Aggregation Estimation for 2D Moving Points*

Scot Anderson
University of Nebraska
Lincoln, Nebraska 68588
scot@cse.unl.edu

Abstract

The new Max-Count *aggregation operator presented finds the maximum number of 2D moving points that overlap a moving query rectangle within a specified time interval. Two linearly moving points define the query rectangle which may move, and grow or shrink over time. We give the first* constant running-time *estimation algorithm to solve the* Max-Count *aggregation problem in two dimensions. The estimation algorithm uses 4-dimensional buckets to index objects. The implementation of the* Max-Count *estimation algorithm and the experimental data show that the method exhibits good accuracy and speed.*

1. Introduction

Advances in GIS, embedded systems, and GPS technology with the ability to track moving objects have increased the interest in spatiotemporal databases. Aggregation operators pose new challenges for moving objects and spatiotemporal databases. Recently, Revesz and Chen [24] pointed out that new aggregate operations such as *Max-Count* can be defined on those databases.

Definition 1 Let S be a set of moving points in the plane. Given a moving rectangle R, defined by two moving points Q_1 and Q_2 as the lower-left and upper-right corners of R, and a time interval T, the **Max-Count** operator finds the maximum number of points in S that R can contain at any time instance within T.

Consider the following example where the research space probes, Pioneer 10 and 11, traversed the main astroid belt located between Mars and Jupiter.

Example 1 The astroid belt is a donut shaped area that is approximately 280 million km wide and 80 million km thick. The material in the belts travel in orbit around the sun at speeds up to 20 km/sec and range in size from dust particles to rock chunks as big as Alaska. Traversing this belt can be a dangerous exercise and had to be done by both the Pioneer 10 and 11 space probes. Pre-mission planning took the form of shooting for perceived holes in the asteroid belt. Knowing the maximum speed of objects in the asteroid belt, a spatiotemporal system with the *Max-Count* aggregate operator could scan for objects not visible at the beginning of a mission and predict the maximum number of objects that must be traversed in a path through the belt allowing the mission to conserve fuel and maintain safety. Since objects in the belt travel in orbit around the Sun, it is reasonable to approximate them as moving in two dimensions with linear motion for the small region traversed by the spacecraft. Let

$$Asteroid(Id, X, Y, T) \tag{1}$$

contain moving asteroids in the region that must be traversed by the space probe where X and Y give the location at time T. Let

$$SpaceProbe(X, Y, T) \tag{2}$$

be an area around the space probe at time T where the number of objects must not exceed a threshold. One may try to implement in SQL a *Max-Count* aggregation by first applying a *Count* aggregation with grouping on the time instances t, and then following it by a *Max* aggregation as follows:

```
select max(objCnt)
from (select count(Asteroid.Id) as objCnt
      from   Asteroid, SpaceProbe
      where  Asteroid.X = SpaceProbe.X and
             Asteroid.Y = SpaceProbe.Y and
             Asteroid.T = SpaceProbe.T and
             start_time <= T and
             T <= end_time
      group-by T);
```

However, the above does not work in relational databases for two reasons. First, $Asteroid$ and $SpaceProbe$ would

*This research was supported in part by NSF grant EIA-0091530 and a NASA Space and EPSCoR grant. The author would like to thank Peter Revesz for helpful comments about this research work.

require infinite relations, which are not allowed in relational databases. Second, there would be an infinite number of groups, because t is a continuous variable. That is also not allowed in any implementation of the Count aggregation operator in relational databases.

We can solve the first problem by using a finite representation of the infinite relations $Asteroid$ and $SpaceProbe$. One option is to use a constraint database [9, 21, 26], such as the MLPQ or DEDALE system. Constraint databases provide convenient finite representations for linearly moving objects [3, 8, 7]. However, even in constraint databases, the group-by clause cannot generate an infinite number of groups in any current implementation.

In this paper we focus on linearly moving objects which can approximate motion using piecewise linear parts [13, 23, 27].

1.1. Main Result

As Example 1 shows, we must implement the *Max-Count* aggregation operator in a special way. We present an efficient implementation that *estimates* the answer in some fixed constant time. As the constant is increased, the accuracy of the estimation improves. Our implementation shows that the constant need not be very large to have a high accuracy.

1.2. Previous Work

The *Max-Count* aggregation was introduced recently by Chen and Revesz [24, 5]. These two papers deal with the *one-dimensional* case for *Max-Count*. The first gave an $O(\log N)$ time and $O(N^2)$ space solution, where N is the number of moving points. The second gave a constant-time estimation for the one-dimensional case. Revesz [22] gave an ECDF-B-Tree method to estimate the Count aggregation value in $M \log N$ time where M is a parameter affecting the accuracy. Table 1 compares the results of these earlier papers with our current result.

Dim.	**Time**	**Space**	**Type**	**Ref.**
1	$O(log\ N)$	$O(N^2)$	Exact	[24]
k	$O(M \log N)$	$O(N \log N)$	Est.	[22]
1	$O(1)$	$O(1)$	Est.	[5]
2	$O(1)$	$O(1)$	Est.	current

Table 1. Max-Count aggregation complexity

Our extension of the *Max-Count* aggregation to the 2-dimensional case requires a higher mathematical complexity and has a greater practical potential as shown in Example 1.

Our work is related to previous work on indexing structures, count aggregation, and estimation techniques as follows.

Indexing structures, Range and Count aggregation: Example 1 shows that the *Count* and *Max* aggregation operators have only a titular relationship to the *max-count* aggregation. Nevertheless, several techniques used in the max-count problem are also used in other indices and algorithms designed for range, max/min, and count queries. We summarize several of these related techniques.

The index structure of Kollios et al. [15] finds the 1-dimensional moving points contained in a window in expected logarithmic time. The index structure of Agarwal et al. [2] finds the 2-dimensional moving points contained in a rectangle in $O\left(\sqrt{N}\right)$ time. Gunopulos et al. [11, 10] give a histogram structure designed to approximate the density of multi-dimensional data sets that allows buckets to overlap. Papadopoulos et al. [20] give an index structure for range queries over 2-dimensional moving points. Gupta et al. [12] gave a technique for answering spatiotemporal queries on 2-dimensional objects that move along curves. Zhang et al. [34] propose the *multiversion SB-tree* to perform range temporal aggregates: SUM, COUNT and AVG.

Cai and Revesz [4] and Saltenis et al. [28] gave two different R*-tree based index techniques for moving objects. Tao et al. [29] proposed a new index structure for predictive *Count* aggregation queries for 2-dimensional moving points. Tayeb et al. [31] adapt a variant of the quadtree structure to solve the problem of indexing dynamic attributes. Lin et al. [18] gave an index structure for past, present, and predictive positions of moving points. Mokhtar et al. [19] use computational geometry to evaluate past, present, and future positions of moving objects. Finally Hadjieleftheriou et al. [14] address density-based queries in spatiotemporal databases.

Each of these indexing methods that return the moving points in a query window or rectangle can be easily modified to return instead the *Count* of the number of moving points.

Count Estimation: Our work is related to several other papers that estimate the count aggregate operation on spatiotemporal databases. Acharya et al. [1] gave an algorithm that can estimate the *Count* of the number of the rectangles that intersect a query rectangle for *selectivity estimation*. Choi and Chung [6] and Tao et al. [30] proposed methods that can estimate the *Count* of the moving points that intersect a query rectangle. Most of these estimation algorithms use *buckets* as basic building structures of the index. We extend this idea by using 4-dimensional hyper-buckets in our algorithms.

Object Position Evaluation: Kollios et al. [16] gave a predictive method based on dual transformations. Li and Revesz [17] and Revesz and Wu [25] gave a method for interpolating spatiotemporal data. Wolfson et al [33, 32] gave a method for generating pseudo trajectories of moving objects.

1.3. Organization

The rest of the paper is organized as follows. In Section 2 we introduce the 4-dimensional index structure and describe the algorithm used to generate it. In Section 3 we give the algorithm that estimates the *Max-Count* aggregate operator with 2-dimensional linearly moving points. Section 4 describes the implementation and experimental results of the estimation algorithm. Finally, Section 5 presents conclusions and future work.

2. 4-Dimensional Spatial Index

Indexing linearly moving points into buckets requires a mapping of those points into a static representation. This allows an easier classification and examination of the linearly moving points. In Section 3 this particular static index structure allows us to discover spatial relationships between points or groups of points relative to time.

Given a set of 2-dimensional moving points in the plane, we map the points into a dual 4-dimensional space using the following definition.

Definition 2 Let a moving point O be described by a set of linear equations of the form:

$$O(t) = \begin{cases} x = v_x t + x_0 \\ y = v_y t + y_0 \end{cases} \quad (3)$$

O is represented in the 4-dimensional dual space by the following static point O' called the *quad-representation* of O.

$$O' = (v_x, x_0, v_y, y_0) \quad (4)$$

Our index algorithm partitions the 4-dimensional input grid from the index algorithm into *hyper-buckets* such that the point density in each hyper-bucket is as uniform as possible. The density of the buckets form a histogram that is used in the *Max-Count* aggregation operator. Hence the index algorithm affects the *Max-Count* aggregation operator in two ways. First, the index algorithm calculates part of the approximation, thus accurate estimation in the *Max-Count* algorithm implicitly depends on the accuracy of the index algorithm to create uniform hyper-buckets. Second, the number of buckets required to meet a specified uniformity impacts the *constant* used for the number of buckets in the *Max-Count* algorithm. Therefore, increasing the accuracy while maintaining the bucket count (or maintaining the accuracy while lowering the bucket count) will generally improve the performance of the *Max-Count* algorithm.

```
Hyper-Bucket-Indexing($S$,$c$,$n$,$H$)
input: $S$ the set of 4-dimensional points to be indexed,
    $c$ Cell size, and $n$ number of hyper-buckets.
output: The histogram $H$.
Place the points into cells
$H$ gets all the cells in the 4-dimensional space.
while $H$ has less than $n$ buckets
    for each bucket $B_i$ in $H$ do
        Compute the spatial-skew of $B_i$ and
        find the split point along its dimensions
        that will produce the maximum reduction
        in spatial-skew
    end for
    Pick the bucket $B$ whose split will lead to the
    greatest reduction in spatial-skew.
    Split $B$ into two buckets $B_1$ and $B_2$ and assign
    regions from $B$ to $B_1$ and $B_2$.
end while
Assign each cell in the input to the hyper-bucket
whose Minimum bounding rectangle contains
the center of the rectangle
```

Figure 1. Hyper-bucket indexing algorithm.

To form hyper-buckets, we first overlay the 4-dimensional space with a hyper-grid. Each hypercube in the hyper-grid is assigned a value representing the number of points contained within it. We refer to the hyper-cubes with their density as *cells*. We can choose small enough cells such that each cell will have a uniform distribution of points. Second, we form the hyper-buckets by grouping together some of the adjacent cells. To explain our grouping, we first define the *spatial-skew*, extending a definition in [1], as follows.

Definition 3 Consider a grouping $\mathcal{G}$ with hyper-buckets B_i where $1 \leq i \leq \beta$. Let n_i be the number of cells in B_i. The spatial-skew s_i of a bucket B_i is the statistical variance of the spatial densities of all cells grouped within that bucket. The spatial-skew S of the entire grouping is the weighted sum of spatial-skews of all the hyper-buckets $\sum_1^\beta n_i \times s_i$.

This hyper-grid determines the first approximation for the buckets. Given this input structure we build the hyper-buckets using the algorithm in Figure 1. The hyper-buckets form a histogram that is the input to the *Max-Count* algorithm.

3. Max-Count over 2-Dimensional Linear Motion

Point domination plays a pivotal role in determining count as a continuous function of time. Given that continuous function we can maximize it to determine the *Max-Count* over time. The function will determine what fraction of a hyper-bucket is contained in the query space. Hence the function will return a hyper-volume from which we can calculate the estimated count during a time interval that a hyper-bucket is in the query space. Several time intervals may need to be examined to determine the final *Max-Count*

Point domination in one dimension is a simple concept. Given moving points $Q(t) = vt + d$ and $P(t) = xt + y$, we say that Q dominates P if and only if $P(t) < Q(t)$. This gives the following condition:

$$y < -t(x - v) + d \tag{5}$$

The slope of the line through the query point gives a spatial relationship dependent on time. When comparing several 1-dimensional points P_i, $1 \leq i \leq n$, to a query point Q, the visualization is a 2-dimensional graph where points are dominated if they lie below the line. For 2-dimensional linearly moving points, we have a 4-dimensional space and this verification must be done in two independent steps because there is no analogous value to slope in a hyper-plane that could be used. This requires a change to a different kind of visualization since it is difficult to visualize 4-dimensional space. Below we describe the functions visualized as two 2-dimensional dual spaces that form the useful continuous function of time needed. Next we give a formal definition to the problem and then develop the algorithm.

Problem Description: Given a set of 2-dimensional linearly moving points S, two query points (Q_1, Q_2), and a time interval $[t_1, t_2]$, the *Max-Count* aggregation returns an estimated count $c_{\max}$ and time $t_{\max} \in [t_1, t_2]$ where each moving point is represented in the form of Equation 3.

Let B be the set of 4-dimensional hyper-buckets in the input, and let each hyper-bucket have an associated density d_i and corner vertices $v_{i,j}$ for $1 \leq j \leq 16$. Given these hyper-buckets and query points, we define equations to determine which hyper-buckets or portions of hyper-buckets are dominated by a query point. By assuming that the density of points in the hyper-buckets is uniform, we can maximize the number of points in discrete hyper-regions separated by the vertices $v_{i,j}$.

Definition 4 Let Q and P be two moving points. Project their quad-representations of the form (v_x, x_0, v_y, y_0) into an x-view plane (v_x, x_0) and a y-view plane (v_y, y_0). For 2-dimensional moving points, Q dominates P if and only if Q dominates P in both the x-view and the y-view.

Although this definition seems that it separates our 4-dimensional space into 2-dimensional spaces, this is actually not the case. It is a convenience to treat dominance in each of the two original dimensions separately as we will see. Decomposing the views into orthogonal views leads to the following definition for the *query space*

Definition 5 Given two moving query points $Q_1(t)$, $Q_2(t)$ and lines $l_{x1}, l_{x2}, l_{y1}, l_{y2}$[1] crossing them in their respective duals with slopes $-t$, the hyper-volume formed by the *intersection* of the bands of the area between l_{x1} and l_{x2} and the area between l_{y1} and l_{y2} is a hyper-tunnel that is called the **query space**.

Using the above, it is now easy to prove the following lemma.

Lemma 1 At any time t the moving points whose quad representation lies left and below (or right and above) l_{x1} and l_{y1} in their respective views are exactly those points that lie to the left (or right) and below (or above) Q_1 in the original 2-dimensional plane. ■

During the times that the query space contains the same corner vertices we can find an equation for the area of the form:

$$A_i(t) = a_i t + \frac{b_i}{t} + c_i \tag{6}$$

The form of Equation 6 comes from the possible 3 to 6-sided polygon shapes that can be formed by the intersection of the query lines with bucket B_i in the $x-$view or $y-$view [5]. As the lines sweep through the hyper-bucket there will be several time intervals that have different shapes and thus different equations. Let $T = [t^{[}, t_1], ..., [t_c, t^{]}])$ be those time intervals. A_i is the area during the time interval starting at t_i and $a_i, b_i, c_i \in \mathcal{R}$. As the line through Q_1 sweeps through the query space each vertex encountered determines an end point for the time intervals. The estimated Max-Count for each time interval is calculated as follows. Given the density d_i of bucket $B_i \in B$, set:

$$\frac{dA_i}{dt} = 0 \tag{7}$$

and solve for t. This is either a maximum or a minimum. We can find the maximum area for $[t_i, t_{i+1}]$ by plugging in the end points and the times from the solution to Equation 7. The process of sweeping through a hyper-bucket is shown in Figure 2. Here time is increasing as we sweep counter clockwise. If we went backward in time from t_i (sweep clockwise from the position of the solid lines) we would eventually encounter a vertex of this bucket and we would have to initiate a new time interval.

[1]The x and y subscripts denote the view to which the lines belong.

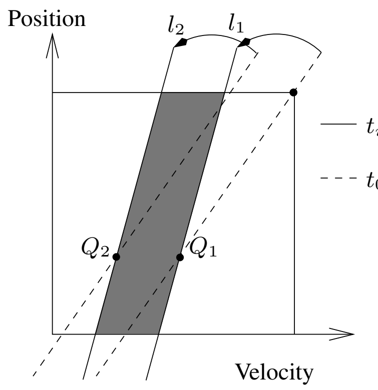

Figure 2. 1-dimensional Bucket Time Sweep

We can construct the 4-dimensional query space mathematically as follows.

Lemma 2 If the query bands in *both* the (orthogonal) x-view and y-view intersect with a hyper-bucket B_i, then B_i is in the query space Q_s, and we can calculate the hyper-volume contained in $B_i \cap\ Q_s$ as $hVol_i = A_{ix}A_{iy}$ where A_{ix} and A_{iy} are the areas of B_i in the x-view and y-view, respectively. ■

Next show that $hVol_i\,(t)$ can be maximized. We know that the form of A_{ix} and A_{iy} is that given in 6 and that Lemma 2 gives $hVol_i = A_{ix}A_{iy}$. Hence the form of $hVol$ with *constant renaming* is:

$$hVol_i(t) = at^2 + bt + c + \frac{d}{t} + \frac{e}{t^2} \tag{8}$$

This is the equation for each intersection of the query space with a bucket. We may have multiple buckets that intersect the query space at any one time, but clearly Equation 8 is closed under addition, thus the sum of the intersections still has the form of Equation 8. Taking the derivative and setting it to 0 gives:

$$2at^4 + bt^3 - dt - 2e = 0 \tag{9}$$

There exists the possibility of four real solutions to this problem. If one of the constants a, b, d or e is zero we have a simplified equation to solve. There are nine different versions of Equation 9 that have unique, general solutions.

The general solution of Equation 9 obtained using Mathematica contains just over 3700 characters using the original constants. For space considerations we omit an enumeration of the solutions here.

The solutions can be checked for real values along with the end points and the maximum of them selected as the *Max-Count* for this region. This leads to the following lemma.

Lemma 3 Given the time interval in which $hVol_i\,(t)$ does not sweep through any vertices, the maximum or minimum count over the interval can be calculated in constant time. ■

Definition 6 Let H be a histogram of hyper-buckets. Let $Q_1(t)$ and $Q_2(t)$ be two query points. Let $(t^{\lceil}, t^{\rceil})$ be the query time interval. We define the *Time Partition Order* to be the set of time instances $TP = \{t_1, t_2, ..., t_k\}$, such that $t_1 = t^{\lceil}$ and $t_k = t^{\rceil}$ and for each time interval $[t_i, t_{i+1})$ the set of bucket corner vertices that lie within the query bands remains the same.

Throughout the time interval $[t_i, t_{i+1})$ the number of points within the query band can be estimated by a function of the form given in Equation 8. The Definitions and Lemmas above lead to the *Max-Count* aggregation estimation algorithm given in Figure 3.

Max-Count(H, Q_1, Q_2)
input: A histogram H, query points $Q_1\,(t)$ and $Q_2\,(t)$ and a query time interval $\left(t^{\lceil}, t^{\rceil}\right)$.
output: The estimated Max-Count value.

1. Find all the bucket corner vertices in H. Find the lines between the vertices and the dual of the query points in both the x-view and y-view. Order the lines by their slopes to find the Time Partition Order of the time interval $\left(t^{\lceil}, t^{\rceil}\right)$.
2. For each time interval associated with the Time Partition Order calculate the function of time having the form given in (8).
3. For each such function of time, calculate the maximum value within the corresponding time interval. Store the result in a list.
4. The maximum value in the list is the final result.

Figure 3. Max-Count Algorithm

This leads to the following Theorem.

Theorem 1 Let H be a histogram of points with K number of buckets. Let $Q_1\,(t)$ and $Q_2\,(t)$ be two moving query points and let $\left(t^{\lceil}, t^{\rceil}\right)$ be a time interval. It takes $O(K \log K)$ time to calculate the estimated *Max-Count* value. Since K is a constant, we have running time $O(1)$. ■

4. Experimental Results

This section examines the impact of various parameters on the accuracy of the *Max-Count* estimation algorithm.

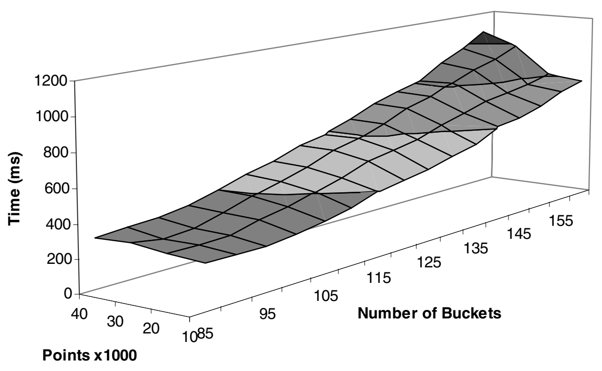

Figure 4. Running Times

We randomly generated several data sets consisting of 2-dimensional moving points which have both position and velocity in each dimension distributed between 0 and 1000 according to the Zipf distribution. Following the examples of [6, 30, 24], we assume a higher density of points at higher initial positions and velocities. The quad space representation of the points is in a hyper-cube that has a length, width, height and depth of 1000 or a hyper-volume of 1×10^{12} $units^4$. This hyper-volume is $10,000$ times larger than the area considered in the one dimensional case [24].

4.1. Experimental Parameters

We consider accuracy in terms of the following parameters:
Number of Buckets: The histogram algorithm is adapted to four dimensional space from [1] and allows us to specify the number of buckets in the histogram. We varied the number of buckets from 20 to 160 in increments of 5.
Number of Points: This is the number of 2-dimensional moving points in the histogram. This value ranged from $10,000$ to $40,000$ points.
Query Range: This is the distance between the quad representation of the query points. Here we used 5 distances ranging from 20 up to 200.
Query Type: This describes the placement of the query points in the histogram. If Q_1 and Q_2 lie in a dense region, we consider this to be a *dense query*. Similarly if Q_1 and Q_2 lie in a sparse region, then the query is said to be a *sparse query*.

4.2. Observations

The *cell size* parameter for the bucketing algorithm affects the accuracy of this algorithm. Specifically if the cell size is too large the assumption of uniform density breaks down. Experimentally we found that this can be minimized by choosing a grid size that is strictly less than the distance between the query points.

We found that the relationship between sparse and dense queries depends on the number of buckets in the regions considered. A sparse region will be partitioned first because the greatest reduction in skew occurs when the skew of one of the buckets goes to 0 while also reducing the skew of the other bucket. This is most clearly seen in Figure 6. Up to a certain number of buckets we see better or at least more uniform result for sparse queries. As we increase the number of buckets, we also find a better approximation in the dense regions where the density of points is more uniform.

Running Time: The experiments ran on a Celeron 2 GHz computer with 512 MB of RAM. Our algorithm that finds an accurate result is a naive algorithm that considers each time a point enters or leaves a bucket. It takes on the order of at least one half hour to run our large data sets. Figure 4 shows the running times of our estimation algorithm in milliseconds for each data set size and number of buckets. The running times include the time to recalculate the time partition order and execute the query.

Range: The significance of the range parameter affects accuracy in two ways. First, as a function of the uniformness of density. If we do not have enough buckets, then a large range may compensate for this in *some*, but not all instances. Second, if the range is close to the chosen cell size in the grid and the points are not truly uniform inside each cell, then the data may be very inaccurate. We experienced an error rate of very close to 50% with the query range below 20. Notice that Figures 5-8 do not reach $range = 20$ and even look counter-intuitive in Figure 6. What is not shown is that error increases dramatically below twenty because the points are so close together.

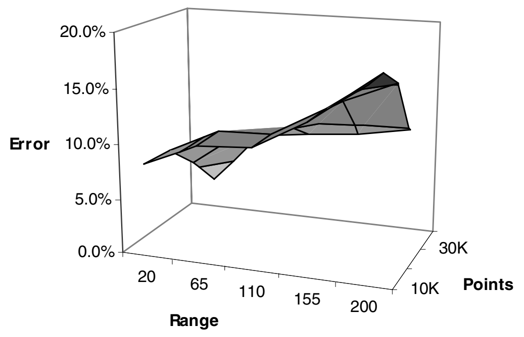

Figure 5. Sparse Query: 85 Buckets

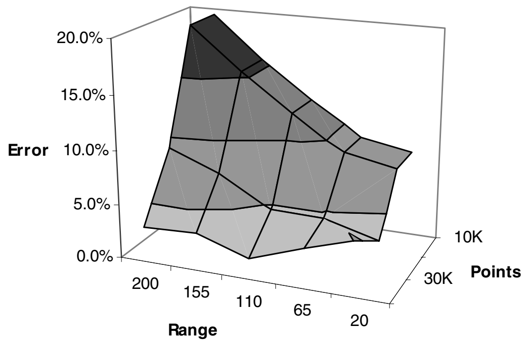

Figure 6. Dense Query: 85 Buckets

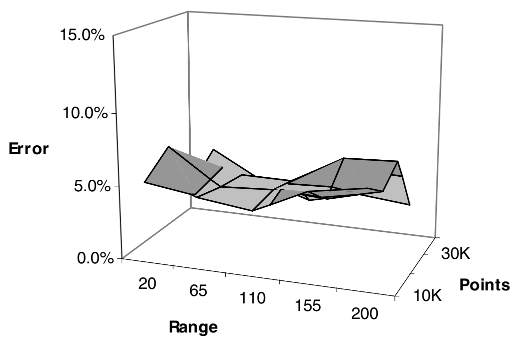

Figure 8. Dense Query: 160 Buckets

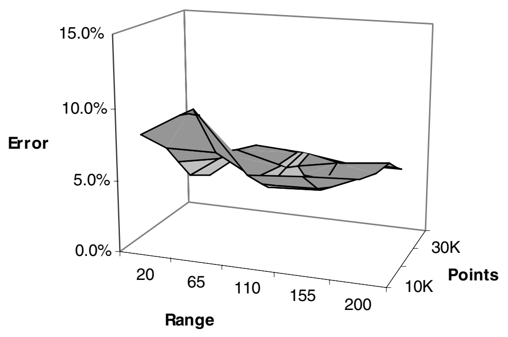

Figure 7. Sparse Query: 160 Buckets

4.3. Experiment: Up to 85 Buckets

In the first section of this experiment we looked at both sparse and dense queries with up to 85 buckets. Figures 5 and 6 show that we are starting to reach a plateau of accuracy. This means that to get uniformly better results we would need to nearly double the number of buckets.

Discussion: Figure 5 shows that bucketing in the sparse region has reached a uniform skew reduction in all the data sets. Hence we see an error that is fairly uniform below 20%.

Figure 6 shows that bucketing in the dense region has *not* reached a uniform density for the $10,000$ and $20,000$ point data sets while it *has* reached a more uniform density in the $30,000$ and $40,000$ point data sets. Here the error has already started to decrease for the $30,000$ and $40,000$ point data sets and reach the $5\% - 10\%$ error range.

4.4. Experiment: Up to 160 Buckets

In the second section of this experiment we looked at both sparse and dense queries with 160 buckets.

Discussion: Figures 7 and 8 show the error in all regions of the parameters reaching a plateau of around 5% error for both dense and sparse queries. We see the effects of the range creeping into the sparse query in Figure 7. Clearly we have reached an acceptable error rate at 160 buckets.

5. Conclusions and Future Work

The *Max-Count* in 2-dimensional space is an aggregate operator that occurs naturally in spatiotemporal databases. Our *Max-Count* estimation algorithm provides a practical solution for linearly moving points. The implementation of the algorithm gives results that match expectations for both running time and accuracy with running times between 10 and $1,212$ milliseconds and accuracy reaching 5% at 160 buckets.

Several natural questions arise for future work. One problem is to build an index that will work with this method that is updatable or to adapt our method to an existing updatable index. A second problem is to extend this method to three spatial dimensions without loosing accuracy or efficiency. A third idea is to generalize the *Max-Count* operator to a new kind of aggregation operator that allows the definition of a specified window size and then determines the location and time for a maximum count.

References

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *ACM SIGMOD International Conference on Management of data*, pages 13–24, 1999.

[2] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points (extended abstract). In *19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 175–186, 2000.

[3] M. Cai, D. Keshwani, and P. Z. Revesz. Parametric rectangles: A model for querying and animation of spatiotemporal databases. In *7th International Conference on Extending Database Technology*, pages 430–444, London, UK, 2000. Springer-Verlag.

[4] M. Cai and P. Revesz. Parametric R-tree: An index structure for moving objects. In *10th COMAD International Conference on Management of Data*, pages 57–64. Tata McGraw-Hill, 2000.

[5] Y. Chen and P. Revesz. Max-count aggregation estimation for moving points. In *11th International Symposium on Temporal Representation and Reasoning*, pages 103–108, 2004.

[6] Y.-J. Choi and C.-W. Chung. Selectivity estimation for spatio-temporal queries to moving objects. In *ACM SIGMOD International Conference on Management of Data*, pages 440–451, 2002.

[7] J. Chomicki, S. Haesevoets, B. Kuijpers, and P. Z. Revesz. Classes of spatio-temporal objects and their closure properties. *Ann. Math. Artif. Intell.*, 39(4):431–461, 2003.

[8] J. Chomicki and P. Revesz. A geometric framework for specifying spatiotemporal objects. In *Sixth International Workshop on Temporal Representation and Reasoning*, pages 41–46, 1999.

[9] S. Grumbach, P. Rigaux, and L. Segoufin. The dedale system for complex spatial queries. In *ACM SIGMOD International Conference on Management of Data*, pages 213–224, 1998.

[10] D. Gunopulos, G. Kollios, J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *The VLDB Journal*, 14(2):137–154, 2005.

[11] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *ACM SIGMOD International Conference on Management of Data*, pages 463–474. ACM, May 2000.

[12] S. Gupta, S. Kopparty, and C. V. Ravishankar. Roads, codes and spatiotemporal queries. In *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 115–124, 2004.

[13] R. H. Guting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[14] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *Advances in Spatial and Temporal Databases, 8th International Symposium*, pages 306–324. Springer, 2003.

[15] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems*, pages 261–272, 1999.

[16] G. Kollios, D. Papadopoulos, D. Gunopulos, and J. Tsotras. Indexing mobile objects using dual transformations. *The VLDB Journal*, 14(2):238–256, 2005.

[17] L. Li and P. Revesz. Interpolation methods for spatiotemporal geographic data. *Journal of Computers, Environment, and Urban Systems*, 28(3):201–227, 2004.

[18] D. Lin, C. S. Jensen, B. C. Ooi, and S. Saltenis. Efficient indexing of the historical, present, and future positions of moving objects. In *Sixth International Conference on Mobile Data Management*, Ayia Napa, Cyprus, May 2005.

[19] H. Mokhtar, J. Su, and O. Ibarra. On moving object queries: (extended abstract). In *21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems*, pages 188–198, 2002.

[20] D. Papadopoulos, G. Kollios, D. Gunopulos, and V. J. Tsotras. Indexing mobile objects on the plane. In *International Conference on Database and Expert Systems Applications*, Aix en Provence, France, September 2–6 2002.

[21] P. Revesz. *Introduction to Constraint Databases*. Springer-Verlag, 2002.

[22] P. Revesz. Efficient rectangle indexing algorithms based on point dominance. In *12th International Symposium on Temporal Representation and Reasoning*. IEEE Computer Society Press, 2005.

[23] P. Revesz, R. Chen, and M. Ouyang. Approximate query evaluation using linear constraint databases. In *Eighth International Symposium on Temporal Representation and Reasoning*, pages 170–175, 2001.

[24] P. Revesz and Y. Chen. Efficient aggregation over moving objects. In *10th International Symposium on Temporal Representation and Reasoning, Fourth International Conference on Temporal Logic*, pages 118–127, 2003.

[25] P. Revesz and S. Wu. Spatiotemporal reasoning about epidemiological data. *Artificial Intelligence in Medicine*, 2006.

[26] P. Rigaux, M. Scholl, L. Segoufin, and S. Grumbach. Building a constraint-based spatial database system: model, languages, and implementation. *Inf. Syst.*, 28(6):563–595, 2003.

[27] P. Rigaux, M. Scholl, and A. Voisard. *Introduction to Spatial Databases: Applications to GIS*. Morgan Kaufmann, 2000.

[28] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *ACM SIGMOD international conference on Management of Data*, pages 331–342, 2000.

[29] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *29th International Conference on Very Large Data Bases*, 2003.

[30] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *19th International Conference on Data Engineering*, 2003.

[31] J. Tayeb, Ö. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. *Comput. J.*, 41(3):185–200, 1998.

[32] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.

[33] O. Wolfson and H. Yin. Accuracy and resource consumption in tracking and location prediction. In *Symposium on Spatial and Temporal Databases (SSTD)*, pages 325–343, 2003.

[34] D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 237–245, 2001.