Venn Sampling: A Novel Prediction Technique for Moving Objects

Yufei Tao
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
taoyf@cs.cityu.edu.hk

Dimitris Papadias
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
dimitris@cs.ust.hk

Jian Zhai
Department of Computer Engineering
and Information Technology
City University of Hong Kong
Tat Chee Avenue, Hong Kong
jian.zhai@student.cityu.edu.hk

Qing Li
Department of Computer Engineering
and Information Technology
City University of Hong Kong
Tat Chee Avenue, Hong Kong
itqli@cityu.edu.hk

Abstract

Given a region q_R and a future timestamp q_T, a "range aggregate" query estimates the number of objects expected to appear in q_R at time q_T. Currently the only methods for processing such queries are based on spatio-temporal histograms, which have several serious problems. First, they consume considerable space in order to provide accurate estimation. Second, they incur high evaluation cost. Third, their efficiency continuously deteriorates with time. Fourth, their maintenance requires significant update overhead.

Motivated by this, we develop Venn sampling (VS), a novel estimation method optimized for a set of "pivot queries" that reflect the distribution of actual ones. In particular, given m pivot queries, VS achieves perfect estimation with only $O(m)$ samples, as opposed to $O(2^m)$ required by the current state of the art in workload-aware sampling. Compared with histograms, our technique is much more accurate (given the same space), produces estimates with negligible cost, and does not deteriorate with time. Furthermore, it permits the development of a novel "query-driven" update policy, which reduces the update cost of conventional policies significantly.

1. Introduction

Spatio-temporal databases manage multi-dimensional objects whose location changes continuously with time. Related research typically assumes linear object movement. Specifically, the location $p(t)$ of a d-dimensional point p at time t is given by $p(t)=p(0)+p_V t$, where $p(0)$ is the location of p at time 0, and p_V its velocity ($p(0)$, $p(t)$, p_V are d-dimensional vectors). Whenever the velocity p_V changes, p sends an update message to a central server, which collects information from all objects in order to answer various predictive queries. The most common query type is the *range aggregate* (RA) search, which (based on the current data at the server) returns the number of objects that will fall in a rectangle q_R at a (future) timestamp q_T (e.g., "how many flights are expected to appear over the airspace of Hong Kong 10 minutes from now"). The results of such queries are used for prevention of possible congestions (in traffic supervision and flight control) or network overloading (in mobile computing and location-based services).

RA queries can be processed using two main methodologies. The first one assumes that the server indexes the detailed information of all (moving) objects using a (typically disk-based) spatio-temporal access method [AAE00, SJLL00, AA03, TPS03]. Given a query q, the qualifying objects are retrieved (using the index) and their count is returned as the result. This technique, however, is inefficient in applications (e.g., those handling data streams [DGR03]) with substantial data updates. In this case, the maintenance of the index leads to excessive overhead (especially if I/O operations are involved), thus seriously compromising the system's query processing ability.

The second type of methodologies [CC02, HKT03, TSP03], which is receiving increasing attention, aims at accurately *estimating* the RA results. These approaches are motivated by the fact that approximate aggregates (with small error) are often as useful as the exact ones in practice, but can be obtained much more efficiently. In particular, such estimation methods require only a fraction of the dataset or a small amount of statistics and are significantly faster than exact retrieval. These properties are especially appealing in circumstances where the system memory is limited (relative to the dataset size), yet real-time processing is important. In this paper, we focus on approximate RA processing.

1.1 Motivation

Currently, the only effective methods for estimating RA retrieval are based on histograms. As explained in the next section, however, spatio-temporal histograms have several serious shortcomings (i): they incur large space consumption (which increases exponentially with the dimensionality) in order to provide satisfactory accuracy; (ii) they entail high processing cost; (iii), their precision continuously deteriorates with time and eventually the histogram must be re-built (by accessing the *entire*

database); (iv), their incremental maintenance requires significant overhead for handling data updates. These problems motivate alternative approximation techniques, among which sampling is the most obvious candidate due to its successful application in conventional databases (mostly for selectivity estimation).

It is well-known that if every possible query has the same probability to be issued, a random sample of the database indeed minimizes the expected prediction error [AGP00]. In practice, however, queries follow certain "locality patterns" [CDN01], i.e., some values of the (query) parameters are more frequent than others. For example, in a traffic-supervision system, users are typically more concerned about the traffic volume in central regions than in the suburbs. This motivates *workload-aware methods*, which aim at maximizing the estimation accuracy for a set of *pivot queries* (extracted from historical statistics) that reflect the actual query distribution. Since future queries are expected to be similar to the pivot ones, workload-aware sampling provides better overall precision than random sampling (which essentially attempts to optimize all queries, many of which may never be raised).

Unfortunately, the unique characteristics of moving objects render most existing sampling techniques inapplicable. First, the extremely frequent updates necessitate incremental maintenance of the sample set, eliminating algorithms that target static or "append-only" data [V85]. Second, the server keeps limited information about the dataset, invalidating methods that need to access a large portion of the database [GLR00]. Third, the exact result of a query is not computed (the estimate produced from the samples is directly returned to the user), canceling techniques that utilize query feedback [BGC01]. Finally, spatio-temporal sampling must take into account the underlying client-server architecture. Namely, in addition to optimizing space/time efficiency, it should also minimize the messages exchanged between the server and objects for maintaining the sample.

1.2 Contributions

This paper proposes *Venn sampling* (VS), a workload-aware methodology based on solid theoretical foundation. VS generalizes *stratified sampling* [CDN01] (a well-known workload-aware method) to highly dynamic environments, while at the same time it reduces significantly its space requirements. In particular, if m is the number of pivot queries, the memory consumption of VS is $O(m)$, as opposed to $O(2^m)$ for stratified sampling. VS guarantees *perfect estimation* (i.e., no error at all) for pivot queries and, therefore, incurs small error for actual queries. Interestingly, this is achieved using a *fixed* sample set, namely, the samples of VS remain the same *even if* the object information changes continuously.

VS is a general methodology that can be applied in a large number of scenarios. Its application to moving objects motivates a novel *query-driven update* policy.

Specifically, an object issues an update to the server only when it starts/ceases to satisfy some pivot query, *independently of* its velocity changes (recall that the conventional policy requires an update whenever an object's velocity changes). Query-driven updates constitute a natural concept because, intuitively, although real systems may involve millions of continuously moving objects, we only care about the ones that influence some query. We experimentally verify that VS outperforms spatio-temporal histograms on every aspect, i.e., *accuracy*, *evaluation efficiency*, *space consumption*, *communication overhead* and *performance deterioration with time*.

The rest of the paper is organized as follows. Section 2 introduces the related work on spatio-temporal histograms and sampling techniques. Section 3 formally defines the problem and overviews the proposed architecture. Section 4 presents the theory of VS and discusses its general applicability. Section 5 contains a comprehensive experimental evaluation, and Section 6 concludes the paper with directions for future work.

2. Related Work

Section 2.1 reviews spatio-temporal histograms for RA processing, and illustrates their defects. Section 2.2 surveys previous database sampling techniques.

2.1 Spatio-temporal histograms

Each d-dimensional linearly moving object can be converted to a point in a $2d$-dimensional space, capturing both its coordinates (at reference time 0) and velocities. Figure 2.1a illustrates an example with $d=1$. Point p_1 (in the converted space) represents an object whose location at time 0 (the horizontal axis) is at coordinate 15, and whose current velocity (the vertical dimension) equals 20 (i.e., its position $p_1(t)$ at time t is computed as $15+20 \cdot t$). Similarly, point p_2, with location 35 and velocity 20, corresponds to a moving object with motion $p_2(t)=35+20 \cdot t$.
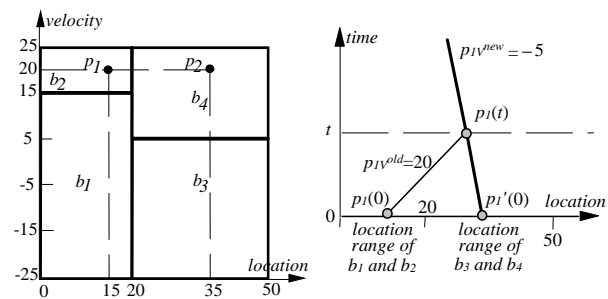
(a) A 1D histogram   (b) Updating the histogram
Figure 2.1: Spatio-temporal histograms

A spatio-temporal histogram [CC02, HKT03, TSP03] partitions the $2d$-dimensional (converted) space into disjoint hyper-rectangular *buckets*. Each bucket b is associated with the number n_b of points in its extent. In Figure 2.1a, the histogram consists of 4 buckets $b_1, \ldots, b_4$; for instance, the extent of b_2 covers the range [0, 20] ([15,

25]) on the location (velocity) dimension, and includes only one point p_1 (i.e., $nb_2=1$). The number of buckets is subject to the amount of available space. In general, a larger number leads to better estimation accuracy.

To answer a RA query q using the histogram, the processing algorithm first computes, for every bucket b, its "contribution" b^{con} to the query result, which equals the number of objects in b that are *expected* to satisfy q. Then, the (approximate) result of q is obtained by summing the contributions of all buckets. In particular, b^{con} is calculated as $n_b \cdot P_b$, where P_b is the probability that an object in b qualifies q (see [CC02, TSP03] for details).

The spatio-temporal histogram can be dynamically maintained in the presence of object updates. Assume that object p_1 in Figure 2.1a changes its velocity (from $p_{1V}^{old}=20$) to $p_{1V}^{new}=-5$ at time t. As a result, p_1 no longer belongs to bucket b_2 (since -5 is not in the velocity range [15, 25] of b_2); thus, nb_2 decreases by 1. Deciding the bucket that covers the updated p_1 should take into account the fact that the histogram includes objects' location at the reference time 0 (instead of the current time). To illustrate this, Figure 2.1b shows the previous update in the *location-time* space. The position $p_1(t)$ (of p_1 at time t) is a point on the horizontal line *time=t* (the segment connecting $p_1(0)$ and $p_1(t)$ has slope $p_{1V}^{old}=20$). We find a point $p_1'(0)$ at time 0 such that it crosses $p_1(t)$ with the new velocity $p_{1V}^{new}=-5$. Specifically, $p_1'(0)$ is the intersection between the horizontal axis and the line with slope -5 crossing $p_1(t)$. Thus, the bucket that includes the new p_1 should be the one (i.e., b_3) whose location *and* velocity ranges contain $p_1'(0)$ and p_{1V}^{new}, respectively. The counter of this bucket is increased by 1.

The accuracy of histograms relies on the assumption that the data distribution inside each bucket is uniform. However, the number of buckets required to achieve this property increases exponentially with the dimensionality. Unfortunately, this problem is particularly serious in the spatio-temporal context, as the dimensionality of the histogram actually *doubles* that of the data space. Indeed, the application of histograms has been limited to $d=2$ only (e.g., for $d=3$, a prohibitive number of buckets are necessary to satisfy the uniformity assumption in the resulting 6D space [BGC01]). Further, using a histogram to answer a RA query requires solving some complex formulae, which can be evaluated only with numerical approaches [TSP03]. The high cost of numerical evaluation severely compromises the query efficiency.

Another disadvantage of spatio-temporal histograms is that their quality continuously deteriorates with time. This is illustrated in Figure 2.2 in the location-time space ($d=1$). Object p_1 updates its velocity to p_{1V} at time t_1, and its new $p_1(0)$ (computed as in Figure 2.1b) is inside the location space. At some later timestamps ($t_1<t_2<t_3$), objects p_2, p_3 change their velocities to the *same value* as p_1 ($p_{1V}=p_{2V}=p_{3V}$), at the *same location* $p_1(t_1)=p_2(t_2)=p_3(t_3)$. The computed $p_2(0)$ and $p_3(0)$, however, fall outside the

location space. Further, notice that the larger the update timestamp (e.g., $t_2<t_3$), the more distant the reference location $p(0)$ is from the space boundary (i.e., $p_2(0)<p_3(0)$).
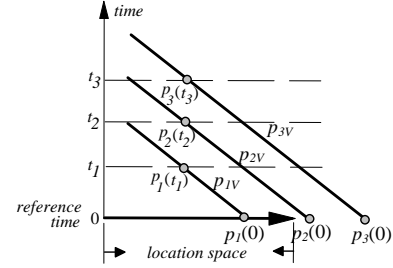

Figure 2.2: Quality degradation with time

In fact, if the location space has range [0, L], then by solving inequality $0 \le p(t)-p_V \cdot t \le L$ we obtain that $p(0)$ lies inside [0, L] only if p_V is in the range [$(L-p(t))/(-t)$, $p(t)/t$] (remember that p_V can be negative). Obviously, as t increases, this "permissible range" continuously shrinks. Eventually, the $p(0)$ of most objects p fall outside [0, L], in which case the histogram becomes useless and must be re-built, choosing a *larger* timestamp as the reference time [TSP03]. The re-building requires accessing the entire database, and thus incurs considerable overhead. Besides periodic re-building, the histogram must be modified with *every* data update. This requires significant communication overhead (for objects to transmit their new information). As a result, in practice histogram-based systems cannot support large datasets.

2.2 Sampling in databases

In *random sampling*, every object has the same probability to be sampled. Given a RA query q, the estimation algorithm counts the number a of sample records qualifying q, and then predicts the query result as $a \cdot (n/k)$, where k is the sample size, and n the dataset cardinality. Numerous variations of random sampling have been proposed in the literature [V85, LNS90, GMP97, GM98, L00, CDD+01]. Other approaches [AGP00, GLR00, WAA01] sample objects with different probabilities, which depend on certain dataset or query properties. Jermaine [J03] proposes a general method to enhance the sampling efficiency.

The technique most related to our solution is *stratified sampling* (SS) [CDN01], which optimizes the estimation accuracy for a set of pivot queries (which reflect future query distribution). It utilizes the concept of *fundamental strata*. Assume that we have two pivot queries q_1 and q_2. All the records can be classified into four strata s_1, s_2, s_3, s_4, depending on whether they satisfy (i) only q_1, (ii) only q_2, (iii) both q_1 and q_2, and (iv) neither query. Obviously, these sets s_i ($1 \le i \le 4$) are mutually exclusive. Furthermore, each pivot query either retrieves *all or none* of the tuples in each s_i (e.g., q_1 returns all records in s_1 and s_3, but none in s_2 and s_4). SS first scans the database and assigns each tuple to exactly one stratum. Then, the algorithm selects

randomly, from the objects assigned to each stratum s_i, a record o_i and associates it with a *weight* w_i that equals the number of tuples in s_i. The pairs $\{(o_i, w_i) \mid 1 \leq i \leq 4\}$ constitute the final sample set.

Given a query q, SS estimates the number of qualifying objects as the sum of weights w_i of the samples o_i satisfying q. For example, the estimation of q_1 equals $w_1 + w_3$, since objects o_1, o_3 qualify it but o_2, o_4 do not. In general, SS answers all pivot queries *precisely* (i.e., no error at all) which, however, is achieved with huge space requirements. Specifically, given m pivot queries, the total number of strata equals $O(2^m)$. Hence, the space consumption of SS increases exponentially with m (which in practice is at the order of 100).

To alleviate this problem, Chaudhuri, et al. suggest keeping the best k samples (see [CDN01] for measuring the quality of each sample), where k ($<<2^m$) is subject to the available memory. This solution, however, still incurs serious problems. First, it loses the quality guarantee that pivot queries are always captured precisely, leading to increased error for actual queries. Second, obtaining the best k samples still requires $O(2^m)$ space, since all the intermediate strata must be retained for this purpose. Third, the smaller sample set cannot be dynamically maintained (a more detailed discussion about this appears in Section 4.4).

3. Problem Definition and System Architecture

Let DS be a set of d-dimensional linearly moving points. We represent the location of object p at time t as a $d \times 1$ vector $\boldsymbol{p}(t) = \{\boldsymbol{p}(t)[0], \boldsymbol{p}(t)[1], \ldots, \boldsymbol{p}(t)[d]\}^T$, where $\boldsymbol{p}(t)[i]$ ($1 \leq i \leq d$) denotes the coordinate of p on the i-th axis. Similarly, vector $\boldsymbol{p}_V = \{\boldsymbol{p}_V[0], \boldsymbol{p}_V[1], \ldots, \boldsymbol{p}_V[d]\}^T$ indicates the velocities of p along various dimensions. Without loss of generality, we consider that each coordinate $\boldsymbol{p}(t_C)[i]$ ($1 \leq i \leq d$) at the current time t_C falls in a unit range [0, 1], and each velocity $\boldsymbol{p}_V[i]$ in $[V_{min}, V_{max}]$, where V_{min} (V_{max}) denotes the minimum (maximum) velocity.

Given a rectangular region q_R and a timestamp q_T, a *range aggregate* (RA) query q retrieves the number of objects expected to appear in q_R at time q_T, based on their current motion parameters. The rectangle q_R is represented using its two opposite corners $\boldsymbol{q_{R-}}$, $\boldsymbol{q_{R+}}$ (each being a d-dimensional point), and the projection of q_R on the i-th ($1 \leq i \leq d$) dimension is $[\boldsymbol{q_{R-}}[i], \boldsymbol{q_{R+}}[i]]$. Formally, the result of q is RA(q) = COUNT$\{p \in DS \mid \boldsymbol{q_{R-}}[i] \leq \boldsymbol{p}(q_T + t_C)[i] \leq \boldsymbol{q_{R+}}[i]$ for all $1 \leq i \leq d$, where t_C is the current time, and $\boldsymbol{p}(q_T + t_C)[i] = \boldsymbol{p}(t_C)[i] + \boldsymbol{p}_V \cdot q_T\}$. Note that q_T is always relative to the current time t_C, e.g., $q_T = 2$ should be interpreted as "two timestamps *later*".

The goal is to accurately estimate the results of RA queries. The accuracy of a query is measured as its relative error $|act - est|/act$, where act and est denote the actual and estimated results, respectively. Let Q be a *workload* with m queries and act_i (est_i) be the actual (estimated) result of the i-th query ($1 \leq i \leq m$). Then, the *workload error* $WE(Q)$ is defined as:

$$WE(Q) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left[\left(act_i - est_i \right) \middle/ act_i \right]^2} \qquad (3\text{-}1)$$

Next we discuss the intuition behind *Venn sampling* (VS). As with stratified sampling (SS), VS is workload-aware, namely, it is optimized for a set PvQ of m *pivot queries*. Specifically, PvQ is decided in a way such that, if a method can efficiently support PvQ, then it is expected to perform well also for actual queries, which are similar to those in PvQ (see [CDN01] for a general method of selecting pivot queries). We consider that the query distribution changes infrequently [CDN01] so that PvQ remains constant for relatively long periods of time.

VS obtains *perfect estimation* for all the pivot queries (i.e., zero error for PvQ). Unlike SS which requires $O(2^m)$ space for this purpose, the space consumption of VS is only $O(m)$. The sample set of VS contains m moving objects o_1, o_2, … o_m, and each o_i ($1 \leq i \leq m$) is associated with a weight w_i (which may be positive or negative). Interestingly, the sample objects *do not necessarily belong to the underlying dataset*. Instead, they are "formulated" according to the pivot queries (as elaborated in the next section), and are *not* modified along with the data updates. Given a RA query q, VS estimates its result in the same way as SS: the estimate *est* equals the sum of the weights of samples satisfying q.

Figure 3.1 demonstrates the architecture of the proposed system. Specifically, VS maintains the actual results act_1, act_2, …, act_m of all m pivot queries (i.e., act_i is the *exact* number of objects qualifying the i-th query). We use an $m \times 1$ vector $\textbf{ACT} = \{act_1, act_2, \ldots, act_m\}^T$ to encode these results concisely. Meanwhile, VS continuously monitors the quality of the sample set. For this purpose, the "quality control" component calculates an $m \times 1$ vector $\textbf{EST} = \{est_1, est_2, \ldots, est_m\}^T$, where est_i is the estimation of the i-th query ($1 \leq i \leq m$) using the current samples. Then, it computes the error $WE(PvQ)$ of PvQ by equation 3-1 using $\textbf{EST}$ and $\textbf{ACT}$. If the error is above a certain threshold ε, the sample weights (but *not* the sample objects) are adjusted so that $\textbf{EST} = \textbf{ACT}$ (i.e, $WE(PvQ) = 0$). This (weight) adjustment requires only solving a set of linear equations, and is highly efficient.

Following the common convention, we assume that the objects have basic computing capabilities (including a small amount of memory) and that they communicate with the server using a wireless network. During initialization, the VS system broadcasts[1] the set PvQ of pivot queries to all the objects, which store them locally using $O(m)$ space. Each update message from an object p has the form $<s_p^{old}, s_p^{new}>$. Specifically, s_p^{old} is an $m \times 1$ bit vector $(b_1, b_2, \ldots, b_m)^T$ called *signature*, where $b_i = 1$ ($1 \leq i \leq m$) if the *previous* information of p (i.e., before the update) satisfies the i-th query, and $b_i = 0$ otherwise. The

---

[1] The cost of broadcasting is significantly smaller than that of sending a large number of individual messages and does not increase with the dataset cardinality.

signature s_p^{new} is also an $m\times1$ bit vector, except that each bit denotes if the object's *new* information qualifies the corresponding query. When an update arrives, the system maintains the actual result as $\mathbf{ACT} = \mathbf{ACT} - s_p^{old} + s_p^{new}$.
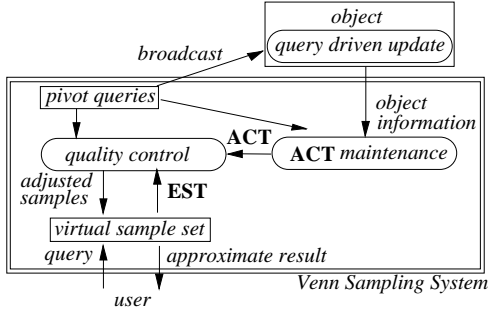

Figure 3.1: System architecture

An update message is necessary only when *the object starts or ceases to satisfy some pivot query*. To achieve this, every time an object issues an update, it preserves the s_p^{new} just transmitted (to the server) locally as s_p^{old}. At the next timestamp, the object p obtains its current location and velocity, and computes the new s_p^{new} (i.e., the set of pivot queries it satisfies now). An update is sent to the server only if $s_p^{new} \neq s_p^{old}$. Figure 3.2a illustrates this process with two objects whose (i) position at the current time 0 is denoted by black points, and (ii) their velocity on each axis is represented by an arrow with a number indicating the speed (p_1 moves northeast with speed $\sqrt{8}$ and p_2 northwest with speed $\sqrt{18}$). The shaded area corresponds to the region q_R of a pivot query q with $q_T=2$. At time 0, the result of q is 1, since (based on the current information) $p_1(2)$ falls in q_R. As shown in Figure 3.2b, at the next timestamp p_1 ceases to satisfy the query ($p_1(3)$ is out of q_R) and issues an update to the server. Although at the same timestamp, p_2 incurs a velocity change (the direction becomes north), it does *not* inform the server because the change causes no effect to the query result (p_2 still does not qualify q).
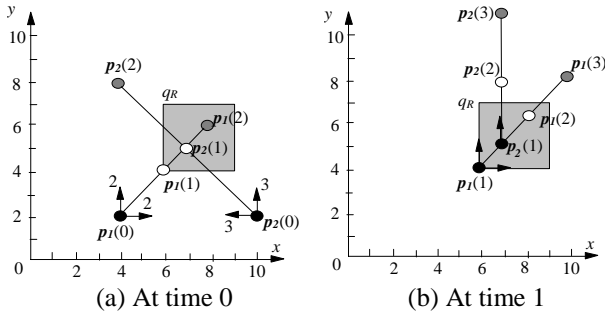

(a) At time 0   (b) At time 1
Figure 3.2: Illustration of query-driven update

Compared to the conventional *velocity-driven* update policy, the above *query-driven* approach significantly reduces the number of necessary update messages (sent by objects to the server) in most practical situations because it avoids messages from objects that do not influence any pivot query. For example, if all the pivot queries are in certain regions of the data space (e.g., around the centers of urban areas), then objects outside these regions (e.g., in the suburbs) do not need to issue updates no matter *how often* they change their velocities.

4. Theory of Venn Sampling

Section 4.1 presents the main algorithmic framework of VS. Section 4.2 solves a problem fundamental to obtaining the optimal sample weights. Section 4.3 elaborates the sample computation. Finally, Section 4.4 discusses the application of VS to other problems and clarifies its connection with stratified sampling.

4.1 Main idea

VS is motivated by the *Venn diagram*, which is a common tool for expressing the relationships among sets. In our context, we define the *Venn area* $\Theta(q)$ of a RA query q as the set of *all possible* objects that satisfy it. Specifically, if we regard a moving point o as a pair of vectors ($o(t_C)$, o_V), then $\Theta(q)$ includes all vector pairs such that $o(q_T)$ (calculated as $o(t_C) + o_V \cdot q_T$) is covered by q_R, or formally, $q_{R-}[i] \leq o(q_T)[i] \leq q_{R+}[i]$ on all dimensions $1 \leq i \leq d$. In particular, $\Theta(q)$ is *independent* of the dataset DS, but depends *solely* on the parameters q_R, q_T of q.

Figure 4.1a shows the Venn areas $\Theta(q_1)$, $\Theta(q_2)$ of two queries, which divide the space into 3 disjoint regions that can be represented using different *signatures*. Signature '00' indicates the area outside $\Theta(q_1)$ and $\Theta(q_2)$, corresponding to the set of objects that satisfy neither q_1 nor q_2. Signature '10' ('01') denotes the region inside $\Theta(q_1)$ ($\Theta(q_2)$), covering objects qualifying *only* q_1 (q_2). These 3 signatures are *valid* with respect to q_1 and q_2, while signature '11' is *invalid* because there is no object that can simultaneously satisfy *both* q_1 and q_2 (i.e., $\Theta(q_1)$ does not intersect $\Theta(q_2)$). Figures 4.1b, 4.1c, 4.1d illustrate the other possible relationships between $\Theta(q_1)$ and $\Theta(q_2)$, together with the valid signatures in each case. Specifically, in Figure 4.1c (4.1d), every object that satisfies q_1 (q_2) also qualifies q_2 (q_1), and hence, the signature '10' ('01') is invalid.

(a) $\Theta(q_1)$, $\Theta(q_2)$ are disjoint   (b) $\Theta(q_1)$ overlaps $\Theta(q_2)$

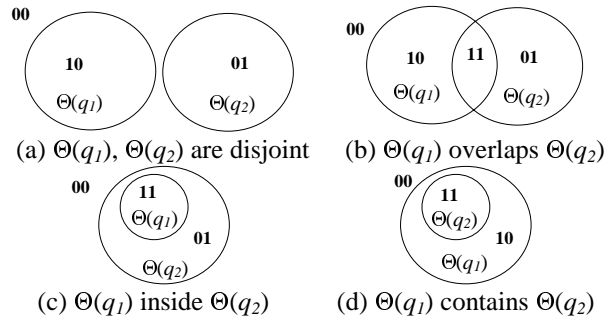(c) $\Theta(q_1)$ inside $\Theta(q_2)$ (d) $\Theta(q_1)$ contains $\Theta(q_2)$
Figure 4.1: Signature examples

In general, given m queries, a signature s is an $m\times1$ bit vector, and the total number of possible (including valid and invalid) signatures is 2^m. If the i-th ($1 \leq i \leq m$) bit of signature s is 1, we say that the i-th query is *active* in s; otherwise the query is *inactive*. For example, in Figure

4.1b, q_1 is active in signatures '10', '11', but inactive in '01'. By the definition of Venn areas, any object p is *covered* by *exactly one* signature s such that p satisfies *only* the active queries of s but not the inactive ones.

VS maintains m sample objects o_1, o_2, …, o_m that are covered by m valid signatures s_1, s_2, …, s_m (decided according to the pivot queries). **S** is a $m \times m$ matrix whose i-th column ($1 \leq i \leq m$) equals s_i. **W** is the $m \times 1$ vector $\{w_1, w_2, …, w_m\}^T$ of the sample weights. Recall that the estimate est_i for the i-th pivot query q_i is the sum of the weights of the samples satisfying it. Thus, the $m \times 1$ vector **EST**$=\{est_1, est_2, …, est_m\}^T$ can be obtained as the product of **S** and **W**:

$$\mathbf{S} \times \mathbf{W} = \mathbf{EST} \qquad (4\text{-}1)$$

Assuming, for example, that $m=4$, the system keeps samples o_1, o_2, o_3, o_4 covered by signatures $s_1=\{0110\}^T$, $s_2=\{1010\}^T$, $s_3=\{0110\}^T$, $s_4=\{1111\}^T$, respectively. Hence, the samples that satisfy pivot query q_1 include o_2, o_4, and thus the estimate est_1 of q_1 equals w_2+w_4. The estimates est_i for other pivots q_i ($1 \leq i \leq 4$) can be derived in a similar manner, leading to the following equation:

$$\mathbf{S} \times \mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} w_2 + w_4 \\ w_1 + w_3 + w_4 \\ w_1 + w_2 + w_3 + w_4 \\ w_4 \end{bmatrix} = \mathbf{EST} \qquad (4\text{-}2)$$

Ideally, **EST** should be equivalent to **ACT**$=\{act_1, act_2, …, act_m\}^T$, in which case, the results of all pivot queries are *precisely* captured. In other words, the "ideal" vector **W** (i.e., the best weights for the samples) should satisfy:

$$\mathbf{S} \times \mathbf{W} = \mathbf{ACT} \qquad (4\text{-}3)$$

The above formula can be regarded as a set of m equations (each row of **S** decides one equation) for m variables (i.e., the m weights in **W**). Therefore, the **W** that achieves zero error for PvQ can be uniquely solved from equation 4-3, provided that matrix **S** is *non-singular* (i.e., no row of **S** can be represented as a linear combination of the other rows). Finding such an **S**, however, is non-trivial due to the fact that the columns of **S** are not arbitrary but confined to the valid signatures (which in turn depend on PvQ). For example, **S** in equation 4-2 *cannot* be used to obtain **W** because its 4th row equals the sum of first two rows minus the 3rd one. In the next section, we provide an algorithm that produces a non-singular **S**, which, interestingly, depends only on PvQ, but is not affected by **W** and **ACT**.

Figure 4.2 illustrates the high-level algorithms of VS. During initialization, VS notifies all objects about PvQ via broadcasting, and then computes the non-singular matrix **S** using algorithm *compute_S* discussed in Section 4.2. Next, for each signature s_i ($1 \leq i \leq m$) in **S**, it obtains a sample o_i through algorithm *find_samples* discussed in Section 4.3. Both *compute_S* and *find_samples* need to be invoked *only once*[2] during the entire execution of VS.

---

[2] These two algorithms are invoked again only when PvQ changes, which as mentioned earlier, happens rather infrequently.

After the initialization, **S** is flushed to the disk; the samples, as well as vectors **W**, **ACT**, **EST** (all initialized to 0), are retained in memory.

The *quality control* component (in the architecture of Figure 3.1) continuously evaluates the estimation error $WE(PvQ)$ (equation 3-1) for the pivot queries, according to the current **EST** and **ACT** (whose maintenance is discussed in Section 3). If the error is larger than a specified threshold ε, it means that the sample weights **W** can no longer accurately capture PvQ. In this case, **W** is re-computed by solving equation 4-3. The matrix **S** is fetched from the disk to perform the re-computation. Since the new **W** leads to exact estimation of **ACT**, vector **EST** is simply updated to **ACT**, and does not need to be modified again until the next re-computation of **W** (recall that **EST** depends on only **S** and **W**, as in equation 4-1).

Algorithm *VS_initialization* (m, PvQ)
//PvQ contains m pivot queries;
1. broadcast PvQ to all objects
2. invoke ***compute_S*** (m, PvQ) to obtain matrix **S**
3. for each column s_i of **S** //s_i is a valid signature
4. o_i=***find_sample*** (s_i, m, PvQ) //decide the sample
5. save **S** to disk
6. **W**=0; **ACT**=0; **EST**=0
End *VS_initialization*

Algorithm *VS_quality_control* (PvQ, ε)
//ε is the quality threshold
1. while (true)
2.   compute $WE(PvQ)$ by equation 3-1
3.   if $WE(PvQ) > \varepsilon$
4.   load **S** from disk, and solve **W** from equation 4-3
5.   set **EST** to **ACT**
End *VS_quality_control*
Figure 4.2: Algorithms of Venn sampling

Another remark concerns the storage of matrix **S**. Due to its relatively large size $O(m^2)$, we do not keep it in memory but load it whenever **W** is re-computed. The total cost of loading **S** is small because (i) as shown in the experiments, re-computation of **W** is necessary only after a long period of time (since the last computation), and (ii) **S** can be stored sequentially so that no random I/O access occurs during its loading. Moreover, since **S** consists of only "0" and "1", it can be effectively compressed (in a lossless way) using, for example, *wavelets* [DR03]. Although such compression slightly increases the CPU overhead, it reduces significantly the I/O cost of reading **S**, thus shortening the overall access time.

Before proceeding, we provide an important observation: *to achieve perfect estimation for all pivot queries, the sample size must be at least m*. In general, if the sample size is k, then matrix **S** in equation 4-3 has k rows, which offer k equations to solve the m weights of **W**. If $k<m$, the equation-set is "over-determined" (i.e., fewer equations than variables), in which case no exact **W** can possibly exist. Further, our subsequent discussion shows that the sample size needs to be exactly m to capture PvQ (i.e., VS requires the smallest sample size).

4.2 Finding non-singular matrix S

In this section, we discuss how to find a ($m \times m$) matrix **S**, whose columns are composed of valid signatures (formulated according to *PvQ*), and at the same time, its rows are mutually linearly independent. As a first step, we aim at identifying a *feasible ordering* $\{q_1', q_2', …, q_m'\}$ of the pivot queries, such that the Venn area of the *i*-th ($1 \le i \le m$) query q_i' is *not* totally covered by the union of the Venn areas of the preceding ($i-1$) queries. Equivalently, this means that for every $1 \le i \le m$, there exists some moving point that satisfies q_i', but does not qualify any of $q_1', q_2', …, q_{i-1}'$. As an example, Figure 4.3a shows the Venn diagram for *m*=3 queries. Ordering $\{q_1, q_2, q_3\}$ is not feasible because the Venn area $\Theta(q_3)$ of q_3 is covered by $\Theta(q_1) \cup \Theta(q_2)$. The feasible orderings in this case are: $\{q_2, q_3, q_1\}$, $\{q_3, q_2, q_1\}$, or $\{q_3, q_1, q_2\}$.
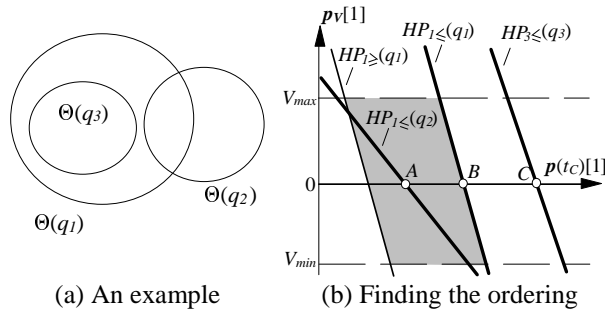

(a) An example   (b) Finding the ordering
Figure 4.3: Illustration of feasible ordering

Figure 4.4 presents a method *find_ ordering* that produces a feasible ordering for any pivot set *PvQ*. To illustrate the idea, recall that a moving point *p* satisfies a query *q* if and only if $q_{R-}[i] \le p(q_T+t_C)[i] = p(t_C)[i] + p_V[i] \cdot q_T \le q_{R+}[i]$ on all dimensions $1 \le i \le d$. Without loss of generality, let us focus on the first dimension *i*=1, and consider a 2D space whose x- (y-) axis corresponds to $p(t_C)[1]$ ($p_V[1]$). Thus, for an object *p* to qualify *q*, its ($p(t_C)[1]$, $p_V[1]$) must fall in the intersection of two half-planes $HP_{1\le}(q)$: $p(t_C)[1] + p_V[1] \cdot q_T \le q_{R+}[1]$ and $HP_{1\ge}(q)$: $p(t_C)[1] + p_V[1] \cdot q_T \ge q_{R-}[1]$ (treating q_T, $q_{R+}[1]$, $q_{R-}[1]$ as constants). The shaded area of Figure 4.3b illustrates such intersection for query q_1 (recall that $p_V[1]$ must also be in the range [V_{min}, V_{max}]).

Algorithm *find_ordering* (*m*, *PvQ*)
//*PvQ* contains *m* pivot queries
1. for each query q_i ($1 \le i \le m$) in *PvQ*
2.   get the intersection p_i between $HP_{1\le}(q_i)$ and $p_V[1]$=0
3. sort p_i in ascending order of their $p(t_C)[1]$; let $\{p_1', p_2', …, p_m'\}$ be the sorted order
4. return $\{q_1', q_2', …, q_m'\}$ (q_i' is the query that produced p_i')
End *find_ordering*

Figure 4.4: Algorithm *find_ordering*

Find_ordering considers the half-planes $HP_{1\le}(q_i)$ of all pivot queries q_i ($1 \le i \le m$). Specifically, it computes the intersection between $HP_{1\le}(q_i)$ with the x-axis, and sorts the intersection points in ascending order of their x-coordinates. In Figure 4.3b (where *m*=3; for simplicity

$HP_{1\ge}(q_2)$ and $HP_{1\ge}(q_3)$ are omitted), the sorted order is $\{A, B, C\}$, and thus *find_ordering* returns $\{q_2, q_1, q_3\}$ whose half-planes $HP_{1\le}(q)$ produce points *A*, *B*, *C*. To see that this ordering is indeed feasible, note that the Venn area $\Theta(q_1)$ cannot be covered by $\Theta(q_2)$, since there exists an object (corresponding to point *B*) that satisfies q_1 but not q_2. Similarly, $\Theta(q_3)$ is not covered by $\Theta(q_1) \cup \Theta(q_2)$ due to point *C*.

Given a feasible ordering $\{q_1', q_2', …, q_m'\}$ of *PvQ*, we compute matrix **S** using an inductive approach. To illustrate the approach, let $\mathbf{S}_i$ ($2 \le i \le m$) be a non-singular $i \times i$ matrix whose columns are valid signatures according to the first *i* queries $q_1', q_2', …, q_i'$ of the ordering. Note that $\mathbf{S}_m$ is exactly the matrix **S** to be computed. Our algorithm first obtains $\mathbf{S}_2$ (i.e., by considering only q_1', q_2'). Then, based on $\mathbf{S}_i$ ($2 \le i \le m-1$), we obtain $\mathbf{S}_{i+1}$ by incorporating the ($i+1$)-st query q_i'. This process is repeated until $\mathbf{S}_m$ has been constructed. The computation of $\mathbf{S}_2$ is trivial. As shown in Figure 4.1, for any relationship between the Venn areas of q_1', q_2', we can always find two valid signatures to build the targeted $\mathbf{S}_2$. For example, in Figure 4.1c the signatures $\{11\}^T$ and $\{10\}^T$ are valid and the resulting $\mathbf{S}_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ is indeed non-singular. Next we discuss the inductive step: calculate $\mathbf{S}_{i+1}$ from $\mathbf{S}_i$ ($2 \le i \le m-1$).

Due to the property of feasible ordering, the Venn area $\Theta(q_{i+1}')$ of the ($i+1$)-th query is not totally covered in the union of $\Theta(q_1'), …, \Theta(q_i')$. It follows that the ($i+1$)×1 vector $\{0…01\}^T$ (i.e., all the bits are 0 except the last one) is valid with respect to the first $i+1$ queries in the ordering (i.e., there exits some moving point satisfying only the ($i+1$)-th query, but not $q_1', …, q_i'$). Based on this observation, we construct $\mathbf{S}_{i+1}$ from $\mathbf{S}_i$ by adding one column and one row as illustrated in Figure 4.5. In particular, the added (i.e., right-most) column is exactly the vector $\{0…01\}^T$.

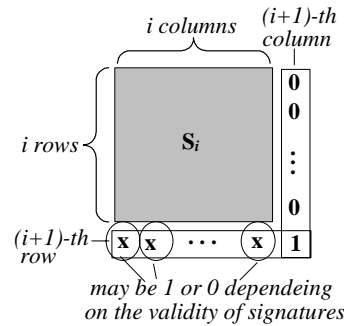**Figure 4.5**: Obtaining $\mathbf{S}_{i+1}$ from $\mathbf{S}_i$

Now it remains to clarify how to decide the ($i+1$)-th row of $\mathbf{S}_{i+1}$. Note that the last bit of this row is already set to 1 by the ($i+1$)-th column $\{0…01\}^T$ we just inserted. For the *j*-th ($1 \le j \le i$) bit (other than the last one), it equals 1 if the resulting *j*-th column (i.e., a signature with respect to q_1', …, q_{i+1}') of $\mathbf{S}_{i+1}$ is valid, and 0 otherwise. We illustrate the process using the feasible ordering $\{q_3, q_1, q_2\}$ in Figure 4.3a. For the first two queries q_3, q_1, we obtain $\mathbf{S}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$

(signatures $\{11\}^T$ and $\{01\}^T$ are valid for q_3, q_1). Based on $\mathbf{S_2}$, $\mathbf{S_3}$ is set to $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$, which is obtained as follows. First, the third column of $\mathbf{S_3}$ is simply $\{001\}^T$. Second, the 1st number in the 3rd row equals 0 because, if it was 1, then the resulting signature (the 1st column of $\mathbf{S_3}$) $\{111\}^T$ would be invalid, i.e., no object can satisfy all 3 queries simultaneously. Finally, the 2nd number in the 3rd row is 1 (the 2nd column $\{011\}^T$ of $\mathbf{S_3}$ is valid).

To conclude, we explain why $\mathbf{S_{i+1}}$ built as in Figure 4.4 is always non-singular, given that $\mathbf{S_i}$ is non-singular. Let r_1, r_2, …, r_{i+1} be the $(i+1)$ rows (each being an $1 \times (i+1)$ vector) of $\mathbf{S_{i+1}}$. It suffices to show that no row can be represented as a linear combination of the other, namely, there does not exist a row r_j $(1 \le j \le i+1)$ such that:

$$r_j = c_1 \cdot r_1 + \ldots + c_{j-1} \cdot r_{j-1} + c_{j+1} \cdot r_{j+1} + \ldots + c_{i+1} \cdot r_{i+1} \qquad (4\text{-}4)$$

where $c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_{i+1}$ are constants among which *at least one is not zero*. Obviously, since the last bit of the $(i+1)$-th row r_{i+1} is 1 (while the last bits of the other rows are 0), equation 4-4 never holds for $j=i+1$ (in this case, the last bits of the equation's two sides do not match). Due to the same reason, the equation does not hold either for any $j \ne i+1$ as long as $c_{i+1} \ne 0$ (i.e., the last bit of the right hand side is non-zero, but that on the left is zero). Therefore, if an r_j should satisfy equation 4-4, c_{i+1} must be 0, meaning that r_j would be linearly dependent on $r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_i$ (i.e., the first i rows other than itself), which contradicts the fact that $\mathbf{S_i}$ is non-singular. The absence of such r_j establishes the non-singularity of $\mathbf{S_{i+1}}$. Figure 4.6 formally summarizes the algorithm *compute_S*. Line 8 invokes *find_sample* (discussed in the next section) to decide if a signature is valid (i.e., the answer is positive only if the result of *find_sample* is not empty).

Algorithm *compute_S* (m, PvQ)

//PvQ contains m pivot queries in feasible ordering
1.   if $m=2$
2.   construct $\mathbf{S}$ depending on the relationship of the Venn areas of the two pivot queries (as shown in Figure 4.1)
3.   else // $m>2$
4. PvQ' contains the first $m-1$ queries in PvQ
        let $\mathbf{S_{m-1}}$=*compute_S* $(m-1, PvQ')$
5.   construct $\mathbf{S_m}$ from $\mathbf{S_{m-1}}$ as follows: add $m \times 1$ vector $\{0 \ldots 01\}^T$ as the m-th column, and add $1 \times m$ vector $\{1 \ldots 11\}$ as the m-th row
6.   for $i=1$ to $m-1$
7.   let s_i be the i-th column of $\mathbf{S_m}$
8.   if *find_sample* $(s_i, m, PvQ)=\varnothing$
9.   set the last bit of this column to 0
10. return $\mathbf{S_m}$

End *compute_S*

Figure 4.6: Algorithm *compute_S*

4.3 Computing the samples

A sample o_i does not have to be a "real" object, but it can be *any moving point* covered by s_i. We do not choose the samples from the dataset DS for the following reasons. First, recall that each sample o_i must be covered by the corresponding signature s_i (i.e., o_i satisfies the active queries of s_i, but dissatisfies all its inactive ones). Such an object may not necessarily exist in DS. Second, even if an object p in DS can be selected as the sample o_i for s_i, when the information of p changes, o_i needs to be updated accordingly increasing the maintenance overhead. In the sequel, we explain the general method to calculate the sample o for a signature s (omitting the subscripts for simplicity).

Without loss of generality, assume that s has a active queries, and (thus) $m-a$ inactive ones. Computing o is equivalent to finding a location $o(t_C)$ and velocity o_V (both are d-dimensional vectors) such that (i) $V_{min} \le o_V[i] \le V_{max}$ (*for all* $1 \le i \le d$), (ii) for each active query q, $q_{R-}[i] \le o(t_C)[i] + o_V[i] \cdot q_T \le q_{R+}[i]$ (*for all* $1 \le i \le d$), and (iii) for each inactive query q, *there exists* a dimension i, such that $q_{R-}[i] > o(t_C)[i] + o_V[i] \cdot q_T$, or $o(t_C)[i] + o_V[i] \cdot q_T > q_{R+}[i]$. Specifically, if we regard $o(t_C)[0]$, $o(t_C)[1]$, …, $o(t_C)[d]$, $o_V[0]$, $o_V[1]$, …, $o_V[d]$ as $2d$ variables, then each (active or inactive) query defines $2d$ linear constraints on them. These variables must satisfy all the constraints of an active query, but need to qualify only *an arbitrary constraint* of an inactive query. Hence, the problem of discovering $o(t_C)$ and o_V is an instance of *disjunctive programming*. Our implementation of *find_sample* adopts a variation of the *backtracking* algorithm [B90].

As an example, consider $m=4$ and signature $s=\{1100\}^T$ that has $a=2$ active queries q_1, q_2 (q_3, q_4 are inactive). In the sequel, we denote the $2d$ constraints defined by q_i $(1 \le i \le m)$ as C_{i-1}, C_{i-2}, …, C_{i-2d}, respectively. We first attempt to find a solution o (including the pair $o(t_C)$, o_V) for all the active queries q_1, q_2. Such a solution must qualify all the $4d$ constraints of q_1, q_2, as well as $V_{min} \le o_V[i] \le V_{max}$ (for all $1 \le i \le d$). Let C be the set of all these $(6d)$ constraints. This is a linear programming (LP) problem which can be efficiently solved in time linear to the size of C [BKOS00]. If solution o is not found, *find_sample* terminates and reports the signature as invalid.

If o exists, we add the first constraint C_{3-1} of inactive query q_3 into C, and invoke the LP solver on the current C. If the solution is not found, we remove C_{3-1} from C, insert the next constraint C_{3-2} of q_3 into C, and execute the LP solver again. If no solution is found again, we try the next constraint of q_3 and repeat these steps. If all the constraints have been attempted, and still no solution exists, *find_sample* claims s is invalid, and terminates. Assume, on the other hand, that a solution is found with constraint C_{3-j} (for some j in $[1,2d]$), the algorithm keeps C_{3-j} in C, and continues to check the constraints of the next inactive query q_4. Similar to the examination of q_3, we add each constraint C_{4-j} into C in turn. If a solution o is obtained, then (since we have considered all queries), *find_sample* has found a moving point o that is covered by this signature. Otherwise (no solution is found with all the constraints of q_4), the algorithm backtracks to the

previous inactive query q_3, removes C_{3-j} from C, inserts $C_{3-(j+1)}$, and repeats the process for q_3.

The complete algorithm *find_sample*, shown in Figure 4.7, includes an additional heuristic. In the previous example, for instance, assume that o is a solution of the constraints of q_1, q_2. After adding a constraint C_{3-j} of q_3 into C, we first check if o satisfies C_{3-j}. If yes, o is directly used as a solution to the current C. The LP solver is invoked (to find the solution of C) only if o does not qualify C_{3-j}.

Algorithm *find_sample* (s, m, PvQ)
//PvQ contains m pivot queries; s is a signature
1. $C=\{V_{min} \leq o_V[i] \leq V_{max}\}$ for $1 \leq i \leq d$
2. $C=C \cup \{$the $2d$ constraints of each active query in $PvQ\}$
3. call LP solver to find solution o for C
4. if o does not exist, return $\varnothing$ //s is invalid
5. $o=$***check_inactive***$(|IaQ|, IaQ, o, C)$
      //IaQ is the set of inactive queries, and $|IaQ|$ is its size
10. return o
End *find_sample*

Algorithm *check_inactive* (t, IaQ, o, C)
//IaQ contains t inactive queries; o is the current solution for C
1. if $t=0$ then return o
2. let q be any query in IaQ
      //let $C_1, \ldots, C_{2d}$ be its $2d$ constraints
3. for $j=1$ to $2d$ //each constraint of the inactive query
4. $C=C \cup C_j$
5.   if o satisfies C_j
6. $o=$***check_inactive***$(t-1, IaQ-\{q\}, o, C)$
7.   else
8.   call LP solver to find solution o for C
9.   if o does not exist
10. $C=C-C_j$, and continue //to the next j
11. $o=$***check_inactive***$(t-1, IaQ-\{q\}, o, C)$
12.   if $o \neq \varnothing$ return o else $C=C-C_{-j}$
13. return $\varnothing$
End *check_inactive*

Figure 4.7: Algorithm *find_sample*

In the worst case *find_sample* needs to invoke LP solver $(2d)^{m-a}$ times (i.e., exponential to the number of inactive queries), but the average cost is much lower (i.e., the solution is usually found quickly). It is important to note that, this overhead is off-line, namely, as mentioned earlier *find_sample* is executed *only once in the initialization stage of VS*. On the other hand, query processing requires only checking each computed sample, and hence, is very efficient.

4.4 Discussion

Although our discussion focuses on RA processing on moving objects, Venn sampling can also be applied to "more traditional" scenarios. For example, a record in a relational table with several attributes can be regarded as a "static" multi-dimensional point (i.e., a special moving object). *Selectivity estimation* aims at predicting the number of points in a region q_R, which can be thought of as a range aggregate query without the parameter q_T.

Hence, given a set of pivot queries, VS can be used for selectivity estimation, also supporting updates.

In fact, the VS framework is general, i.e., the algorithms are not specific to moving objects except: (i) *find_ordering* (Figure 4.4), and (ii) *find_sample* (Figure 4.6). The adaptation of VS to alternative problems requires re-designing only these two components (the other algorithms can be used directly). VS is particularly well-suited for streaming applications where real-time processing is imperative. This is because the expensive computations (*compute_S* and *find_sample*) are performed once, during the initialization stage. The on-line costs include merely (i) updating the **ACT** vector, and (ii) solving a set of linear equations occasionally. Furthermore, processing a RA query requires examining only m samples.

Finally, although stratified sampling provides the initial motivation, it is inapplicable to our problem (and in general, any other problem with similar settings). Recall that SS keeps samples for all the signatures covering some objects in DS (each such signature corresponds to a stratum in Section 2.2). Since these signatures may emerge/disappear with data updates, their samples cannot be pre-computed as in VS. For example, assume that the sample o of signature s modifies its information, and is no longer covered by s. The new sample of s cannot be obtained efficiently because (i) the system does not maintain any other non-sampled data (eliminating the option of choosing another data point as the sample), and (ii) applying *find_sample* (Figure 4.6) to compute a hypothetical object is too expensive (recall that the algorithm is suitable only for off-line execution).

5. Experiments

This section empirically evaluates the efficiency of VS, using datasets created in the same way as in [TSP03]. Specifically, each axis of the 2D space is normalized to unit length. 5k points are randomly selected as "airports" from a real 2D point dataset CA/LB with cardinality 130k/56k (http://www.census.gov/geo/www/tiger/). 100k aircrafts move among the airports as follows. For each aircraft p, two random airports are selected as the source *src* and destination *des*, respectively. The initial location of p is a random point on the segment connecting *src* and *des*. The orientation of the segment also determines the velocity direction of p, while the speed uniformly distributes in the range [0, 20]. Aircraft p flies for a number of timestamps uniformly distributed in [0, 5]. After that, p changes speed, moves (with the updated speed) for another period, and so on. As soon as p reaches the destination *des*, it flies towards another destination, repeating the above movements.

The search region q_R of a query is a square with length *qrlen*, and its query timestamp q_T is uniform in [0, T], where T is the farthest future time that can be queried. The pivot workload PvQ contains m queries with the same *qrlen*, and the location of their q_R is decided in a clustered

manner. For this purpose, we first obtain num_{clu} clusters following the airport distribution, where each cluster is a circle with diameter 0.05. The center of each q_R (which, together with $qrlen$, completely decides q_R) follows uniform distribution in a cluster. All clusters are expected to cover the same number of pivot queries. A *user workload* consists of 1k queries whose $qrlen$ is identical to a given PvQ. The concrete q_R and q_T of a user query q_{user} are decided based on a randomly-selected pivot query q_{pivot}. Specifically, the q_R center of q_{user} deviates from that of q_{pivot} by $l\varepsilon$ on each dimension, where $l\varepsilon$ is uniform in $[0, l\varepsilon_{max} \cdot qrlen]$ (i.e., the maximum deviation is a percentage $l\varepsilon_{max}$ of $qrlen$). Similarly, the timestamp q_T of q_{user} differs from that of q_{pivot} by $t\varepsilon$ uniform in $[0, t\varepsilon_{max} \cdot T]$ ($t\varepsilon_{max}$ is also a percentage). Obviously, higher $l\varepsilon_{max}$ ($t\varepsilon_{max}$) produces a user workload "less similar" to PvQ.

We compare VS with the existing approaches for spatio-temporal estimation: *spatio-temporal histograms* (STH) and *random sampling* (RS) reviewed in Section 2. The performance of each method is measured as its estimation error (calculated by equation 3-1) in answering a user workload. In each experiment, we allocate the same amount of space to all techniques. Specifically, if VS keeps m samples (equal to the number of pivot queries), then the number of buckets (samples) in STH (RS) equals $5.6m$ ($1.2m$) (taking into account different storage formats of various methods). All the experiments were performed using a Pentium IV 2.8 GHz CPU with 512 mega-bytes DDR400 memory.

5.1 Effects of clustering in *PvQ*

We first study the effect of m and num_{clu} on the prediction accuracy. These two parameters decide the number of pivot queries in each cluster (i.e., a high m or low num_{clu} results in densely-clustered queries). The other parameters are fixed to $qrlen=0.05$, $T=10$, $l\varepsilon_{max}=10\%$, and $t\varepsilon_{max}=3\%$. First, we fix num_{clu} to 3 and measure the relative error as a function of m. As shown in Figure 5.1 (for both datasets CA and LB), VS outperforms its competitors significantly in all cases. Particularly, RS completely fails yielding up to 140% error. This is expected because RS performs well only if the sample size is sufficiently large (around 10% of the dataset [CDD+01]), while in our case the size is limited to at most 0.36% (for $m=300$) of the database (100k aircrafts). Similarly, STH also incurs considerable error due to the small number of buckets allowed (as shown in [TSP03], 3k buckets are needed to achieve accurate estimation).

The precision of all methods improves as m increases for different reasons. RS (STH) uses a larger number of buckets (samples). The accuracy of VS, on the other hand, depends on the "similarity" between the user workload and PvQ. For large m (i.e., numerous pivot queries in a cluster), the probability that an actual query can be captured by some pivot increases. Specifically, recall that a user query q_{user} is generated by drifting its q_R, q_T parameters from those of a pivot q_{pivot}. For greater m,

there is a higher chance that q_{user} is very close to *another* pivot query (in the same cluster as q_{pivot}), and hence can be accurately predicted. This phenomenon, referred to as *mutual compensation* in the sequel, explains why VS can provide good estimation even if the user workload differs from PvQ.
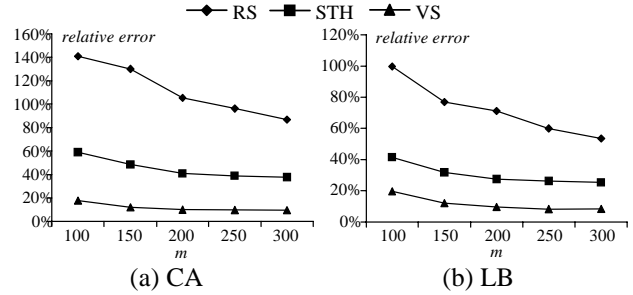

Figure 5.1: Error vs. m ($qrlen=0.05$, $T=10$, $num_{clu}=3$, $l\varepsilon_{max}=10\%$, $t\varepsilon_{max}=3\%$)

To further verify this, Figure 5.2 measures the error by changing the number num_{clu} of clusters, fixing $m=200$. As num_{clu} grows, the number of pivot queries in a cluster drops, thus weakening the effect of mutual compensation, and hence, lowering the VS precision. Nevertheless, VS is significantly better than RS and STH for all tested values of num_{clu}.
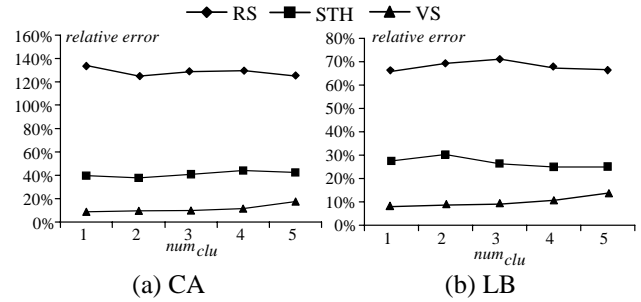

Figure 5.2: Error vs. num_{clu} ($m=200$, $qrlen=0.05$, $T=10$, $l\varepsilon_{max}=10\%$, $t\varepsilon_{max}=3\%$)

5.2 Effects of query parameters

In the sequel, we set $m=200$ and $num_{clu}=3$, and compare alternative methods with respect to parameters $qrlen$ and T (which determine q_R and q_T, respectively). Fixing $T=10$, Figure 5.3 plots the error as $qrlen$ varies from 0.01 to 0.09 (i.e., q_R covers up to 0.81% of the data space). All methods become more accurate for larger $qrlen$ (with VS still being the best method), which is consistent with the well-known fact that probabilistic estimation approaches, in general, perform better for larger query results [AGP00]. In Figure 5.4, $qrlen$ is set to 0.05, and we evaluate the methods using different T. The accuracy of RS and STH is hardly affected by this parameter (confirming the results in [HKT03, TSP03]). VS deteriorates because, for higher T, the pivot queries are less similar to each other due to the greater difference in their q_T (recall that q_T uniformly distributes in $[0, T]$). Lower similarity between pivot queries reduces the effect

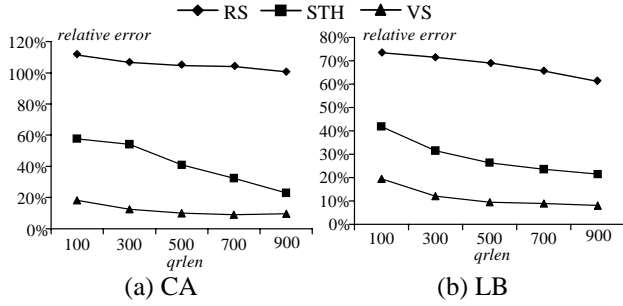of mutual compensation, thus leading to higher error.

Figure 5.3: Error vs. *qrlen* ($m=200$, $T=10$, $num_{clu}=3$, $l\varepsilon_{max}=10\%$, $t\varepsilon_{max}=3\%$)
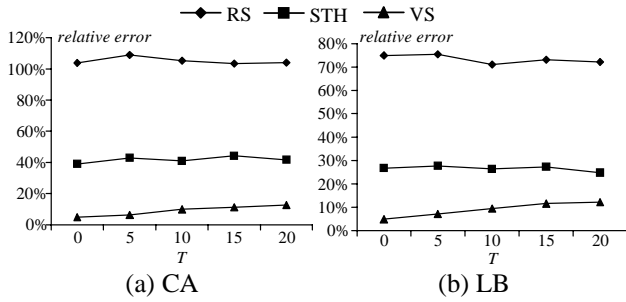

Figure 5.4: Error vs. *T* ($m=200$, $qrlen=0.05$, $num_{clu}=3$, $l\varepsilon_{max}=10\%$, $t\varepsilon_{max}=3\%$)

5.3 Effects of user workload similarity

Next we set *m*, *qrlen*, *T*, num_{clu} to their median values 200, 0.05, 10, 3, respectively, and study the estimation efficiency by varying the similarity between user queries and *PvQ*. Figure 5.5 (5.6) fixes $t\varepsilon_{max}=3\%$ ($l\varepsilon_{max}=10\%$), and illustrates the error as $l\varepsilon_{max}$ ($t\varepsilon_{max}$) increases, adjusting the difference between user/pivot workloads in search regions (query timestamps). Evidently, VS performs satisfactorily even with the largest $l\varepsilon_{max}$ and $t\varepsilon_{max}$ (i.e., the lowest user similarity). In particular, VS yields maximum error below 20%, and is considerably more accurate than RS and STH (which are not affected by $l\varepsilon_{max}$ and $t\varepsilon_{max}$).
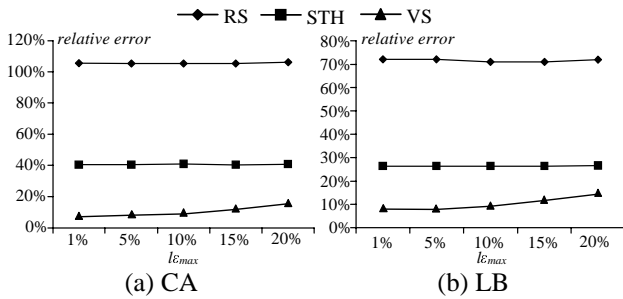

Figure 5.5: Error vs. $l\varepsilon_{max}$ ($m=200$, $qrlen=0.05$, $T=10$, $num_{clu}=3$, $t\varepsilon_{max}=3\%$)

5.4 Deterioration with time

Having evaluated the estimation accuracy on a single timestamp, we proceed to investigate the performance of each method as time progresses. Towards this, we generate aircraft movements for 1k timestamps, and

measure the accuracy of RS, STH, VS for the same user workload every 10 timestamps. For VS, we re-compute the weight vector (**W** in equation 4-3) whenever the error for *PvQ* is above 10%. Each re-computation requires less than 1ms. As shown in Figure 5.7, the precision of STH continuously deteriorates as time evolves, for the reasons discussed in Section 2.1. The performance of RS and VS is stable and eventually VS outperforms the other approaches by more than an order of magnitude. Interestingly, the error of VS changes in a "jigsaw" manner. The "ascending" parts (of the jigsaw) correspond to the slow accuracy degradation of VS (due to the data distribution changes) between consecutive weight re-computations, while each "descending" part indicates the (immediate) improvement of re-computation. Note that re-computation is performed infrequently (on the average, every 200 timestamps).
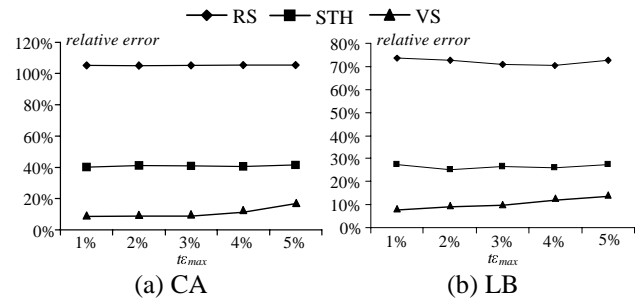

Figure 5.6: Error vs. $t\varepsilon_{max}$ ($m=200$, $qrlen=0.05$, $T=10$, $num_{clu}=3$, $l\varepsilon_{max}=10\%$)
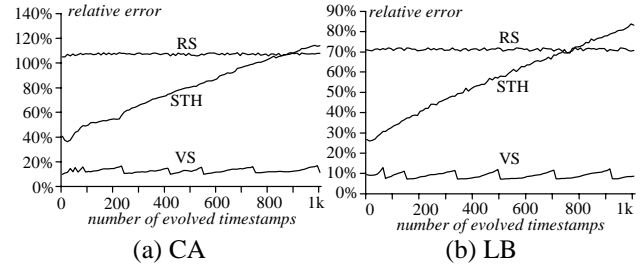

Figure 5.7: Accuracy degradation with time ($m=200$, $qrlen=0.05$, $T=10$, $num_{clu}=3$, $l\varepsilon_{max}=10\%$, $t\varepsilon_{max}=3\%$)

5.5 Processing costs

The last set of experiments evaluates the overhead of query estimation, dynamic maintenance, and pre-computation. We only illustrate the results of CA dataset since those of LB are almost identical. Figure 5.8a (5.8b) compares the query processing time of each technique as a function of *qrlen* (*T*). STH is by far the slowest method because, as mentioned in Section 2.1, it entails expensive numerical evaluation. VS outperforms RS due to its smaller sample size. Figure 5.9 illustrates the number of update messages for dynamically maintaining each method in the experiments of Figure 5.7. VS requires considerably fewer messages than STH, confirming the superiority of data-driven updates over conventional (velocity-driven) policies. RS incurs even fewer messages

because (i) an update is necessary only when a sample object changes its velocity, and (ii) the sample size is very small. However, the update efficiency of RS does not justify its poor estimation accuracy as shown in the previous sections. Finally, Figure 5.10 shows the pre-computation time of VS as a function of the number m of pivot queries. Recall that this overhead is off-line and incurs only once.
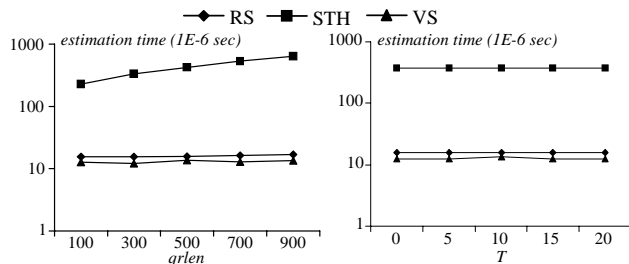
(a) Cost vs. *qrlen* (*T*=10)   (b) Cost vs. *T* (*qrlen*=500)
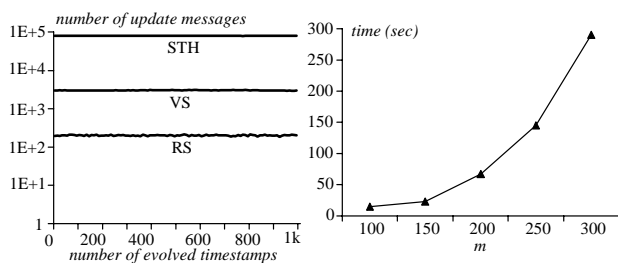**Figure 5.8**: Query estimation time (CA)

Figure 5.9: Message   **Figure 5.10**: Off-line
overhead for updates (CA) initialization cost of VS (CA)

6. Conclusions

We propose Venn sampling, a spatio-temporal estimation technique that outperforms the existing solutions on all aspects including space consumption, evaluation efficiency, robustness with time and maintenance overhead. This paper also opens several exciting directions for future work. The first interesting question is, although VS already achieves perfect pivot estimation using as many samples as the number m of pivot queries, can its performance for *actual queries* be further improved if more space is allowed? An inverse version of the problem is: if the available space is less than $O(m)$ (i.e., precise pivot prediction is impossible), how can we select the "best" signatures to minimize error? Furthermore, it would also be very useful to investigate the performance of VS in other applications (such as selectivity estimation in relational databases) and compare it with alternative techniques for the same problem.

Acknowledgemnts

References

[AA03]   Aggarwal, C., Agrawal, D. On Nearest Neighbor Indexing of Nonlinear Trajectories. *PODS*, 2003.

[AAE00]   Agarwal, P., Arge, L., Erickson, J. Indexing Moving Points. *PODS*, 2000.

[AGP00]   Acharya, S., Gibbons, P., Poosala, V. Congressional Samples for Approximate Answering of Group-by Queries. *SIGMOD*, 2000.

[B90]   Beaumont, N. An Algorithm for Disjunctive Programming. *EJOR*, 48: 362-371, 1990.

[BGC01]   Bruno, N., Gravano, L., Chaudhuri, S. STHoles: A Workload Aware Multidimensional Histogram. *SIGMOD*, 2001.

[BKOS00] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O. Computational Geometry: Algorithms and Applications (second edition). *Springer*, ISBN 3-540-65620-0, 2000.

[CC02]   Choi, Y., Chung, C. Selectivity Estimation for Spatio-Temporal Queries to Moving Objects. *SIGMOD*, 2002.

[CDD+01] Chaudhuri, S., Das, G., Datar, M., Motwani, R., Narasayya, V. Overcoming Limitations of Sampling for Aggregation Queries. *ICDE*, 2001.

[CDN01]   Chaudhuri, S., Das, G., Narasayya. A Robust Optimization-Based Approach for Approximate Answering of Aggregate Queries. *SIGMOD*, 2001.

[DGR03]   Das, A., Gehrke, J., Riedewald, M. Approximate Join Processing Over Data Streams. *SIGMOD*, 2003.

[DR03]   Deligiannakis, A., Roussopoulos, N. Extended Wavelets for Multiple Measures. *SIGMOD*, 2003

[GLR00]   Ganti, V., Lee, M., Ramakrishnan. ICICLES: Self-tuning Samples for Approximate Query Answering. *VLDB*, 2000.

[GM98]   Gibbons, P., Mattias, Y. New Sampling-based Summary Statistics for Improving Approximate Query Answers. *SIGMOD*, 1998

[GMP97]   Gibbons, P., Mattias, Y., Poosala, V. Fast Incremental Maintenance of Approximate Histograms. *VLDB*, 1997.

[HKT03]   Hadjieleftheriou, M., Kollios, G., Tsotras, V. Performance Evaluation of Spatio-temporal Selectivity Estimation Techniques, *SSDBM*, 2003.

[J03]   Jermaine, C. Making Sampling Robust with APA. *VLDB*, 2003.

[L00]   Lynch, J. Analysis and Application of Adaptive Sampling. *PODS*, 2000.

[LNS90]   Lipton, R., Naughton, J., Schneider, D. Practical Selectivity Estimation through Adaptive Sampling. *SIGMOD*, 1990.

[SJLL00] Saltenis, S., Jensen, C., Leutenegger, S., Lopez, M. Indexing the Positions of Continuously Moving Objects. *SIGMOD*, 2000.

[TPS03]   Tao, Y., Papadias, D., Sun, J. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. *VLDB*, 2003.

[TSP03]   Tao, Y., Sun, J., Papadias, D. Analysis of Predictive Spatio-Temporal Queries. *ACM TODS,* 28(4): 295-336, 2003.

[V85]   Vitter, J. Random Sampling with a Reservoir. *ACM TOMS*, 11: 37-57, 1985.

[WAA01]   Wu, Y., Agrawal, D., Abbadi, A. Applying the Golden Rule of Sampling for Query Estimation. *SIGMOD*, 2001.