users at any time. This work explores the cTKDQ problem and proposes a unique indexing structure, called a Close Dominance Graph (CDG), to support the processing of a cTKDQ. The CDG provides comprehensive information regarding the dominance relationship between records, which is vital in answering a cTKDQ with a limited search space. The update process for a cTKDQ is then converted to a simple update affecting a small portion of the CDG. Experimental results show that this scheme is able to offer much better performance when compared with existing solutions.

Index Terms—Anchor set, close dominance graph, continuous query, dominance relationship, streaming model, top-k dominating query.

1 Introduction

In the past decade, the field of data management has witnessed strong demands for preference-based query processing. The two most commonly used techniques in preference-based queries are the *top-k query* [1], [2] and the *skyline query* [3]. A top-*k* query employs some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to rank the records in a data set $\mathbb{R}^d$ with d attributes. The query returns the subset of records of $\mathbb{R}^d$ that yield the k highest scores. The advantage of this popular technique is that the user is able to obtain satisfactory information, both in terms of the number of results and the correspondence with the specified preferences. However, defining an appropriate ranking function is difficult in many applications.

In contrast, skyline queries do not require a ranking function, and do not yield a fixed number of results. The result of a skyline query is a set of records that are not dominated by other records. The dominance relationship depends on the semantics of each attribute. A record p is said to dominate another record q if and only if p is not worse than q in all dimensions and is better than q in at least one dimension. Unlike the results of top-*k* queries that change according to the different preference function, the result of skyline queries are fixed for the same record set. The main advantage of a skyline query is that the user does not need to specify a ranking function. In addition, a skyline query gives users the ability to find a potentially important object due to its superiority in one or more attributes. The drawback of a skyline query is that the size of the query result is not bounded. For a totally anti-correlated record set [3], a skyline query yields the entire record set as the result. However, for a fully correlated record set [3], a skyline query returns exactly one record, which is not informative for most users.

Recently, another type of query, known as a top-k dominating query (TKDQ), was proposed in an attempt to provide a new perspective on data processing [4], [5]. TKDQ combines the ranking rule used in the top-*k* query and the notion of dominance relationship utilized in the skyline query. A score is defined for each record r, denoted by $score(r)$, which is the number of records dominated by r. It is also called the *domination score* of r, and it essentially represents the superiority of r among all records in the dataset. The query returns a subset of k records that have the highest scores. TKDQ is unique in that it provides a controlled number of results that offer high dominance power, which can be useful in many application domains.

Recent development of information and communication technology has resulted in improved infrastructure that, when coupled with increased bandwidth, has led many applications to rely on streaming computation models. In a streaming model, new data are

• *Bagus Jati Santoso and Ge-Ming Chiu are with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology.*
E-mail: {d9915804, chiu}@mail.ntust.edu.tw.

continuously generated while stale data are removed from the dataset. Many fields, such as stock market monitoring, early warning systems, network monitoring, and publish/subscribe systems, often adopt this computation model. Such a dynamic environment entails the need for a query processing technique that continuously provides updated query results to users. Adopting the streaming model in query processing is complicated by the fact that the data set is constantly changing, which generally increases the difficulty of designing a data structure to support the processing task.

In recent literature, the problem of TKDQ has been studied under the streaming computation model [6], [7]. Such a query is typically called a continuous top-k dominating query (cTKDQ). The cTKDQ always returns a set of k records with the highest domination scores under the latest state of the dataset. The existing solutions to cTKDQ are either based on a grid indexing structure [6] or on event-driven approaches [7]. Due to the deficiency of the data structures used in these schemes, there is significant room for improvement.

Motivated by the above observation, we re-examine the cTKDQ problem and proposes a unique indexing structure, a Close Dominance Graph (CDG), to support the processing of the query. Unlike the dominant graph (DG) [8], which is constructed to deal with top-k queries, a CDG is tailored to serve as a framework for tackling the cTKDQ problem. The CDG provides comprehensive information regarding the dominance relationship between records, which is vital in answering a cTKDQ with a limited search space. Based on the concept of the CDG, we then propose the notion of an $ANCHOR$ set, a small set of records that can be used to facilitate the processing of each update to the database. Useful properties associated with the $ANCHOR$ set are derived. Essentially, the update process for a cTKDQ is converted to a simple update to the $ANCHOR$ set. Experimental results show that our scheme is able to offer much better performance when compared with existing solutions.

In summary, this paper makes the following contributions:

1) We propose the fundamental structure of CDG to organize the dominance relationship between records so that a cTKDQ can be answered by simple graph traversal.
2) We define the notion of $ANCHOR$ set, which serves to significantly simplify the update process to the database in the streaming model.
3) We derive useful properties associated with the CDG and the $ANCHOR$ set. Based on these properties, we design an efficient algorithm, called $ANCHOR$*-Based Algorithm* (ABA), to process cTKDQ upon each update to the database.
4) The performance of the algorithm is evaluated by both analysis and extensive experiments using synthetic and real-world datasets.
5) We discuss an extension to deal with alternative streaming models.

The rest of the paper is organized as follows. In Section 2, we review related work. In Section 3, we describe the background of cTKDQ and the system model used in the paper. Section 4 presents the proposed data structures and derives the associated properties along with the ABA algorithm. Performance analysis and experimental results are presented in Section 5. In Section 6, we discuss an extension of our method to support alternative streaming models. Finally, Section 7 provides concluding remarks.

2 Related Work

First introduced in the work of [1], [2], top-k query has been one of the most popular preference data retrieval schemes today. Basically, top-k queries use a function to define a score for each record in the dataset. The records are then sorted according to these scores and the top k records are returned to the users. Top-k query has found application in many fields such as information retrieval [9], peer-to-peer system [10], network and system monitoring [11], data stream systems [12], and sensor networks [13], [14], [15], [16]. A comprehensive survey about top-k query in database can be found in [17].

In [8], a data structure called Pareto-Based Dominant Graph (commonly known as DG) was proposed in consideration of the intrinsic connection between top-k problem with aggregate monotone functions and dominance relationship. As a result, DG can be used to reduce the search space in order to answer a top-k query. In [18], a threshold-based technique was proposed for processing a top-k query in a highly distributed environment. The aim of the technique is to minimize the amount of data transferred between nodes. In [19], the authors proposed the reverse top-k query which addresses the problem of determining the weighting vectors in which a given record is on the top k list of a dataset. Related research can also be found in [20].

Pioneered by the work of [3], skyline query provides users with a set of data that are prominent based on the concept of dominance relationship. Extensive research results have since been reported in the literature. In [21], two algorithms, called Bitmap and Index, were proposed to progressively provide skyline points. The authors of [22] proposed a nearest neighbor search-based technique to tackle the problem without having to read the entire dataset. Instead a of centralized system environment, the processing of a skyline query has also been adapted to peer-to-peer systems [23], uncertain databases [24], mobile environment [25], distributed environment [26], and over sliding windows [27]. As an alternative, [28] proposed the concept of representative skyline that

returns k skyline points that best describe trade-offs among different dimensions offered by the full skyline. Furthermore, [29] presented a technique to answer a skyline query by using the Z-order curve approach.

TKDQ is a relatively new topic in the field of data management. First introduced in [4], this query offers a different perspective to users by providing k records of the dataset with the highest domination scores. The authors of [5] further analyzed the TKDQ problem. In [30], several algorithms that are based on the aggregate R-tree were proposed to process the query. In this scheme, a pruning technique was used to reduce the search space needed to obtain the query result. In [31], the problem of probabilistic top-k dominating query was introduced to address uncertain datasets.

The introduction of streaming models presents new challenges for the processing of data queries, in which updated query results must be returned continuously. In [32], the authors provided a review of some important preference queries over data streams. The work of [27], [33], [34] addressed the problem of processing skyline query over dynamic datasets. Works that addressed other types of query over streaming model can also be found in [35], [36].

In [6], Kontaki et al. proposed a grid-based indexing scheme to answer a cTKDQ in subspace. The proposed method returns k records with highest domination scores over dynamic dataset with respect to any subset of available attributes. Assisted by the grid structure, the computation of a domination score is simplified because intensive checking is needed only for records existing in partially dominated cells. Later, the authors proposed an event-based approach to deal with the problem in [7].

In this approach, an event is associated with each record at any time, which captures the closest possible instance for a record to be part of the query result. This approach attempts to reduce the need for costly computation of the exact domination score for a record by processing the event associated with the record only when the event instance expires. The event of a record is essentially established on the assumption that the worst-case scenario happens at each update for the record. In an update, the worst-case scenario occurs if the domination score of the k-th ranked record decreases by one and the score of the record of interest increases by one in the meanwhile. However, if worst-case scenarios do not happen as frequently as assumed, the processing of an event becomes premature, and thus the computation effort induced will be totally wasted. Although an enhanced algorithm with additional optimization techniques were proposed in the paper, reducing ineffectual event-processing effort remains a challenge.

3 Background and System Model

In this section, we formally define the continuous top-k dominating query (cTKDQ) and describe the system model used in this paper. The cTKDQ query stems from TKDQ which is governed by the notion of dominance.

Definition 1 (Dominance). *Let $r_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,d}\}$ represent a record of d dimensions with $r_{i,j}$ denoting the attribute in dimension j, $1 \leq j \leq d$. A record r_x dominates another record r_y, denoted by $r_x \prec r_y$, if and only if the following two conditions are satisfied: 1) $r_{x,j} \leq r_{y,j}$ for all j, $1 \leq j \leq d$, 2) there exists at least one dimension $1 \leq k \leq d$ such that $r_{x,k} < r_{y,k}$.*

Given a set R of records in multidimensional space, for any record $r \in R$, we use $score(r)$ to represent the domination score of r, which is the number of records in R that are dominated by r. A TKDQ query applied to R returns a subset of k records in R that have the highest domination scores.

Consider the scenario in which a stream of records arriving continuously to join the system and stale records continuously to leave the system. A cTKDQ is to monitor the state of the system continuously and always returns updated TKDQ results to users. The aim of this paper is to design a time-efficient algorithm to achieve this goal.

For continuous data streams, there are two basic sliding window types: *count-based* and *time-based*. In a count-based sliding window, the number of active records in the dataset remains constant; the arrival of a new record causes the expiration of the oldest record. Meanwhile, in a time-based sliding window, the expiration time of a record does not depend on the arrival or expiration of other records in the dataset. Rather, the set of active records in the dataset is composed of all records that arrived in the last T time instances; thus the number of active records in the dataset may not be constant. We adopt the count-based sliding window in this paper. However, an extension of our scheme to deal with a time-based sliding window will be discussed in a latter section. With the count-based sliding window, we are interested in the process of each update to the database. Hence, we may treat time as proceeding in a discrete manner, where each update event carries an integer-valued timestamp that is incremented by one at each update.

To facilitate our discussion, we denote the query result of a top-k dominating query by $TOPK$ which contains the k records with the highest domination scores. The smallest domination score in $TOPK$, i.e. the domination score of the k-th record in $TOPK$, is denoted by $score_k$. Since the count-based sliding window is used, the total number of records in the database is kept constant at any time. We also denote the incoming and expiring records for each update event by r_{inc} and r_{exp}, respectively. Table 1 summa-

TABLE 1: Basic Symbols

Symbol	Interpretation
n	number of active records
d	number of dimensions
r_i	the i-th record, in which $i = 1, \ldots, n$
r_{inc}	the new incoming record
r_{exp}	the expired record
$score(r)$	the number of records that dominated by r
$TOPK$	the set of k records with the highest domination scores
$CAND$	the set of candidate records of cTKDQ result
$ANCHOR$	the subset of candidate records that do not dominate any candidate record
$score_k$	the score of k-th records in $TOPK$
$p \prec q$	record p dominates q
$p \rightarrow q$	record p closely dominates q in CDG
$P(r)$	the set of records p_i, in which $i = 1, 2, \ldots$ and $p_i \rightarrow r \in$ CDG
$C(r)$	the set of records c_i, in which $i = 1, 2, \ldots$ and $r \rightarrow c_i \in$ CDG
$A(r)$	the set of records a_i, in which $i = 1, 2, \ldots$ and $a_i \prec r$
$D(r)$	the set of records d_i, in which $i = 1, 2, \ldots$ and $r \prec d_i$
$\overline{sky}(R)$	the skyline query result of set R

rizes the basic symbols used in this paper.

4 The Proposed Scheme for Processing cTKDQ

To answer a cTKDQ, we propose a graph-based data structure that maintains dominance relations identified over a set of records. Utilizing the unique properties of the structure, we can greatly reduce the search space when updating the answer to the query.

Definition 2 (Close Dominance Relation). *Consider a set R of records in a multidimensional space. Let $A(x)$ represents the set of records in R that dominate record $x \in R$, i.e. $A(x) = \{y | y \in R, y \prec x\}$. A record $r \in R$ is said to be **closely dominated** by another record $s \in R$, if and only if $s \in A(r)$ and s does not dominate any other record $t \in A(r)$. This relationship is denoted by $s \rightarrow r$.*

In the following discussion, we use $A(r)$ and $P(r)$ to denote the set of records that dominate and closely dominate a record r, respectively. In addition, we use $D(r)$ and $C(r)$ to represent the set of records that are dominated and closely dominated by r, respectively. Obviously, $P(r) \subseteq A(r)$ and $C(r) \subseteq D(r)$.

Definition 3 (Close Dominance Graph). *Consider a set R of records in a multidimensional space. The close dominance graph (or CDG) of R is a directed graph, in which R is the vertex set and any two vertices s and r are connected by a directed link from s to r, if and only if $s \rightarrow r$ exists.*

If $y \prec x$ exists, then there is at least one directed path from y to x. Clearly, a CDG is acyclic.

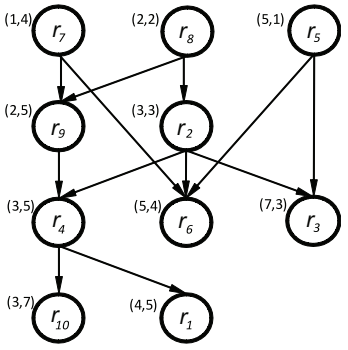


Fig. 1: An example of a CDG for a set of ten records.

Fig. 1 illustrates a CDG for ten records of two-dimensional space, where r_4 is dominated by $A(r_4) = \{r_2, r_7, r_8, r_9\}$. Since r_2 and r_9 do not dominate any record in $A(r_4)$, the relations $r_2 \rightarrow r_4$ and $r_9 \rightarrow r_4$ exist in the CDG. Record r_8 dominates all other records except r_5 and r_7. In the figure, there exist two paths from r_8 to r_4, although r_8 does not closely dominate r_4.

Since all records that are dominated by r become descendants of r in the CDG, we can compute $score(r)$ by simply counting the number of descendants of r in the CDG by downward traversal.

4.1 Candidate Records for The Query Result

As stated earlier, we assume a count-based sliding window model in this paper, i.e., each insertion of a new record triggers the removal of the oldest remaining record. The dynamic nature of the database makes the $TOPK$ result susceptible to change during each update process. A naive approach to finding $TOPK$ upon each update is to compute the latest domination scores for the records by performing domination checks among them. Such an approach is computationally expensive, however. In this paper, we propose the use of a CDG graph to support the update process and obtain the new query result.

Consider a set of records R, which is being updated with a new incoming record r_{inc} while an existing record r_{exp} is being expelled from the system. The new set of records is represented by S, i.e., $S = (R - \{r_{exp}\}) \bigcup \{r_{inc}\}$. Note that $score(r)$ can be increased or decreased by at most one in each update. To distinguish the difference between the system state before and after an update, we use $score_k^B$, $score^B(r)$, and $TOPK^B$ to represent $score_k$, $score(r)$, and the query result of a cTKDQ Q over a set B of records, respectively. Finding a set of candidate records from S in order to answer Q after the update occurs is governed by the following theorem.

Theorem 1. *Suppose that an update changes the system state from R to S with $S = (R - \{r_{exp}\}) \bigcup \{r_{inc}\}$. For a cTKDQ Q, a record $r \in S$ cannot belong to $TOPK^S$ if the following two conditions are both met: (1) $score^S(r) <$*

$score_k^R - 2$ and (2) there exists some $z \in S$ such that $z \rightarrow r$ exists in the updated CDG and $score^S(z) < score_k^R - 1$.

Proof. There are at least $k-1$ records in $TOPK^R$ that will remain in S and have domination scores greater than or equal to $score_k^R - 1$. Thus, each of those records has a domination score greater than $score^S(r)$ due to condition (1). Since we have $score(z) < score_k^R - 1$, we know that z cannot belong to $TOPK^R$, which means that it cannot possibly be a member of the above-mentioned $k-1$ records. Since z dominates r, $score^S(r) < score^S(z)$. Therefore, there are at least k records in S whose domination scores are greater than $score^S(r)$. Thus, r cannot possibly be included in $TOPK^S$. □

Theorem 1 confines the set of records that can be included in the updated query result $TOPK^S$. For example, suppose that Fig. 1 illustrates the system state at the preceding time instance, which is redrawn in Fig. 2(a), and a top-3 dominating query is applied to the system. The domination scores of the records are listed in Fig. 2(a), where the query result consists of r_8, r_2, and r_7 (dark-shaded in the figure). The value of $score_k$ at this time is $score(r_7) = 5$. Suppose that a new update occurs with a new record r_{11} replacing the oldest, namely r_1 (as in Fig. 2(b)). Since $score_k^R = 5 > score^S(r_4) + 2 = 3$, and we have $r_9 \rightarrow r_4$ with $score^S(r_9) < score_k^R - 1$, r_4 cannot possibly become part of the updated query result at the current time, as per Theorem 1. Similarly, r_3, r_6, and r_{10} cannot be included in the new query result either. Moreover, since $score(r_{11}) = 1$ is less than $score_k$ by more than two and $r_5 \rightarrow r_{11}$ exists, the new record r_{11} cannot be part of the updated result. This would then leave us with five records, namely, r_2, r_5, r_7, r_8, and r_9, to consider for the updated query result.

4.2 *ANCHOR* and Updating *TOPK*

Theorem 1 allows us to exclude a large proportion of records in the new record set S from being considered for the updated query result. This significantly reduces the search space for finding the new solution. We use $CAND^S$ to represent the subset of records of S that do not meet Theorem 1, i.e., the candidate records that may possibly be part of $TOPK^S$. For any record $r \in TOPK^R$, where $r \neq r_{exp}$, r must be included in $CAND^S$. This is because r's domination score can decrease by at most one after the update is processed. In other words, all records in $TOPK^R$ that remain in set S shall be treated as candidates for answering the query. In the example of Fig. 2(b), we have $CAND^S = \{r_2, r_5, r_7, r_8, r_9\}$.

Although we can search $CAND^S$ to find the new query result for a given cTKDQ in the presence of an update, it is computationally expensive to calculate $CAND^S$. Instead of explicitly calculating $CAND^S$, our scheme exploits the property that if $r \rightarrow r'$ with $r' \in CAND^S$, then $r \in CAND^S$. Essentially, we convert the problem of searching the query result into a task of traversing the CDG. To facilitate the traversal scheme, we propose a special subset of $CAND^S$ to help answer the query. Our scheme maintains only this subset of $CAND^S$, which is called the $ANCHOR$ set and is defined as follows:

Definition 4 (***ANCHOR***). *The $ANCHOR$ set with respect to S, denoted by $ANCHOR^S$, is the subset of records of $CAND^S$ that do not dominate any other record in $CAND^S$.*

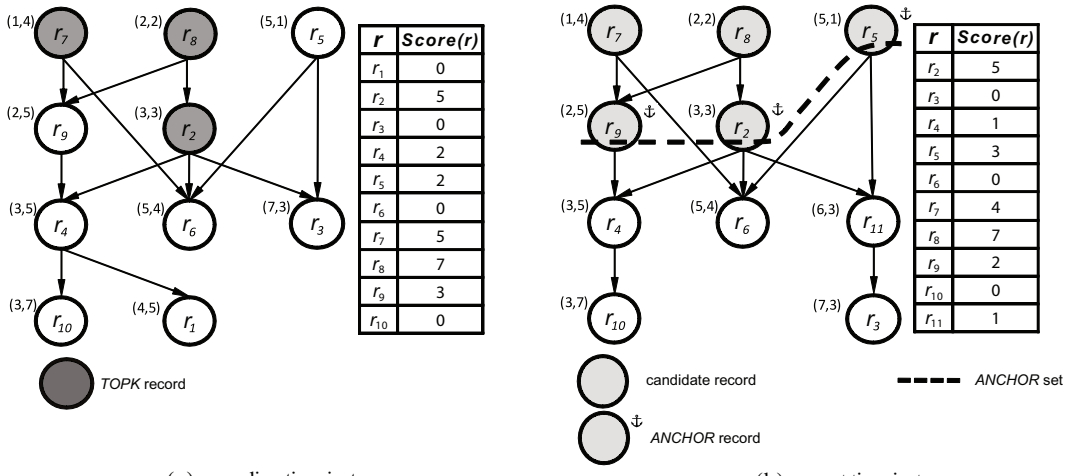
For example, in the system of Fig. 2(b), we have $ANCHOR^S = \{r_2, r_5, r_9\}$ (denoted by the dashed line in the figure) with $CAND^S = \{r_2, r_5, r_7, r_8, r_9\}$. Clearly, $r \in CAND^S$ and $r \notin ANCHOR^S$ if and only if r dominates at least one record $r' \in ANCHOR^S$. In other words, any record in $CAND^S$ is either in $ANCHOR^S$ or dominates some record in $ANCHOR^S$. Accordingly, any record that is located above $ANCHOR^S$ in the CDG always belongs to $CAND^S$. Moreover, we need to make sure that any record $r \notin CAND^S$ is dominated by at least one record $r' \in ANCHOR^S$. This property is crucial for our approach and is given by the following theorem.

Theorem 2. *$r \notin CAND^S$ if and only if r is dominated by at least one record $r' \in ANCHOR^S$.*

Proof. If some record $r' \in ANCHOR^S$ dominates r, then it is obvious that r cannot belong to $CAND^S$. Consider the case that $r \notin CAND^S$. In this case, there exists some record x such that $x \rightarrow r$ and $score^S(x) < score_k^R - 1$. If $x \notin CAND^S$, then x has to have some parent node in the CDG, say y, such that $y \rightarrow x$ and $score^S(y) < score_k^R - 1$. Of course, y is an ancestor record of r. Since the CDG is acyclic, if we continue the above process, we must eventually visit a node z, $z \in A(r)$, such that either $P(z) = \varnothing$ or all $p \in P(z)$ have $score^S(p) \geq score_k^R - 1$. In either case, we have $z \in CAND^S$. Note that $score^S(z) < score_k^R - 1$. We now show that z cannot dominate any other record in $CAND^S$. For proof by contradiction, assume that z indeed dominates some record $w \in CAND^S$. We can easily see that there exists some record $q \in CAND^S$ such that $z \prec q$ exists. Since $score^S(q) < score^S(z)$, we can deduce that $score^S(q) < score_k^R - 2$. Considering that $score^S(z) < score_k^R - 1$, it must be true that $q \notin CAND^S$, which is a contradiction. Therefore, z does not dominate any record in $CAND^S$. Thus, $z \in ANCHOR^S$ and z dominates r.

□

Based on the above discussion, we know that a record $r \notin ANCHOR^S$ must either dominate or be dominated by one or more $r' \in ANCHOR^S$. In fact, $ANCHOR^S$ can be regarded as forming the bottom of the $CAND^S$ and it partitions the rest of the data



(a) preceding time instance (b) current time instance

Fig. 2: An example of a system update.

set. Since $CAND^S$ is above and includes $ANCHOR^S$ in the CDG, we can obtain $TOPK^S$ by traversing the CDG upward starting from the $ANCHOR^S$. Furthermore, the new $ANCHOR$ set can be readily derived from the previous one, i.e., $ANCHOR^S$ can be obtained by a simple update to $ANCHOR^R$. In essence, the update process starts from $ANCHOR^R$ and traverses the CDG upward or downward to determine the elements in $ANCHOR^S$. Details of the algorithm are given in a later section. Theorems 3 and 4, given below, provide useful bounds for such traversals. The following lemma is used in the proof of Theorem 4.

Lemma 1. *In any system state R, suppose that there exists a path of length m from r to some $r' \in ANCHOR^R$ in the CDG of R. If $m > 3$, then $score^R(r) \geq score_k^R + (m-4)$.*

Proof. Denote such path by $r \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots r_{m-1} \rightarrow r'$ with $m > 3$. To facilitate our proof, let us denote the previous system state by Q. That is, system changes from state Q to R. Since $r' \in ANCHOR^R$, then $score^R(r_{m-1}) \geq score_k^Q - 1$ follows. Consider the record r_{m-3}. We have $score^R(r_{m-3}) > score_k^Q$. Two cases are considered below :
(a) $m = 4$:
In this case, we obtain that $score^R(r) > score_k^Q + 1$. Note that there are no more than $k-1$ records in Q whose domination scores are greater than $score_k^Q$, while the remaining records in Q necessarily have domination scores that are less than or equal to $score_k^Q$. By considering that the domination score of any record in Q can increase by at most one when the system switches to state R, we know that there are no more than $k-1$ records in Q whose domination scores can be greater than $score_k^Q + 1$ in state R. Hence, it must hold that $r \in TOPK^R$ due to the fact that $score^R(r) > score_k^Q + 1$, which implies that $score^R(r) \geq score_k^R = score_k^R + (m-4)$.
(b) $m > 4$:
In this case, we have $score^R(r_{m-4}) > score_k^Q + 1$ with $r_{m-4} \neq r$. By a similar argument, we know that $r_{m-4} \in TOPK^R$, which implies that $score^R(r_{m-4}) \geq score_k^R$. It follows that $score^R(r) \geq score_k^R + m - 4$. □

Theorem 3. *Suppose that an update changes the system from R to S, with $S = (R - \{r_{exp}\}) \bigcup \{r_{inc}\}$. For a record $r \in R$ with $r \notin CAND^R$, if there exists a path from some record $r' \in ANCHOR^R$ to r in the CDG of S that has path length greater than three, then $r \notin ANCHOR^S$.*

Proof. Let the path be represented by $r' \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots r_m \rightarrow r$ with $m \geq 3$. Among the records r_i, for $1 \leq i \leq m$, at least $m-1$ of these must exist in R. Therefore, the dominance relations remain valid. Two cases are considered here :
(a) None of the r_i's is r_{inc} :
In this case, we have $r_i \in R$ for all $1 \leq i \leq m$. Obviously, we have $score^R(r_1) < score_k^R$. Furthermore, we can deduce that $score^R(r_m) < score_k^R - m + 1$ and $score^R(r) < score_k^R - m$. Since $m \geq 3$, we have $score^R(r_m) < score_k^R - 1$ and $score^R(r) < score_k^R - 2$, which implies that $r \notin CAND^S$ and, hence, $r \notin ANCHOR^S$.
(b) $r_{inc} = r_j$ for some $1 \leq j \leq m$:
In this case, we have all $r_i \in R$, except r_j. Two subcases are considered below.
(b.1) $j \neq m$:
Considering state R, we can deduce that $score^R(r_m) < score_k^R - m + 2$ and $score^R(r) < score_k^R - m + 1$. Given $m \geq 3$, we have $score^R(r_m) < score_k^R - 1$ and $score^R(r) < score_k^R - 2$. Furthermore, since $j < m$, we know that $score^S(r_m) \leq score^R(r_m)$ and $score^S(r) \leq score^R(r)$, which then yields $score^S(r_m) < score_k^R - 1$ and $score^S(r) < score_k^R - 2$. We conclude that $r \notin CAND^S$ and, hence, $r \notin ANCHOR^S$.
(b.2) $j = m$, so $r_{inc} = r_m$:
By a similar argument, we determine that

$score^R(r_{m-1}) < score_k^R - m + 2$ and $score^R(r) < score_k^R - m + 1$. Given that $m \geq 3$, we then have $score^R(r_{m-1}) < score_k^R - 1$ and $score^R(r) < score_k^R - 2$. Since $j = m$, it is clear that $score^S(r_{m-1}) \leq score^R(r_{m-1}) + 1$ and $score^S(r) \leq score^R(r)$, which then yields $score^S(r_{m-1}) < score_k^R$ and $score^S(r) < score_k^R - 2$. Since r_{m-1} dominates r_m, we have $score^S(r_m) < score_k^R - 1$. We then conclude that $r \notin CAND^S$ and, hence, $r \notin ANCHOR^S$.

□

Theorem 4. *Suppose that an update changes the system from R to S, with $S = (R - \{r_{exp}\}) \bigcup \{r_{inc}\}$. For a record $r \in R$ with $r \in CAND^R$, if there exists a path from r to some record $r' \in ANCHOR^R$ in the CDG of S that has path length greater than four, then $r \notin ANCHOR^S$.*

Proof. Let the path be represented by $r \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots r_n \rightarrow r'$ with $n \geq 4$. We have two cases :
(a) None of r_i's is r_{inc} :
In this case, all $r_i \in R$ and thus, by Lemma 1, we have $score^R(r_1) \geq score_k^R + n - 4$. As $n \geq 4$, it holds that $score^R(r_1) \geq score_k^R$. Therefore, we can deduce that $score^S(r_1) \geq score_k^R - 1$. This implies that $r_1 \in CAND^S$. Hence $r \notin ANCHOR^S$ as it dominates r_1.
(b) $r_j = r_{inc}$ for some $1 \leq j \leq n$:
In this case, all of r_i's, where $i \neq j$, are in R. According to Lemma 1, we have $score^R(r) \geq score_k^R + n - 4$. As $n \geq 4$, it follows that $score^R(r) \geq score_k^R$. Since $r_{inc} = r_j$ is dominated by r in S, we can deduce that $score^S(r) \geq score_k^R$. Note that this result applies to all records in $P(r_1)$, including r. Therefore, we can conclude that $r_1 \in CAND^S$, and thus $r \notin ANCHOR^S$ because it dominates r_1 in the CDG of S. □

Theorems 3 and 4 confine the scope of records that need be examined when updating the $ANCHOR$ set. In reality, the actual number of records inspected is much less than these upper bounds.

4.3 Initial Construction of the CDG

When the system commences operation, we need to construct the CDG from scratch. In this initialization phase, the number of records has not yet achieved a full sliding window. Note that no record is deleted from the system until the sliding window is completely filled. Although this paper mainly focuses on regular update process in which record insertion and deletion occur simultaneously, we discuss the initial construction of CDG in this section for the purpose of completeness. There are two approaches for constructing a CDG at this stage: static and incremental. The static approach assumes that the query result is outputted to a cTDKQ only when the sliding window is full, while incremental methods offer query result even before the sliding window is filled. Since the incremental approach builds initial CDG in an incremental manner as data arrive in the system, its operation is similar to the regular update process. To facilitate our exposition, we describe the static method in this section and will defer the description of the incremental method to the following section, in which the regular update process is addressed.

Our static method assumes that new records are collected at the start of the system's operation until the sliding windows is full. It then calls Algorithm 1 to construct the initial CDG and the associated entities. In the algorithm, $INITIAL$ represents the set of records that are collected initially. The algorithm builds the CDG by repeatedly performing a sequence of two-step operations. In the first step, we obtain the skyline query result for a given set of records using skyline query algorithm [37]. Next, we insert the records of the query result into the CDG progressively. The skyline-insertion operation is repeated with respect to the remaining records. The algorithm ends when no record remains to be inserted into the CDG. This procedure is performed by the while loop at line 2, in which the close-dominance relations are identified by the FindInitial() function (line 8). The function traverses down the existing CDG starting from the records in SKY, which contains only the first skyline query result, to search for records that closely dominate a new record. FindInitial() is also used later to find the initial $ANCHOR$ set. The $type$ variable is used to distinguish these operations, with "C" identifying close-dominance relations and "A" obtaining the $ANCHOR$ set.

Note that this procedure ensures that records that are newly added to the CDG cannot possibly dominate any record already existing in the CDG, which simplifies the construction process. The flag $exact(r)$ is used to indicate whether $score(r)$ carries the exact domination score of record r. Our scheme does not require that all records maintain the exact domination scores at all times. The function ScoreUpdate() updates the domination score for a record r provided that $exact(r) = TRUE$. Since a record may be visited more than once during the traversal operation, we use $visited(r)$ as a flag to indicate whether r has been visited.

After the algorithm completes the construction of the CDG, it proceeds to obtain the initial TOPK and $score_k$, and computes the initial $ANCHOR$ set. As described earlier, we call the function FindInitial() to obtain the initial $ANCHOR$ set because such an operation shares similar acts with the process of identifying close-dominance relations described earlier. For this process, the function employs another function, IsInitialCand(), to determine whether a record belongs to the $CAND$ set. This $ANCHOR$ set serves as an input to the first regular update process.

Algorithm 1 Constructing Initial CDG

Input: $INITIAL$
Output: CDG, $TOPK$, $ANCHOR$
1: $SKY \leftarrow$ skyline query result of $INITIAL$
2: **while** $|\ INITIAL\ | > 0$ **do**
3: $LAYER \leftarrow$ skyline query result of $INITIAL$;
4: **for all** record $r \in LAYER$ **do**
5: remove r from $INITIAL$;
6: reset $visited(r')$, $\forall r' \in$ CDG;
7: **for all** record $s \in SKY$ **do**
8: **if** $s \prec r$ **then** FindInitial(s, "C", r);
9: set $exact(r) = TRUE$;
10: reset $visited(r')$, $\forall r' \in$ CDG;
11: ScoreUpdate(r, 1);
12: $TOPK \leftarrow$ the k records in CDG that have highest score;
13: $score_k \leftarrow$ score of k-th record in $TOPK$;
14: reset $visited(r')$, $\forall r' \in$ CDG;
15: **for all** record $s \in SKY$ **do** FindInitial(s, "A", $NULL$);
16: **return** CDG, $TOPK$, and $ANCHOR$;

17: **function** ScoreUpdate(r, $update$)
18: $visited(r') = TRUE$;
19: **for all** $r' \in P(r)$ **do**
20: **if** $visited(r') = FALSE$ **then**
21: **if** $exact(r')$ **then** $score(r')$ += $update$;
22: ScoreUpdate(r', $update$);

23: **function** IsInitialCand(r)
24: **if** $(score(r) \geq score_k - 2) \vee (P(r) = \varnothing) \vee$ $(score(r') \geq score_k - 1, \forall r' \in P(r))$ **then**
return $TRUE$;
25: **else return** $FALSE$;

26: **function** FindInitial(s, $type$, $pivot$)
27: $visited(s) = TRUE$;
28: $expanded = FALSE$;
29: **for all** record $s' \in C(s)$ **do**
30: **if** ($type$ = "C" $\wedge$ $s' \prec pivot$) $\vee$ ($type$ = "A" $\wedge$ IsInitialCand(s')) **then**
31: $expanded = TRUE$;
32: **if** $visited(s') = FALSE$ **then** FindInitial(s', $type$, $pivot$);
33: **if** ($type$="C") $\wedge$ ($expanded = FALSE$) **then**
34: insert relation $s \rightarrow pivot$ in CDG;
35: **else if** ($type$="A") $\wedge$ ($expanded = FALSE$) **then**
36: insert s to $ANCHOR$;

4.4 The *ANCHOR*-Based Algorithm

In this section, we present our main algorithm, called $ANCHOR$-Based Algorithm (ABA), which is used to process a regular update to the database. In addition to producing the updated query result for a given cTKDQ, ABA has to perform a bookkeeping task to ensure that subsequent updates can be correctly processed. The pseudocode for ABA is given in Algorithm 2.

Here we assume that the system changes from state R to state S with $S = (R - \{r_{exp}\}) \bigcup \{r_{inc}\}$. The input to ABA is $ANCHOR^R$ and $score_k^R$. The ABA can be divided into three parts which are carried out sequentially: updating the CDG, generating the $ANCHOR$ set, and computing the updated query result. The process of updating the CDG involves the removal of r_{exp} and the addition of r_{inc} to the existing CDG. When removing r_{exp} from the CDG (lines 2-9), we first decrease the scores for all records that dominate r_{exp}. Then we save the records in $P(r_{exp})$ and $C(r_{exp})$ into $TEMP$ and $TEMC$, respectively, and remove r_{exp}. In addition, we must reconstruct close-dominance relations around r_{exp} to ensure the correctness of the CDG. To this end, it may be necessary to add links between the records in $TEMP$ and those in $TEMC$ (line 5). This additional procedure commences by examining whether there is another path from a record in $TEMP$ to a record in $TEMC$. If no other path exists between them, a direct link (i.e. a close-dominance relation) is added to the CDG.

Moreover, if r_{exp} is included in $ANCHOR^R$, then we delete r_{exp} from $ANCHOR^R$ and check whether its children (or its parents) should be inserted into $ANCHOR^R$. We then insert all such records, if any, into $ANCHOR^R$ (lines 6-9). This is necessary to ensure that the forthcoming update to the $ANCHOR$ set is correctly performed. In the algorithm, we examine the records in $TEMC$ first. If any record in $TEMC$ is added to $ANCHOR^R$, then the algorithm is not required to check the records in $TEMP$. This operation is performed by the RelateAndInsert() function.

Next, ABA inserts the new record r_{inc} into the CDG (lines 10-15). In this step, we have to update close-dominance relations of the CDG to account for the addition of r_{inc}. Here we can use the *kd-tree* indexing scheme [38], [39] to help identify $P(r_{inc})$ and $C(r_{inc})$ in the updated CDG. The *kd-tree* is a well-known technique with outstanding performance for search operations that involve multidimensional spaces. Furthermore, we remove all links that previously existed between the records in $P(r_{inc})$ and $C(r_{inc})$. In addition, we update the scores of all records that dominate r_{inc} by calling the ScoreUpdate() function. Finally, to ensure the consistency of CDG, we need to account for the possibility that r_{inc} cannot be reached by traversing (upward or downward) the CDG from the records in $ANCHOR^R$ (line 15). To this end, if every record in $ANCHOR^R$ neither dominates r_{inc} nor is dominated by r_{inc}, we add r_{inc} to $ANCHOR^R$. This is done to facilitate subsequent update to the $ANCHOR$ set, even though $ANCHOR^R$ can never really contain r_{inc} in reality. One can readily see that Theorems 3 and 4 remain valid in this case.

Once the CDG update has been completed, ABA proceeds to compute $ANCHOR^S$, the new $ANCHOR$ set. In our scheme, $ANCHOR^S$ can simply be derived from $ANCHOR^R$. In this respect, Theorems 3 and 4 provide a useful insight. For this task, the algorithm uses the function UpdateAnchor(), in which we examine each record $r \in ANCHOR^R$ by first examining whether r is a member of $CAND^S$ using the function IsCandidate(). If r is part of $CAND^S$, we proceed to examine its descendant records using

Algorithm 2 $ANCHOR$-Based Algorithm

Input: $ANCHOR^R$, $score_k^R$
Output: $TOPK^S$, $ANCHOR^S$, $score_k^S$
1: reset $visited(r)$, $\forall r \in$ CDG;
2: ScoreUpdate($r_{exp}, -1$);
3: $TEMP \leftarrow P(r_{exp})$ and $TEMC \leftarrow C(r_{exp})$;
4: remove r_{exp} and its relations in CDG;
5: **if** $\nexists\, p \in TEMP$, $c \in TEMC$, s.t. path between p and c exists in CDG **then** build relation $p \rightarrow c$ in CDG;
6: **if** $r_{exp} \in ANCHOR^R$ **then**
7: remove r_{exp} from $ANCHOR^R$;
8: RelateAndInsert($TEMC$);
9: **if** $\nexists\, c \in TEMC$, s.t. $c \in ANCHOR^R$ **then** RelateAndInsert($TEMP$);
10: get $P(r_{inc})$ and $C(r_{inc})$ using *kd-tree* searching;
11: build relation ($p \rightarrow r_{inc}, \forall p \in P(r_{inc})$) and ($r_{inc} \rightarrow c, \forall c \in C(r_{inc})$);
12: remove all $p \rightarrow c$ in CDG, where $p \in P(r_{inc})$, $c \in C(r_{inc})$;
13: reset $visited(r), \forall r \in$ CDG;
14: ScoreUpdate(r_{inc}, 1);
15: RelateAndInsert($\{r_{inc}\}$);
16: UpdateAnchor();
17: reset $visited(r), \forall r \in$ CDG and set $score_k^S = 0$;
18: **for all** records $r \in ANCHOR^S$ **do** GetTOPK(r);
19: **return** $TOPK^S$, $ANCHOR^S$, $score_k^S$;

20: **function** RelateAndInsert(T)
21: **for all** records $r \in T$ **do**
22: **if** $\nexists\, r' \in ANCHOR^R$, s.t. $(r \prec r') \vee (r' \prec r)$ **then** insert r into $ANCHOR^R$;

23: **function** IsCandidate(r)
24: **if** $exact(r) =$ *FALSE* **then**
25: compute $score(r)$;
26: $exact(r) =$ *TRUE*;
27: **if** $(score(r) \geq score_k^R - 2) \vee (P(r) = \varnothing) \vee (\nexists\, r' \in P(r)$, s.t. $score(r') < score_k^R - 1)$ **then return** *TRUE*;
28: **return** *FALSE*;

29: **function** IsAnchor(r)
30: **if** $(C(r) = \varnothing) \vee (\nexists\, c \in C(r)$, s.t. IsCandidate(c)) **then return** *TRUE*;
31: **return** *FALSE*;

32: **function** InsertAnchor(r)
33: **if** $r \notin ANCHOR^S$ **then** insert r into $ANCHOR^S$;

34: **function** CheckChildrenAndInsert(r)
35: **for all** records $c \in C(r)$ **do**
36: **if** IsCandidate(c) **then**
37: **if** IsAnchor(c) **then** InsertAnchor(c);
38: **else** CheckChildrenAndInsert(c);
39: **if** IsAnchor(r) **then** InsertAnchor(r);

40: **function** CheckParentAndInsert(r)
41: **for all** records $p \in P(r)$ **do**
42: **if** IsCandidate(p) **then**
43: **if** IsAnchor(p) **then** InsertAnchor(p);
44: **else** CheckParentAndInsert(p);

45: **function** UpdateAnchor()
46: **for all** records $r \in ANCHOR^R$ **do**
47: **if** IsCandidate(r) **then** CheckChildrenAndInsert(r);
48: **else** CheckParentAndInsert(r);

49: **function** GetTOPK(r)
50: $visited(r) =$ *TRUE*;
51: **if** $exact(r) =$ *FALSE* **then**
52: compute $score(r)$;
53: $exact(r) =$ *TRUE*;
54: **if** $|TOPK^S| < k$ **then** insert r into $TOPK^S$;
55: **else if** $score_k^S \leq score(r)$ **then**
56: remove the k-th record from $TOPK^S$;
57: insert r into $TOPK^S$;
58: update $score_k^S$;
59: **for all** records $p \in P(r)$ **do**
60: **if** $visited(p) =$ *FALSE* **then** GetTOPK(p);

the CheckChildrenAndInsert() function. If there exists one or more descendant records that are part of $ANCHOR^S$, such records are added to $ANCHOR^S$. Note that path lengths to reach such records are no more than three, as per Theorem 3. Otherwise, r itself is added to $ANCHOR^S$. On the other hand, if r is not part of $CAND^S$, UpdateAnchor() examines the ancestor records of r in a similar manner. The path length to reach any feasible record is no more than four, as per Theorem 4. The function CheckParentAndInsert() is used to perform this task.

Due to the high cost of computing the exact domination score, ABA computes the exact score for a record only when the record may be added to $CAND^S$. Furthermore, unlike previous methods, we compute the domination score at most once for each record by traversing the CDG downward from the record of interest. We note that while traversing downward, we may compute exact scores for all records in $D(r)$ simultaneously, which further results in a reduction of the computation cost.

In the final stage of the ABA, we obtain the updated query result $TOPK^S$ using $ANCHOR^S$, as described earlier. This function is specified by GetTOPK(), which traverses the CDG upward from the records in $ANCHOR^S$. The accuracy of ABA is verified by confirming that the $ANCHOR$ set is accurately updated when processing an update to the database.

Thus far, we have focused on processing regular updates to the system, in which the sliding window has already been filled. In fact, ABA can be readily extended to deal with the initialization of the system before this condition is reached. In this case, ABA constructs initial CDG as data arrives in an incremental manner with minor modifications as described below. First, since no record is deleted from the system at this stage, we can simply skip the operation associated with record deletion (lines 1-9) in the ABA. Second, $TOPK$, $ANCHOR$, and $score_k$ are not defined until

there are k records in the system. Hence, there is no need to update $TOPK$ and $ANCHOR$ sets before we have k records in the system and, thus, the operation of lines 15-19 in ABA can be ignored during this period. Once the k-th arrival occurs, the system returns all k existing records as the initial $TOPK$ set to the user. In addition, the first $ANCHOR$ is given as the set of all record r for which $C(r) = \varnothing$. We also derive the initial $score_k$ at this time.

5 Performance Evaluation

In this section, we first analyze the performance of the ABA algorithm, followed by presenting experimental results for the algorithm over several different data sets of distinct characteristics. We also compare the proposed algorithm with the existing solutions.

5.1 Performance Analysis

In the ensuing analysis, we assume that records in a record set R are uniformly distributed in a d-dimension space as commonly adopted in previous research [7], [35] and there are n records in R, i.e. $\mid R \mid = n$. The arrival of each new record is accompanied by the removal of the oldest record. We will focus on the update process specified by the ABA algorithm. More specifically, our analysis will respectively address the three major operations of ABA, namely, updating the CDG, generating the $ANCHOR$ set, and computing the new query result.

Recall that the domination score of a record is exactly the number of records that are dominated by the record, i.e., the size of the descendants of the record in the associated CDG graph. Consider an arbitrary record $r \in R$. With uniform distribution, we can readily see that the probability that r dominates another arbitrary record is $(\frac{1}{2})^d$ since the probability that r is smaller than r' in a dimension is $\frac{1}{2}$. Therefore, the average size of $D(r)$ is $\frac{n-1}{2^d}$, which is the same as that of $A(r)$. According to [40], the expected number of skyline records for a set X that contains m records in d-dimensional space is given by $E(\mid \overline{sky}(X) \mid) = O\left(\frac{\ln^{d-1}(m)}{(d-1)!}\right)$, where $\overline{sky}(X)$ represents the result of a skyline query over the set X. Note that the set of records that are closely dominated by r is exactly the set of records that form the skyline result of $D(r)$. Therefore, the average number of records that are closely dominated by r is approximately

$$E(\mid C(r) \mid) = O\left(\sum_{i=1}^{n-1} P_i \cdot \frac{\ln^{d-1}(i)}{(d-1)!}\right)$$

where P_i is the probability there are exactly i records in $D(r)$ and $P_i = \binom{n-1}{i}\left(\frac{1}{2^d}\right)^i\left(1-\frac{1}{2^d}\right)^{(n-1)-i}$. We readily have that $E(\mid P(r) \mid) = E(\mid C(r) \mid)$.

Subsequently, we analyze the time complexity of updating the CDG. When handling the removal of the expiring record r_{exp}, we need to check each of r_{exp}'s parent records, r_{exp}'s children records, and also the parents of r_{exp}'s children (r_{exp} itself is excluded). Therefore, the associated computation cost is proportional to the total number of these records. Furthermore, we need to decrease the domination score for each record in $A(r_{exp})$, for which the average cost is $O\left(\frac{n}{2^d}\right)$. In contrast, when handling the addition of the new incoming record r_{inc}, we use the *kd-tree* to find r_{inc}'s parent and children records with a cost of $O(log\ n)$ [38], [39]. After updating the CDG with the addition of r_{inc}, we need to increase the domination scores of all records in $A(r_{inc})$. This operation is traversal-based and has a time complexity similar to that of the previous operation of decreasing the domination scores. Note that the exact domination score is computed at most once for any record. Moreover, when we compute the exact domination score for a record r, we can simultaneously obtain the exact domination scores for records in $D(r)$ (if they have not yet been computed) by taking advantages of the close dominance relations embraced in the CDG. This task also has time complexity that is similar to the above-mentioned operations for increasing and decreasing the scores.

Next, we analyze the cost of maintaining the $ANCHOR$ set at each update. Assume that the system state changes from R to S upon an update to the database. In ABA, we need to examine the records in $ANCHOR^R$ in order to obtain $ANCHOR^S$ when processing the update. More specifically, for each record $r \in ANCHOR^R$, we may need to traverse the CDG from r either downward by no more than three hops or upward by no more than four hops in the worst case. When n is large, the set $CAND^R$ only occupies a small proportion of the set R. In other words, $\mid D(r) \mid$ tends to be proportionally much larger than $\mid A(r) \mid$, thus traversing the CDG downward will be more costly than traversing the graph upward. Consider the downward traversal from all records in $ANCHOR^R$. The induced computation cost is proportional to the total number of records visited. According to Theorem 2, for any record $r' \in R$ and $r' \notin CAND^R$, there is at least one record in $ANCHOR^R$ that dominates r'. Let $NCAND^R$ represent the set of records in R that are not in $CAND^R$, i.e. $NCAND^R = R - CAND^R$. The records visited as a result of the downward-traversal operation can be readily computed as follows. First, we take skyline query over the set $NCAND^R$ and then remove the result $\overline{sky}(NCAND^R)$ from $NCAND^R$. We then repeat the above process two more times. The union of the three skyline result sets constitutes the maximum set of records that need be visited for the above-mentioned downward traversal operation. Hence, the average computation cost for the maintenance of the $ANCHOR$ set is $O\left(3 \cdot \frac{\ln^{d-1}(n)}{(d-1)!}\right)$. Once the $ANCHOR$

set is updated, the new $TOPK$ result can be obtained by traversing the CDG upward from the $ANCHOR$ set. Hence, the traversal cost depends on the total number of candidates, which is proportionally small when n is large.

5.2 Experiment and Performance Comparison

In this paper, we have conducted experiment over different datasets in order to practically evaluate the performance of the proposed method. We also compare our results with those obtained from the existing solutions in [7].

All algorithms in this experiment are implemented in C++ and conducted on a Core i7 CPU@2.93GHz Win 7 machine, with 4GB of memory. Three different datasets are used in the experiment. The IND (independent) and ANT (anticorrelated) datasets are synthetic datasets [3], while the real-life dataset called FC (Forest Cover) (http://archive.ics.uci.edu) is a multivariate dataset with 581,012 records and 54 attributes. The default values for the parameters are: $d = 4$, $k = 256$, and $n = 3M$ (for FC, $n = 300K$).

5.2.1 Sizes of $CAND$ and $ANCHOR$, and Memory Usage of ABA

Since the sizes of the candidate and $ANCHOR$ sets are critical for the efficiency of the ABA algorithm, we first evaluate these sizes with respect to dimensionality (d), number of answers (k), and window size (n). Let δ denote the average ratio of the size of candidate set to the window size, i.e $\delta = \frac{|CAND|}{n}$, and μ denote the average ratio of the size of $ANCHOR$ set to the window size, i.e $\mu = \frac{|ANCHOR|}{n}$, in the following discussion.

TABLE 2: δ and μ vs. dimensionality (d)

d	ANT ($n = 3M$)		IND ($n = 3M$)		FC ($n = 300K$)	
	δ	μ	δ	μ	δ	μ
2	0.041%	0.007%	0.036%	0.016%	0.13%	0.007%
4	0.076%	0.023%	0.03%	0.017%	0.11%	0.011%
7	0.13%	0.064%	0.029%	0.019%	0.09%	0.016%
10	0.18%	0.071%	0.028%	0.02%	0.09%	0.028%
12	0.18%	0.096%	0.028%	0.02%	0.09%	0.028%

TABLE 3: δ and μ vs. number of answers (k)

k	ANT ($n = 3M$)		IND ($n = 3M$)		FC ($n = 300K$)	
	δ	μ	δ	μ	δ	μ
64	0.037%	0.02%	0.012%	0.009%	0.05%	0.007%
256	0.076%	0.023%	0.03%	0.017%	0.11%	0.011%
512	0.081%	0.024%	0.038%	0.019%	0.19%	0.014%
768	0.112%	0.032%	0.046%	0.02%	0.3%	0.018%

TABLE 4: δ and μ vs. window size (n)

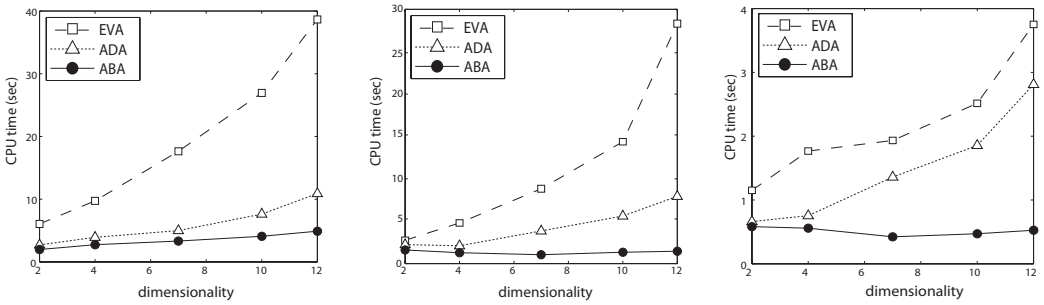
n	ANT		IND		n	FC	
	δ	μ	δ	μ		δ	μ
$1M$	0.136%	0.033%	0.066%	0.031%	100K	0.34%	0.033%
$2M$	0.099%	0.031%	0.039%	0.021%	200K	0.17%	0.017%
$3M$	0.076%	0.023%	0.03%	0.017%	300K	0.11%	0.011%
$4M$	0.067%	0.023%	0.027%	0.013%	400K	0.09%	0.009%
$5M$	0.062%	0.022%	0.026%	0.012%	500K	0.07%	0.006%

Table 2 gives δ and μ versus d for the datasets. The results show that ABA yields very small candidate and $ANCHOR$ sets for all datasets. In particular, $ANCHOR$ set occupies no more than 0.1% of the window size in all cases, which is advantageous for the update process. In the table, ANT gives a larger δ than IND. In fact, the experimental results (not shown in the paper) indicate that ANT generates a larger δ than FC with the same window size of $n = 300K$. This is because, in comparison with IND and FC, the difference of domination scores for records in ANT is smaller in general, thus leading to a larger size of candidate set. Note that δ tends to decline for both IND and FC datasets when d increases. This is because two arbitrary records are less likely to have dominance relation when d grows, which leads to lower domination scores for the records in general. Therefore, it becomes more selective for a record to be included in the candidate set. However, contradictory to the cases for IND and FC, δ slightly increases for ANT when d grows larger. This is due to the fact that variation of the domination scores of the records decreases for ANT when d increases, which tends to yield a larger candidate set. Furthermore, when d becomes larger, the records in a candidate set are more likely to be flattened, and thus more candidate records are included in the corresponding $ANCHOR$ set. Such trend is observed for all datasets.

Table 3 gives δ and μ with respect to different values of k. Note that when k increases, $score_k$ decreases in general, thus rendering a record more likely to be included in the candidate set. Such effect is readily observed for all datasets in the table.

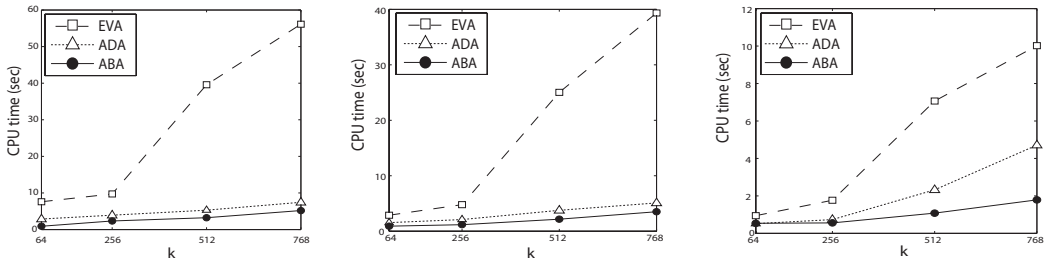
In Table 4, we show δ and μ with respect to different values of n, the window size. Both δ and μ steadily decline for all the datasets when n increases. In particular, FC demonstrates a declining rate that leaves the actual sizes of the candidate and $ANCHOR$ sets virtually unchanged across different values of n.

Next, we examine the memory usage of the proposed scheme. Table 5 shows the maximum sizes of memory used by ABA with different dimensionality and window size. We can clearly see that the memory usage by ABA increases with both the increment of dimensionality and window size. Note that ABA consumes more memory when it runs with the ANT dataset in comparison with the IND dataset. This is



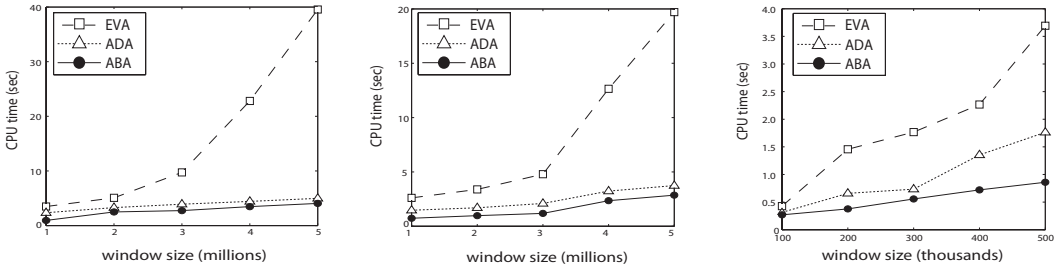
(a) ANT (b) IND (c) FC

Fig. 3: CPU time vs. dimensionality (d).



(a) ANT (b) IND (c) FC

Fig. 4: CPU time vs. number of answers (k).



(a) ANT (b) IND (c) FC

Fig. 5: CPU time vs. window size (n).

because a record has a higher degree, in average, with the ANT dataset than with the IND dataset.

TABLE 5: Memory Usage of ABA

dataset	n	d				
		2	4	7	10	12
ANT	1M	8.738 MB	9.049 MB	9.131 MB	9.179 MB	9.292 MB
	2M	17.405 MB	17.412 MB	17.456 MB	17.869 MB	18.324 MB
	3M	26.069 MB	26.075 MB	26.081 MB	26.096 MB	26.822 MB
	4M	33.005 MB	34.735 MB	34.751 MB	34.762 MB	34.791 MB
	5M	43.398 MB	43.401 MB	43.471 MB	43.483 MB	43.494 MB
IND	1M	8.074 MB	8.076 MB	8.352 MB	8.368 MB	8.487 MB
	2M	16.082 MB	16.086 MB	16.099 MB	16.602 MB	16.646 MB
	3M	24.086 MB	24.088 MB	24.091 MB	24.523 MB	24.768 MB
	4M	32.094 MB	32.098 MB	32.105 MB	32.115 MB	32.211 MB
	5M	35.497 MB	37.436 MB	37.820 MB	38.441 MB	40.099 MB
FC	100K	1.889 MB	1.931 MB	2.933 MB	4.401 MB	4.564 MB
	200K	3.622 MB	3.759 MB	4.348 MB	4.823 MB	4.876 MB
	300K	5.644 MB	5.661 MB	5.972 MB	6.063 MB	7.442 MB
	400K	7.409 MB	8.821 MB	9.347 MB	9.446 MB	9.704 MB
	500K	8.946 MB	9.013 MB	9.939 MB	9.947 MB	10.111 MB

5.2.2 Runtime of the Algorithms

In the experiment, we have measured the runtime required by ABA to process updates to the database. We also compare our results with those obtained using the Event-Based Algorithm (EVA) and the Advanced Algorithm (ADA) proposed in [7]. In the ensuing discussion, runtime is measured for 1000 updates.

First, we compare the performance of the algorithms with respect to different dimensionality d for the above-mentioned datasets. Fig. 3 shows the results when d varies from 2 to 12. ABA clearly outperforms the other algorithms in virtually every case of the experiment. The advantages of ABA over other algorithms become especially evident when the dimensionality of the datasets increases. Note that the runtime steadily increases for EVA and ADA when d grows larger with all datasets. However, when ABA is used, it first declines with IND and FC from 2 to

7 dimensions and then increases mildly from 7 to 12 dimensions. This is because when d is small, the degree of each record in the CDG is comparatively high which leads to a higher cost of obtaining the corresponding new $ANCHOR$ records for an individual existing $ANCHOR$ record. Meanwhile, it takes more time to update the CDG when the dimensionality grows due to a higher dominance checking cost. This factor is more evident when d increases from 7 to 12.

Next, we examine how performance of the algorithms reacts to the variation of the number of answers (k). In Fig. 4, we depict the experimental results with k varies from 64 to 768. ABA obviously offers much better performance over the other algorithms. As anticipated, when k increases, all of the algorithms require more runtime to complete the update process. However, ABA is least sensitive to the increment of k in this respect when compared with the other algorithms.

Finally, we study the performance with respect of the window size (n). The experimental results are illustrated in Fig. 5 in which the window size varies from 1M to 5M for ANT and IND datasets, and from 100K to 500K for FC dataset. Again, ABA is the winner in this part of the experiment. All of the algorithms under study require more runtime when the window size increases with ABA demonstrating the smallest increasing rate. This is important for the scalability of the proposed scheme.

6 Alternative Streaming Models

In this section, we discuss how the proposed algorithm can be adapted to other streaming models. One common type of streaming model is the count-based sliding window with multiple arrivals and removals. Unlike the previous model, there are m new records arriving, accompanied by m records expiring, at each update. Since the domination score of an existing record can increase or decrease by at most m under such model, Theorem 1 is valid if the conditions are changed to as follows: (1) $score^S(r) < score^R_{k+m-1} - 2m$ and (2) there exists some $z \in S$ such that $z \rightarrow r$ exists in the updated CDG and $score^S(z) < score^R_{k+m-1} - m$. Consider the process of each update in this case. Although we can simply repeat the operation specified in ABA m times, with one addition and one removal at each time, we can, in fact, update the CDG with all arrivals and departures, and then follow the same operation as ABA to update the $ANCHOR$ set. However, the IsCandidate() function need to be modified according to the conditions specified above.

A more complicated streaming model to consider is a time-based sliding window with multiple arrivals and removals. In this model, the numbers of arrivals and departures are not equal and are different for different updates. Assume that an update, involving e expiring records and i incoming records, changes system state from R to S. Theorem 1 remains valid if the conditions are modified as follows: (1) $score^S(r) < score^R_{k+e-1} - (e+i)$ and (2) there exists some $z \in S$ such that $z \rightarrow r$ exists in the updated CDG and $score^S(z) < score^R_{k+e-1} - e$. Similarly, we can handle this streaming model by simply revising the IsCandidate() function accordingly.

7 Conclusion

This paper has proposed an unique indexing structure, called Close Dominance Graph (CDG), to serve as a framework for supporting the processing of a cTKDQ. The comprehensive information contained in CDG is highly useful for reducing the search space when processing an update to the database. In particular, we have presented the notion of $ANCHOR$ set to facilitate the processing of each update to the database. Useful properties associated with CDG have been derived in this paper. Essentially, the update process is converted to a simple update to the $ANCHOR$ set as specified in the ABA algorithm. We have evaluated the performance of the proposed scheme both analytically and by extensive experiments using synthetic and real-world datasets. The experimental results show that ABA significantly outperforms existing solutions in virtually all of the settings.

There are several possible directions for future study. First, it would be interesting if the idea can be extended to distributed computing environment, in which the entire database is distributed among multiple sites. In this case, the main concern will center on efficient transfer of data among these data sources. There is also possibility that CDG can be utilized to solve other data processing problems, in which domination power matters.

Acknowledgments

The authors are grateful to the anonymous referees for their valuable suggestions and comments which have helped improve the quality of the paper. This work was supported by the National Science Council, Taiwan, under grants NSC 100-2221-E-011-078, 101-2221-E-011-100, and 102-2218-E-011-007.

References

[1] R. Fagin, "Combining fuzzy information from multiple systems (extended abstract)," in *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS)*, 1996, pp. 216–226.

[2] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS)*, 2001, pp. 102–113.

[3] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. Int'l Conf. Data Engineering (ICDE)*, 2001, pp. 421–430.

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, Mar. 2005.

[5] A. N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos, "Domination mining and querying," in *Proc. Int'l Conf. Data Warehousing and Knowledge Discovery (DAWAK)*, 2007, pp. 145–156.
[6] M. Kontaki, A. Papadopoulos, and Y. Manolopoulos, "Continuous top-k dominating queries in subspaces," in *Proc. Panhellenic Conf. Informatics (PCI)*, 2008, pp. 31–35.
[7] M. Kontaki, A. Papadopoulos, and Y. Manolopoulos, "Continuous top-k dominating queries," *IEEE Trans. Knowledge and Data Eng.*, vol. 24, no. 5, pp. 840–853, May 2012.
[8] L. Zou and L. Chen, "Pareto-based dominant graph: An efficient indexing structure to answer top-k queries," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 5, pp. 727–741, May 2011.
[9] T. Tran, H. Wang, S. Rudolph, and P. Cimiano, "Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data," in *Proc. Int'l Conf. Data Engineering (ICDE)*, 2009, pp. 405–416.
[10] R. Akbarinia, E. Pacitti, and P. Valduriez, "Reducing network traffic in unstructured p2p systems using top-k queries," *Distributed and Parallel Databases*, vol. 19, no. 2-3, pp. 67–86, May 2006.
[11] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2003, pp. 28–39.
[12] T. Chen, L. Chen, M. Ozsu, and N. Xiao, "Optimizing multi-top-k queries over uncertain data streams," *IEEE Trans. Knowledge and Data Eng.*, vol. 25, no. 8, pp. 1814–1829, Aug. 2013.
[13] M. Wu, J. Jianliang Xu, X. Xueyan Tang, and W.-C. Wang-Chien Lee, "Top-k monitoring in wireless sensor networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 7, pp. 962–976, Jul. 2007.
[14] H. Jiang, J. Cheng, D. Wang, C. Wang, and G. Tan, "A general framework for efficient continuous multidimensional top-k query processing in sensor networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 9, pp. 1668–1680, Sept. 2012.
[15] M. Ye, W.-C. Lee, D. L. Lee, and X. Liu, "Distributed processing of probabilistic top-k queries in wireless sensor networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 25, no. 1, pp. 76–91, Jan. 2013.
[16] G. Wang, J. Xin, L. Chen, and Y. Liu, "Energy-efficient reverse skyline query processing over wireless sensor networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 24, no. 7, pp. 1259–1275, Jul. 2012.
[17] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, vol. 40, no. 4, pp. 1–58, 2008.
[18] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis, "On efficient top-k query processing in highly distributed environments," in *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2008, pp. 753–764.
[19] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag, "Monochromatic and bichromatic reverse top-k queries," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 8, pp. 1215–1229, Aug. 2011.
[20] Z. He and E. Lo, "Answering why-not questions on top-k queries," *IEEE Trans. Knowledge and Data Eng.*, preprint, 14 Aug. 2012, doi:10.1109/TKDE.2012.158.
[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2001, pp. 301–310.
[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2003, pp. 467–478.
[23] S. Wang, Q. Vu, B. Ooi, A. Tung, and L. Xu, "Skyframe: a framework for skyline query processing in peer-to-peer systems," *The VLDB Journal*, vol. 18, no. 1, pp. 345–362, 2009.
[24] X. Liu, D. Yang, M. Ye, and W. Lee, "U-skyline: A new skyline query for uncertain databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 25, no. 4, pp. 945–960, Apr. 2013.
[25] X. Lin, J. Xu, and H. Hu, "Range-based skyline queries in mobile environments," *IEEE Trans. Knowledge and Data Eng.*, vol. 25, no. 4, pp. 835–849, Apr. 2013.
[26] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "Skypeer: Efficient subspace skyline computation over distributed data," in *Proc. Int'l Conf. Data Engineering (ICDE)*, Apr. 2007, pp. 416–425.
[27] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: efficient skyline computation over sliding windows," in *Proc. Int'l Conf. Data Engineering (ICDE)*, 2005, pp. 502–513.
[28] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *Proc. Int'l Conf. Data Engineering (ICDE)*, 2009, pp. 892–903.
[29] K. C. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-sky: an efficient skyline query processing framework based on z-order," *The VLDB Journal*, vol. 19, no. 3, pp. 333–362, Jun. 2010.
[30] M. L. Yiu and N. Mamoulis, "Multi-dimensional top-k dominating queries," *The VLDB Journal*, vol. 18, no. 3, pp. 695–718, Jun. 2009.
[31] X. Lian and L. Chen, "Top-k dominating queries in uncertain databases," in *Proc. Int'l Conf. Extending Database Technology (EDBT)*, 2009, pp. 660–671.
[32] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous processing of preference queries in data streams," in *Proc. Conf. Current Trends in Theory and Practice of Comp. Science (SOFSEM)*, 2010, pp. 47–60.
[33] Z. Huang, H. Lu, B. C. Ooi, and A. Tung, "Continuous skyline queries for moving objects," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 12, pp. 1645–1658, Dec. 2006.
[34] H. Lu, Y. Zhou, and J. Haustad, "Continuous skyline monitoring over distributed data streams," in *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM)*, 2010, pp. 565–583.
[35] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," in *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2006, pp. 635–646.
[36] Z. Shen, M. Cheema, X. Lin, W. Zhang, and H. Wang, "A generic framework for top-k pairs and top-k objects queries over sliding windows," *IEEE Trans. Knowledge and Data Eng.*, preprint, 18 Sept. 2012, doi:10.1109/TKDE.2012.181.
[37] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proc. Int'l Conf. Data Engineering (ICDE)*, 2003, pp. 717–719.
[38] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.
[39] A. Moore, "An introductory tutorial on kd-trees," Ph.D. dissertation, Thesis, Tech. Report, 1991.
[40] P. Godfrey, "Skyline cardinality for relational processing," in *Proc. Foundations of Information and Knowledge Systems (FoIKS)*, 2004, pp. 78–97.

Bagus Jati Santoso received the B.S. degree in informatics department from Sepuluh Nopember Institute of Technology (ITS), Surabaya, Indonesia, in 2008. He is currently pursuing the Ph.D. degree at the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, focusing on data engineering.

Ge-Ming Chiu received the BS degree from the National Cheng Kung University, Taiwan, in 1976, the MS degree from the Texas Tech University, in 1981, and the PhD degree from the University of Southern California, in 1991, all in electrical engineering. He is currently a professor in the Department of Computer Science and Information Engineering at the National Taiwan University of Science and Technology, Taipei, Taiwan. While most of his previous research was focused on parallel architecture, distributed computing, and fault tolerance, his current research interests also include data engineering, mobile computing, mobile ad hoc networks, and sensor networks. He is a member of the IEEE Computer Society.