Performance Evaluation of Spatio-temporal Selectivity Estimation Techniques

Marios Hadjieleftheriou
University of California, Riverside
marioh@cs.ucr.edu

George Kollios
Boston University
gkollios@cs.bu.edu

Vassilis J. Tsotras
University of California, Riverside
tsotras@cs.ucr.edu

Abstract

Many novel spatio-temporal applications deal with moving objects. In such environments, a database typically maintains the initial position and the moving function for each object. Instead of updating the database whenever an object position changes (which is not manageable), updates are issued whenever a function parameter (velocity, direction, etc.) changes. For simplicity, we assume that objects move with linear trajectories. Maintaining the moving functions in a database introduces novel problems. For example, the database can answer queries about object positions in the future*: "find all objects that will be in area A, ten minutes from now". In this paper we present a thorough performance evaluation of techniques for estimating the selectivity of such queries. For many applications, selectivity estimation is more important than exact answers. A traffic monitoring system is a typical example: "estimate the number of vehicles that will be within 2 miles from the intersection of highways I10 and I405, five minutes from now". We consider various existing estimators that can be stored in main memory and are updated dynamically. Small size and low update cost are essential requirements for the applicability of such techniques. Furthermore, we propose two new approaches, a technique that uses histograms and a secondary index-based estimator. We run a diverse set of experiments to identify the strengths and weaknesses of every approach, using a wide variety of datasets.*

1 Introduction

Various recent applications involve moving-object databases (cars moving on a highway system, customers of a cellular network, surveillance applications, etc.) To avoid high update rates, such databases typically maintain functional descriptions of the object movements, issuing updates only when parameters of these functions change (velocity, direction, etc.) In such a scenario the database can also answer queries about the *future* positions of the objects, based on the movement functions stored when the query is issued.

The most common functional representation is a linear trajectory [12, 2, 18, 5, 17, 14, 4, 13, 20, 19, 23, 24]. This assumption is based on simplicity (since linear functions are easy to store and update) and practicality (because even complex object movements can be approximated well by a sequence of adjacent line segments). For example, consider an object moving on a line segment. The object is modeled as a point moving with constant *velocity* $\vec{v}$, starting at *time* t_0 from a specific *location* a_0. The position of the object at time t is: $x = v(t - t_0) + a_0$ for $t > t_0$. Whenever $\vec{v}, a_0$ or t_0 are altered a database update is performed.

Most of the previous work has concentrated on range queries ("find which objects will be in area A, 10 minutes from now") [21, 12, 2, 18, 5, 17, 14] and nearest neighbor queries ("find the 10 objects that will be nearest to a given location, 5 minutes from now") [11, 25]. All these techniques provide exact answers. Nevertheless, there are many applications where **selectivity estimation** and **approximate aggregate computation** are preferable to exact answers. For example: "estimate how many vehicles will be within 2 miles from the intersection of highways I10 and I405, 5 minutes from now". In addition to being useful in query optimization, such queries are important when privacy is a concern (by finding "how many" instead of "who").

Problem Definition: Consider a dynamic collection of objects following linear trajectories. Given a region R and a time interval $\Delta T = [qt_s, qt_e]$ ($t_{now} \leq qt_s < qt_e$), we want to estimate the number of objects that *will* pass through R during ΔT (t_{now} is the current time). In this paper we concentrate on objects moving in 1- or 2-dimensional spaces.

Figure 1 shows three objects moving on a line seg-

This work was partially supported by NSF grants IIS-9907477, EIA-9983445 and IIS-0220148.

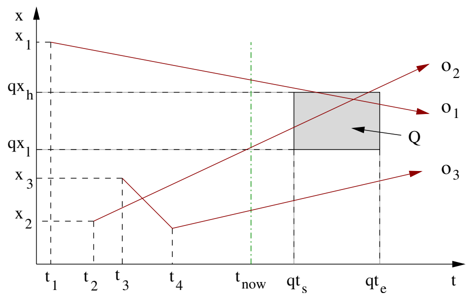


Figure 1: An example of 1-dimensional object movements.

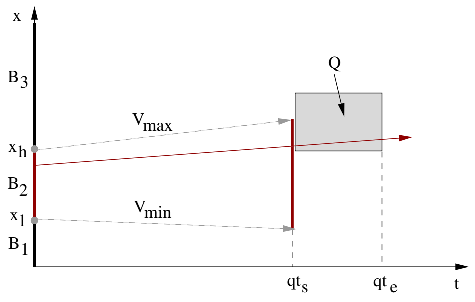


Figure 2: The evolution of a bucket when augmented with velocity information.

ment (1-dimensional space). The vertical axis corresponds to the line segment, while the horizontal axis corresponds to time. The starting location and time instant of every object are also depicted. Object o_3 is shown to issue an update at time t_4. The shaded area corresponds to the range query $Q = [qx_l, qx_h] \times [qt_s, qt_e]$. Only two objects ($o_1$ and o_2) qualify for the answer.

We are aware of only two previous works that address selectivity estimation queries in a spatio-temporal environment. [4] proposed a novel histogram-based technique. That work focused mostly on the 1-dimensional case and made a strong uniformity assumption. [20] extended the work to the multi-dimensional case and dropped the uniformity assumption.

The contributions of this paper are:

- We propose two new spatio-temporal selectivity estimation techniques, one based on histograms and another using a secondary index structure. The first approach uses a *duality* transform of the spatio-temporal space, while the second can be used either on the dual or the original space.

- We present a thorough comparison of all (new and previously) proposed spatio-temporal selectivity estimation techniques as well as a straightforward sampling approach. We test against various datasets using uniform, skewed and other object distributions inspired from practical, real life, examples. Based on this experimental comparison, we draw conclusions about the practicality of the various techniques.

The rest of this paper is organized as follows: Section 2 summarizes previous work, section 3 introduces our proposed selectivity estimation techniques, while section 4 presents the experimental evaluation. Finally, section 5 concludes the paper.

2 Related Work

[4] starts with the simple 1-dimensional case where objects move on a line segment $[X_{min}, X_{max}]$ and are modeled as points following linear trajectories. It is also assumed that the object velocities are distributed uniformly in the range $[V_{min}, V_{max}]$. Given a query Q, the selectivity is then computed as a function of qt_s, qt_e, qx_l, qx_h, X_{min}, X_{max}, V_{min} and V_{max}. Since the location of the objects is not typically uniform, the authors propose building a spatial histogram (like Minskew [1]) that partitions the 1-dimensional segment into consecutive ranges (buckets). The histogram algorithm is modified to calculate the minimum and maximum velocities of all objects contained in each range. Each bucket is thus characterized by three quantities: (1) the bucket range (or bounding region for the 2-dimensional case), (2) the number of objects initially contained in this range, and (3) the minimum and maximum velocities of these objects.

This is equivalent to partitioning the space into time-evolving buckets, by assigning velocities to the end-points of each bucket. Consider bucket $B_2 = [x_l, x_h]$ (on the vertical axis) in figure 2. If its right end-point x_l moves with velocity V_{min} and the left end-point x_h moves with velocity V_{max}, the bucket's extent at time qt_s can be easily calculated. By assuming uniformity of object locations and velocities inside B_2, we can compute the contribution of this bucket to the query Q. The sum of the contributions from all buckets is the total selectivity of the query.

Initially, objects are assigned to buckets by using their locations at time $t = 0$. Since objects issue regular updates the histogram needs to be adjusted accordingly. An object update is given as a tuple: $\{O_{id}, (v, a), (v', a')\}$, where (v', a') are the new parameters of the object moving function. Using this information we can remove the object from its original bucket and reassign it to a new bucket. The problem with this approach is that while the original Minskew histogram assumes uniform distribution of objects within buckets, as updates occur and objects move around, this uniformity can be easily lost. One solution would be to rebuild the histogram every so often, but this leads to increased

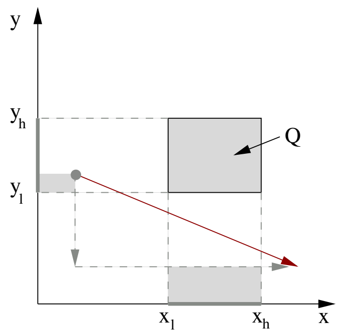


Figure 3: An object intersecting the query projections on both dimensions but not the query region.

computational cost since a complete database scan is required. To address this problem [4] proposes maintaining the histogram over a (small) random sample of the objects and rebuilding it every time instant. This sample is maintained in main memory using a backing algorithm as described in [7].

The technique is extended to handle points moving on a 2-dimensional plane by building a 2-dimensional histogram and taking projections of the buckets, the objects and the queries on every dimension. The 2-dimensional selectivity is computed as the product of the individual 1-dimensional results on each projection.

In addition to the strong assumption of uniform object velocities, [4] over-estimates the selectivity in the 2-dimensional case. Projecting the data and the queries on each dimension introduces false positives, since objects that intersect with the query projections on both dimensions do not necessarily cross the original query (such a case is shown in figure 3).

The technique recently proposed in [20] drops the velocity uniformity assumption and does not use projections in the 2-dimensional case, thus avoiding over-estimation. Consider an object moving in a 2-dimensional space. Let's assume that the object is currently at position (x, y), moving with velocity (v_x, v_y). [20] builds a 4-dimensional histogram, using 4-dimensional points (x, y, v_x, v_y). The resulting buckets have a spatial MBR and a velocity MBR (V-MBR). Each bucket contains all objects that fall inside its spatial MBR with velocities bounded by the V-MBR. Since the velocities of the objects are included in the histogram construction process, the resulting buckets will contain objects that are uniformly distributed inside both the MBR and the V-MBR. Hence, there is no need for the velocity uniformity assumption. [20] uses the spatial and velocity MBRs to derive analytical formulas for computing the contributions of both MBRs to the selectivity of a range query. The authors propose creating the histogram on the whole dataset (not a sample) but rebuilding it less often in order to offset the cost of a full database scan. While the approach in [20] is more complex to implement, it has another advantage, as it is more general and applies also for moving queries (when the spatial range of the query moves during the query time interval).

Both [4, 20] as well as our proposed histogram technique, use the Minskew [1] algorithm for creating histograms. The 2-dimensional Minskew algorithm partitions the universe into a *user specified* number of *disjoint* rectangular buckets (the algorithm can be extended for any number of dimensions). Every bucket is characterized by its spatial extent and the number of objects that fall in it. The goal is to partition the space such that the distribution of objects inside each bucket is uniform. Minskew starts with a very fine uniform grid of cells (the number of cells must be much larger than the total number of requested buckets) and calculates the spatial density of each cell (i.e. the number of objects contained in a cell). Then, it combines a number of cells into a bucket by minimizing the statistical variance (spatial-skew)* of the spatial densities of the cells. Initially, all cells belong to a single bucket that covers the entire universe. Then, the split (in any dimension) that will result in two buckets with minimum *global* spatial-skew, is performed. This procedure continues recursively. The algorithm stops when the required number of buckets has been reached. Since this algorithm is exhaustive, heuristics that produce near optimal buckets with quadratic (on the number of cells) computational cost can be used.

3 New Spatio-temporal Selectivity Estimators

We propose two techniques: The first one is based on histograms and the duality transform of [12]. The second assumes that a secondary index exists on the moving objects. Similarly with [20], the histogram approach builds an estimator using the whole dataset and is rebuilt frequently. The index estimator is also build on the whole dataset but it is dynamically updated.

3.1 Histogram Estimators Using the Duality Transform

The duality transform was used in [12] and recently in [6] for indexing moving objects. The *primal* **space-time** representation (figure 1) is equivalent to its *dual* **velocity-intercept** representation (figure 4). In this transform:

*The variance V of N numbers $f_1, \ldots, f_N$ is equal to $V = \frac{1}{N}\sum_{i=1}^{N}(f_i - \bar{f})^2$, where $\bar{f}$ is the average of $f_1, \ldots, f_N$.

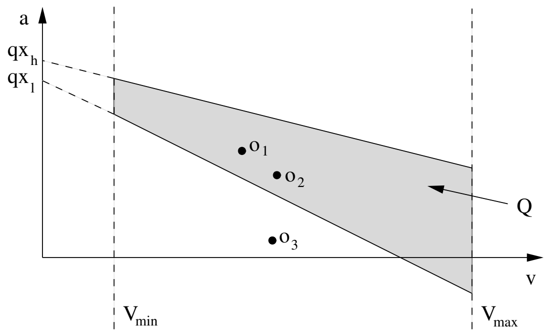


Figure 4: A range query transformation in the dual space. Since trajectories $o_{1,2}$ intersect query Q in the primal space, only the same points are contained in Q in the dual space.

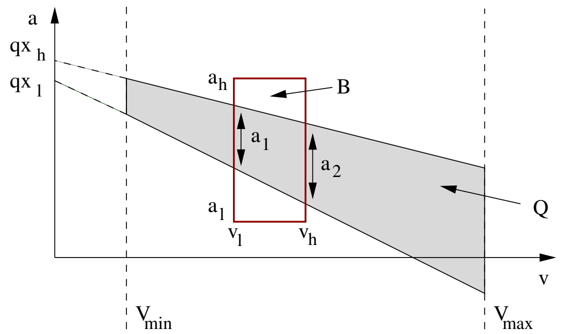


Figure 5: Example of computing the contribution of bucket B to query Q .

- An object trajectory $x = vt + a$ (a line) is transformed into a point (v, a). Coordinate a is the position where the trajectory intersects the vertical axis $t = 0$ in the original space and v is the object velocity (this is called the Hough-X transformation in [10]).

- A range query $Q : [qx_l, qx_h] \times [qt_s, qt_e]$ is transformed into a linear constraint. Object (v, a) will pass though Q if and only if:

$$\begin{cases} \text{if } v > 0: & V_{min} \le v \le V_{max} \bigwedge \\ & a + qt_e v \ge qx_l \bigwedge a + qt_s v \le qx_h \\ \text{if } v < 0: & -V_{max} \le v \le -V_{min} \bigwedge \\ & a + qt_s v \ge qx_l \bigwedge a + qt_e v \le qx_h \end{cases} \quad (1)$$

An advantage of this approach is that lines (which are objects difficult to index [10]) are transformed to points (which can be indexed with a variety of Point Access Methods). The difficulty of course is that the rectangular query of figure 1 is transformed to the shaded trapezoid of figure 4.

Our proposed histogram technique works as follows:

1. Convert the object trajectories to Hough-X space.

2. Partition the Hough-X space into buckets using a 2-dimensional spatial histogram, like Minskew. Each bucket contains the following information:

   (a) An X-MBR $[v_l, v_h] \times [a_l, a_h]$.

   (b) The number of points N that fall inside the X-MBR.

3. Update the histogram as objects are updated.

4. Rebuild the histogram after the number of updates exceeds a given threshold.

The selectivity of query Q is the sum of the contributions from all histogram buckets. Since Minskew tries to maintain a uniform distribution within each bucket B, the contribution of B is proportional to the area of its intersection with Q and the number of objects N contained in B. The intersection of Q (represented by equation (1)) and B (which is a rectangle), can be computed efficiently using any polygon clipping technique [22, 16]. An example is shown in figure 5. The selectivity is $S_B = N\frac{Area_{Q \cap B}}{Area_B}$, where $Area_{Q \cap B} = \frac{(a_1+a_2)(v_h-v_l)}{2}$ and $Area_B = (v_h - v_l)(a_h - a_l)$.

Every time an object issues an update the histogram is adjusted accordingly. Suppose an update $\{O_{id}, (v, a), (v', a')\}$ occurs. We locate the bucket with X-MBR that contains point (v, a) and decrement its counter by one. We find the new bucket with X-MBR that contains point (v', a') and increment its counter by one. When using the Hough-X space to create the histograms the velocity range is bounded by $[V_{min}, V_{max}]$. On the other hand, the intercept range is unbounded. Since intercept calculation is relative to time $t = 0$ and the current time advances indefinitely, the range of intercepts always increases. Thus, the new point (v', a') might fall outside of the intercept range that was used to construct the histogram. In that case either the appropriate boundary bucket has to be enlarged to contain the new point, or the update can be ignored, since the histogram will eventually be rebuild. In our implementation we chose to ignore such updates given that the histogram was rebuilt every ten time instants.

This technique extends to 2-dimensional movements in a straightforward way. A linear trajectory in the 2-dimensional space becomes a 4-dimensional point (v_x, v_y, a_x, a_y) in the Hough-X space. A 4-dimensional histogram has to be constructed and the transformed queries can be represented with an extended linear constraint. However, the 4-dimensional linear constraint has no closed form solution, thus the intersecting volume of a 4-dimensional bucket and the query has to be computed using monte-carlo integration [15].

We should note here the differences and similarities with [20] (the techniques were developed independently). Both techniques drop the velocity uniformity assumption of [4] and built a 4-dimensional histogram on the whole dataset and not a sample. Both approaches rebuild the histogram less frequently (whenever the number of updates exceeds a threshold). The approach discussed here uses the Hough-X space while [20] uses the 4-dimensional space (x, y, v_x, v_y). Similarly to our technique, the selectivity estimation formula in [20] does not have a closed form solution. A complex algorithm based on the trapezoid rule was proposed to compute the selectivity. The monte-carlo approach used here for computing 4-dimensional volume intersections is as efficient (in run-time and accuracy as the experimental section shows) and much simpler to implement than the combined trapezoid rule needed for the MBR and the V-MBR of [20].

The update cost of our technique (as well as that of [20]) is expected to be higher than the technique proposed in [4] for three reasons. First, for the 2-dimensional case our technique uses a 4-dimensional histogram which has higher rebuilding cost when compared to the 2-dimensional histogram of [4] (the 4-dimensional grid is much larger than the 2-dimensional grid). Second, performing a full database scan is more expensive when compared to maintaining a sample in memory. Finally, a monte-carlo computation needs to be performed for every bucket, to calculate the selectivity. Nevertheless (and as it will be apparent from the experimental evaluation), we argue that the estimation accuracy of our technique improves dramatically due to better histogram quality which justifies a slightly increased update cost for the 2-dimensional movements. Moreover, we were able to reduce the update cost by rebuilding the histogram less often (hence reducing the database scans to a minimum) without sacrificing much of the estimator accuracy.

Note, that for 1-dimensional movements, the 1-dimensional histogram of [4] needs a very large grid size in order to produce good estimates. In that case, the 2-dimensional histogram construction of our technique becomes faster. Furthermore, the 2-dimensional Hough-X formulas have a closed form solution (i.e., the monte-carlo computation is not needed).

[12] uses a second space transformation called Hough-Y [10]. Instead of using a velocity-intercept space, the Hough-Y transformation uses the reciprocal of the velocity $n = \frac{1}{v}$ and the intercept divided by the velocity $b = \frac{-a}{v}$. The advantage of this transformation is that the query is represented by a simpler linear constraint, such that the intersections between the query and the buckets can be computed analytically (thus removing the need for the monte-carlo integration). The drawback however is that the resulting distribution of the objects on the Hough-Y universe becomes highly skewed even for data that is non-skewed on the primal space. This is because the division by the object velocity distorts the dual space representation. We also implemented our proposed technique using this dual transformation. Indeed the estimator accuracy deteriorated substantially. For that reason in the rest of the paper we discuss the Hough-X dual transform only.

3.2 Index Based Estimator

This approach assumes that a disk-based access method is available for indexing the moving objects. Various such indexing techniques have been proposed in the recent literature [12, 2, 5, 6, 14, 18, 17]. Typically, an index is maintained for answering exact range and nearest neighbor queries. Assuming that such an index exists, we built a dynamically updatable selectivity estimator. The combination of the existing index, with the proposed estimator, provides a unified solution for a variety of queries (range, nearest neighbor as well as selectivity estimation).

Other advantages of this approach are:

- The estimator is build on the whole dataset and not on a sample leading to better accuracy.
- The estimator does not have to be rebuilt. Only relevant entries are affected during updates and thus it adapts accordingly.
- The size and accuracy can be tuned on-line.

For ease of exposition the following discussion assumes that an R-tree [8] is indexing the complete database in Hough-X space [12, 14]. Other spatio-temporal indices (like the TPR-tree [18] which indexes objects in the primal space) can also be used in a straight-forward manner since the implementation details of our technique are not affected by index updating policies.

The estimator is created as an in-memory hash table that contains a number of entries. Every entry E corresponds to a specific leaf node (data page) L of the R-tree. Each entry has the following properties:

1. An E-MBR, which is the MBR of L.
2. A hash key, equal to L's unique node identifier.
3. The number of objects N contained in the E-MBR (i.e., the number of objects contained in L).

Any entry can be further split in-memory (without affecting the persistent structure) yielding multiple entries

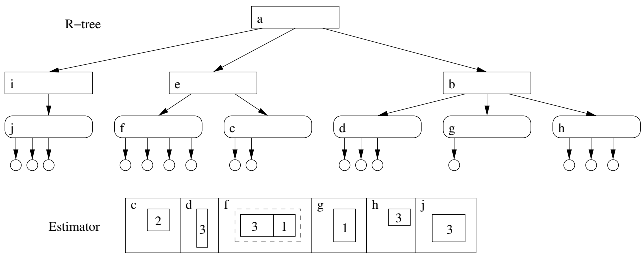


Figure 6: The index based estimator.

per leaf if needed (that share a common hash key). As discussed in the experimental section, such splits result in increased estimator size, but also increased accuracy since the E-MBRs become more fine-grained, partitioning space into smaller regions. Various splitting algorithms may be used for such in-memory splits. For simplicity, we use the R*-tree splitting heuristic [3].

Figure 6 depicts an R-tree (leaf nodes are plotted with rounded corners). The estimator entries are shown at the bottom. The hash table contains one entry per leaf and every entry holds the number of data contained in the associated leaf. In addition, entry f is split into two subentries, both indexed by the same hash key.

The cost of updating the hash table is insignificant. The estimator is modified along with the index. More specifically, whenever a leaf node of the tree is updated (and thus loaded into main memory) the corresponding entry in the hash table is accessed and adjusted as necessary. Since the entries of the estimator correspond to the actual partitioning of the data, imposed by the secondary index, we expect the quality of the estimated results to be related to the quality of the spatial index.

For answering a selectivity query, the hash table entries are scanned sequentially (if the hash table size is considerable, the buckets can instead be organized as an in-memory spatial index). The contribution of entry E to query Q is proportional to the intersecting area of its E-MBR with Q and the number of objects N. Query selectivity can be computed similarly with the histogram technique. Extending the approach for multidimensional environments is straightforward.

The size of the estimator is proportional to the total number of leaf nodes L_s in the secondary index which in turn depends on the total number of data entries M and the average fanout of the index f_{avg} ($L_s = \frac{M}{f_{avg}}$). For example, in a 2-dimensional R-tree with $M = 10^6$ data objects, a page size of 2 Kbytes, where every entry is 20 bytes long (4 floats for the MBR and 1 integer for the id), the node fanout f is 100 entries per node. Given that the average R-tree utilization per node is typically around 70%, the average fanout would be $f_{avg} = 70$. Hence, there are $L_s = \frac{10^6}{70} = 14286$ leaf nodes and the estimator will contain at least as many entries. Given today's large main memories, L_s is still small when compared to M, the total number of moving objects.

4 Experimental Results

All experiments were run on an Intel Pentium(R) 4, 1.60Ghz CPU with 1Gb of main memory. We run several experiments testing the proposed techniques for 1-dimensional and 2-dimensional environments.

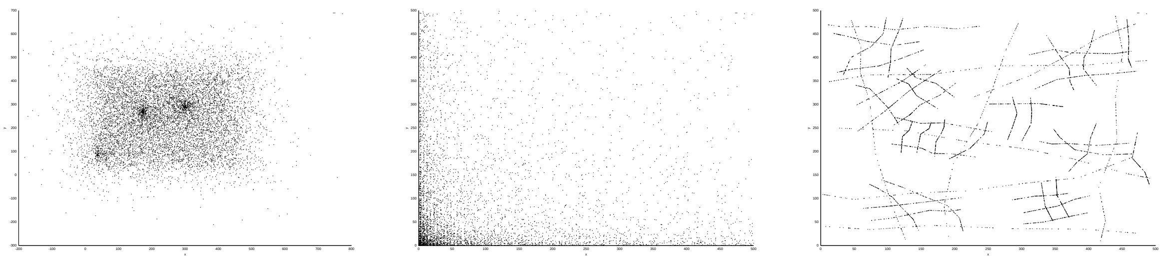
4.1 2-dimensional Movements

Dataset and Query Description. We generated four different datasets of moving objects, containing 1 million objects each. The spatial universe was set to 500 by 500 miles. Object velocities were randomly generated using skewed and uniform distributions, between 10 miles per hour and 110 mph. All simulations were run for 200 time instants (every time instant corresponding to 1 minute). At least 1% of the objects issued an update each time instant.

Each dataset is described by the initial distribution of the moving objects. For the first dataset (denoted as UNIFORM), objects were initially distributed using a uniform distribution and then moved randomly inside the universe. In the second dataset objects again started with a uniform distribution but then each object randomly selected one of three destinations, with a specified probability. Here the three destinations "attracted" the objects (the ATTRACTOR dataset, an example frame after 80 time instants is shown in Figure 7(a)). In the third dataset objects were positioned initially using a skewed distribution and then moved randomly (a SKEWED dataset shown in Figure 7(b) after 20 time instants). Finally, the last dataset represents a network of freeways where a freeway is a collection of connected line segments. The objects followed random paths on this network (a frame of the ROAD dataset is shown in Figure 7(c)). Within each dataset we varied object velocities using uniform or skewed distributions.

We also created four different sets of queries with all combinations of small and large sizes of time intervals and spatial extents (see table 1). Small size corresponds to a time interval with length 1 time instant and a 5 miles range in the spatial domain. Large size corresponds to 5 time instants and 15 miles range. We generated 2 queries from each type for every time instant, giving a total of 1600 queries per simulation. For all techniques based on random samples we run every experiment multiple times and averaged the final results.

Performance Optimizations. We present comparison results for the following techniques: Random sampling (denoted in the rest by S), the spatio-temporal se-



(a) Attractor (b) Skewed (c) Road

Figure 7: Datasets

Table 1: Query sets.

Set	SS	SL	LS	LL
Time	1	1	5	5
Space	1%	1%	3%	3%

lectivity estimation technique of [4] (HC), the spatio-temporal selectivity estimation technique of [20] (HT), histogram on Hough-X (HX) and the R-tree based estimator on Hough-X (R-tree). Since all techniques provide approximate answers we calculated the relative error of each technique as a ratio over the exact answer. The relative error is the ratio $\frac{|E-A|}{E}$, where E and A are the exact and approximate answer, respectively. Queries where the exact answer was zero were ignored.

We present the "optimized" performance for every approach. We first varied different parameters to fine-tuned each technique. We tested the histogram techniques (HC, HT and HX) with varying number of buckets and grid sizes. Both parameters affected the computation cost and the accuracy of the histogram results. We observed that a very small or very large number of buckets or grid size has a negative effect on the estimations. We noticed that a very large grid size increases the computational cost substantially, without providing noticeable improvement in accuracy. We tested using up to 5000 buckets since for higher number of buckets the building cost of the histogram became too expensive to be considered as a viable solution. For our experiments, HX and HT worked better when using up to 3000 buckets, while HC gave better results for up to 1000 buckets.

Another important parameter for the sampling based techniques (S and HC) is the sampling factor. We experimented with up to 1% samples. As expected, for HC the computational cost of the histogram is not affected at all from the sample size, since the core of the histogram construction algorithm depends only on the number of buckets and the grid size. The results on the other hand become more accurate as the sample becomes larger. For simplicity all graphs refer to 1% samples.

We tried a number of R-tree estimators with varying number of hash table sizes. It became apparent that the bigger the estimator size (the more fine-grained the entries of the hash table) the better the accuracy of the results. For the rest of this section the R-tree based estimators have approximately 10000 entries.

Comparisons. Figure 8 plots the sizes of all estimators. The R-tree estimator has the largest size followed by HC and S. The smaller sizes are used by HX and HT. The best compromise between computational cost and accuracy for every technique is achieved for different sizes. For HC and S, since the sample needs to be updated in memory the sample size dominates the total size of the estimator (the number of buckets is much smaller in comparison). The size of the R-tree estimator is large since the secondary index needs more than 10000 leaf entries in order to store 1 million objects, thus making the estimator at least as large. The rest of the techniques are more space efficient because only the histogram buckets are kept in memory. While an interesting comparison parameter, the estimator size was not a real implementation concern since even the larger estimator required only 234 Kbytes of main memory.

Figure 9 plots the normalized rebuilding cost for all techniques and the SKEWED dataset (similar behavior was observed for the rest of the datasets). For HX and HT the update cost is a full database scan plus the histogram construction time (in our experiments the histogram was rebuilt every 10 time instants). For HC we build the histogram from the stored sample every time instant as described in [4]. S has minimum update cost, since the backing algorithm of [7] is not computationally expensive. The R-tree estimator update cost is the time needed to adjust and split the entries of the in memory hash table. It is faster (about four times) than HC since updating occurs on-line. HX and HT have the highest update cost for 2-dimensional movements, due to the 4-

dimensional histogram and the database scan overhead.

Figure 10 shows the query cost for all estimators. For the R-tree, HT and HX the cost is a sequential scan of the buckets plus the time to compute the complex integrations involved. HC does not have the monte-carlo integration overhead. S performs one iteration per query over the sample, and since the sample size is very small the cost per query is minimal. The R-tree is ten times more expensive than HX and HT. The more buckets that intersect with a query the more integrations need to be performed. The R-tree estimator contains twice as many buckets compared to the other techniques, which may also overlap each other (this does not happen with buckets produced with the Minskew algorithm), thus the monte-carlo computations increase proportionally.

Figures 11 to 15 plot the relative error for various datasets. Figure 11 shows the results for the UNIFORM dataset using uniform velocities. This dataset is similar to the datasets used in the performance analysis of [4, 20]. HX and HT performed very well giving from 1% to 5% errors depending on the query types. The R-tree estimator followed (with up to 10% error). S did not work well, yielding more than 60% error in some cases. HC worked better than sampling, but still gave as much as 40% error for LS queries. It is apparent that these techniques (HC and S) are affected substantially from the quality of the sample. The standard deviation of our measurements for HC was 4.3% (SL queries), and for S up to 15.1% (SS queries). The large sparsely populated space and the small sampling factor are responsible for the bad sample quality.

Next we examined the UNIFORM dataset with skewed velocities (Figure 12). The behavior was as above, except that the performance of HC deteriorates substantially when skewed velocities are used. This is because the velocity uniformity assumption does not hold. The other techniques, remain unaffected.

Due to lack of space, in the rest we depict only datasets with skewed velocities. The full list of experiments appears in the extended version of this paper [9].

For the ATTRACTOR dataset with skewed velocities (Figure 13) all techniques gave more than 10% errors for all query types. HX and HT again perform similarly, yielding between 10% and 15% error. The rest of the techniques performed poorly, giving more than 22% errors for all cases. For HC and S the standard deviation of the measurements was up to 2.3% and 7.1% respectively, for some query types. Interestingly, the R-tree estimator is affected by the ATTRACTOR data. We attributed this behavior to the difficulty of the original index to maintain good MBRs as the data become two concentrated to the three destinations.

Similar conclusions are drawn in the SKEWED dataset (Figure 14). The results deteriorate further since the data are skewed from the initial object distribution. For SS queries S gave 21% standard deviation and HC 34%. This means that sampling cannot be used for such datasets. The R-tree estimator also performed poorly since the initial buckets of the index were probably of low quality (given the data skewness). HX and HT did not do well for LS queries, giving more than 30% error.

Surprisingly, for the ROAD dataset (Figure 15) results were different. S yielded less than 14% error for all query types and it was better than HC in all cases, which gave 26% error for SS queries and less than 13% for all other types. For this dataset we generated random query sets with one constraint: Queries should contain at least one line segment. Since objects are highly constrained on the 2-dimensional plane (following specific line segments), a small sample is enough to represent the distribution of the dataset accurately (a smaller active space means better sample quality). On the other hand, the techniques that use a 4-dimensional space lose this property (the space is sparse). HX, HT and R-tree gave more than 19% error for all query types.

Clearly, no technique is best for all datasets. For uniform object distributions HX and HT should be the choice. For skewed distributions, no technique gave satisfactory results (and it remains an interesting open problem). Datasets with skewed object distributions occur very often. For example, imagine the traffic patterns after a big event like the *Super Bowl*. Objects tend to disperse from a stadium in a non-uniform manner following freeways to several different directions (imagine the stadium being at the lower left corner of the universe in Figure 7(b) and cars leaving toward the north and east). High degrees of skewness seem to affect the histogram and the R-tree schemes. For the ROAD dataset results were surprising. Simple sampling techniques actually worked well, yielding reasonable errors and should thus be considered as an alternative.

4.2 1-dimensional Movements

We also examined all estimators for 1-dimensional movements. This is useful for applications involving straight-line segments (e.g., a busy highway). Our 1-dimensional synthetic environment consists of one highway represented as a line segment extending from 0 to 1000 miles. We randomly placed 16 entrances/exits on this highway. The simulations run for 500 time units, each time unit corresponding to 1 minute. In the beginning, 400,000 vehicles are randomly placed on the highway. This corresponds to an average of 400 vehicles per mile. Every vehicle moves toward a specific exit, with a constant speed. When a vehicle reaches its exit, it is removed from the highway and a new vehicle appears through a random entrance. Thus, the number

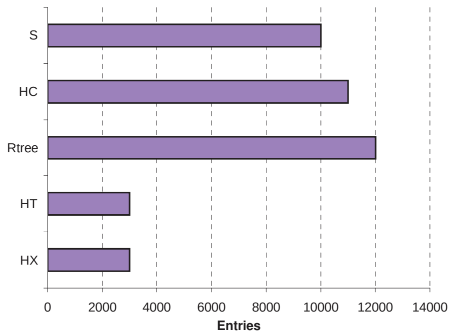


Figure 8: Estimator sizes in number of entries.

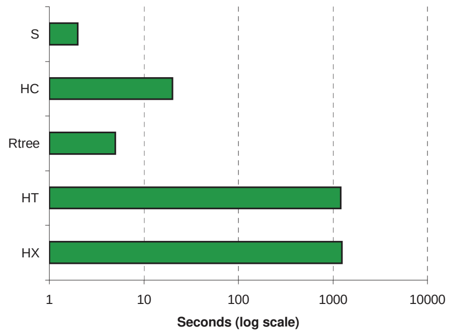


Figure 9: Rebuilding computational cost (logarithmic scale).

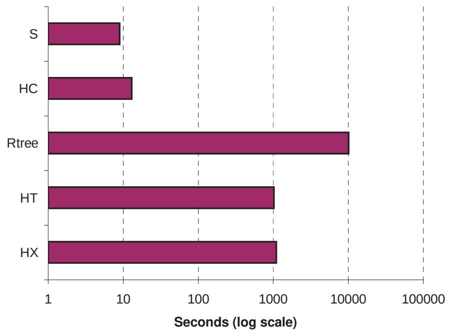


Figure 10: Query computational cost (logarithmic scale).

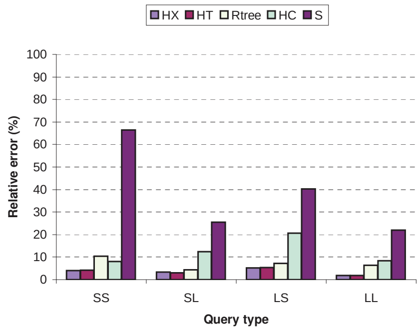


Figure 11: UNIFORM dataset, uniform velocities.

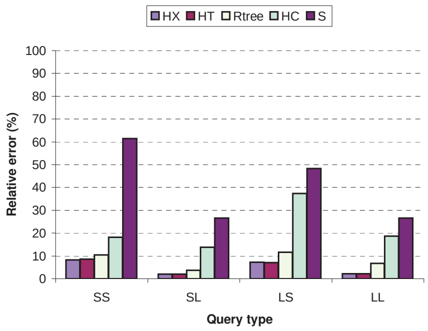


Figure 12: UNIFORM dataset, skewed velocities.

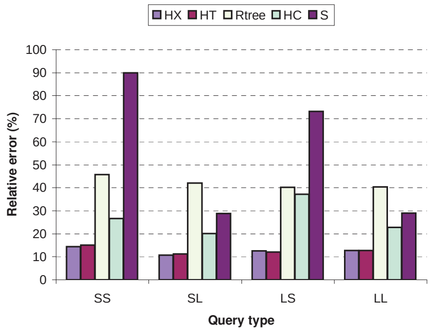


Figure 13: ATTRACTOR dataset, skewed velocities.

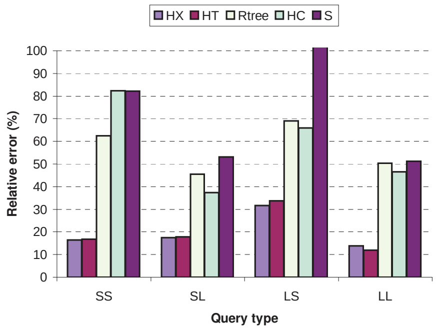


Figure 14: SKEWED dataset, skewed velocities.

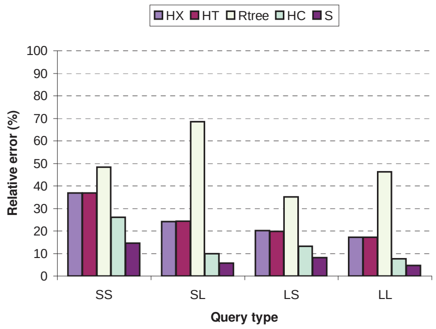


Figure 15: ROAD dataset, skewed velocities.

of vehicles per time instant is kept constant. Every time instant at least 1% of the objects issue an update. We tested both with uniform and skewed velocity distributions. Velocities were chosen between 10 mph and 110 mph. For the skewed distribution we split the vehicles into two groups, one for speeds between 10 and 55 mph and the other between 55 and 110 mph. The first group corresponds to slow moving vehicles in traffic with a few exceptions. The second group corresponds to normal highway conditions with a few speeding vehicles.

Figure 16 shows the results for skewed velocity distribution. For simplicity, HT is not included in the graph as its performance was equivalent to HX. All techniques worked quite well for 1-dimensional movements, with HX and R-tree being the best. It is interesting to note that S is slightly better than HC for the 1-dimensional case, making the histogram construction overhead obsolete in that case.

Memory requirements are the same as for the 2-dimensional experiments. S and HC require much more space than HX, because they keep the sample in memory. R-tree is as large since the resulting tree has many data pages. In terms of computational cost, S is again the fastest technique requiring only 2 seconds to answer all queries. R-tree run in 10 seconds, HX in 600 seconds and HC in 2000 seconds. R-tree is very fast in the 1-dimensional case, since no monte-carlo integration is needed in the 2-dimensional Hough-X space (intersections can be computed with polygon clipping). HX is slow because of the histogram construction time and the database scans. Finally, HC is very slow because the 1-dimensional histogram needs a very large grid size (3000 cells) to produce competitive results. We omit these graphs due to lack of space.

As a conclusion for the 1-dimensional case, we note that if an index is present, the index based estimator gives the best performance/query cost trade-off. S is also a good choice, since it is very efficient and easy to implement when an index is not present. Finally, HX gives very accurate results for an added computational cost which, though, could be adjusted by rebuilding the histogram less often.

5 Conclusions

We conducted a thorough performance evaluation of spatio-temporal selectivity estimation techniques. We considered various existing estimators that can be stored in main memory and are dynamically updated as objects are moving. Furthermore, we proposed two techniques, a simple histogram approach and an index-based estimator. We run a diverse set of experiments on 1- and 2-dimensional environments, using various synthetic datasets. Clearly no technique is best for all types

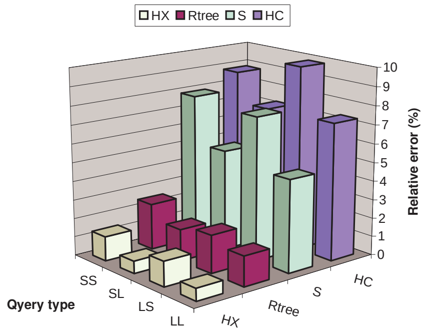


Figure 16: 1-dimensional dataset with skewed velocities.

of datasets. For 1-dimensional movements the index-based estimator is the most robust technique. For 2-dimensional and uniformly distributed data, histograms give the best results. For a freeway network, a simple random sample gives the best answers. For highly skewed environments the techniques did not provide good estimates. Finding a better alternative is left as open problem for future work.

Acknowledgements: We would like to thank Yufei Tao for providing us with an early version of [20] and for many helpful discussions.

References

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

[2] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *In Proc. of the 19th ACM Symp. on Principles of Database Systems (PODS)*, pages 175–186, 2000.

[3] N. Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R* - tree: An Efficient and Robust Access Method for Points and Rectangles. *Proceedings of ACM SIGMOD*, pages 220–231, June 1990.

[4] Yong-Jin Choi and Chin-Wan Chung. Selectivity estimation for spatio-temporal queries to moving objects. In *Proc. of ACM SIGMOD*, 2002.

[5] H. D. Chon, D. Agrawal, and A. El Abbadi. Storage and retrieval of moving objects. In *Mobile Data Management*, pages 173–184, 2001.

[6] K. Elbassioni, A. Elmasry, and I. Kamel. An efficient indexing scheme for multi-dimensional moving objects. In *Proc. of ICDT*, 2003.

[7] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. of the VLDB*, 1997.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD*, pages 47–57, 1984.

[9] M. Hadjieleftheriou, G. Kollios, and V.J. Tsotras. Performance evaluation of spatio-temporal selectivity estimation techniques. In *Technical Report, Dept. of Computer Science, UC-Riverside, Jan.*, 2003.

[10] H. V. Jagadish. On indexing line segments. In *Proc. 16th International Conference on Very Large Databases, Queensland, Australia*, pages 614–625, August 1990.

[11] G. Kollios, D. Gunopulos, and V. Tsotras. Nearest Neighbor Queries in a Mobile Environment. *In Proc. of the Spatio-Temporal Database Management Workshop, Edinburgh, Scotland*, pages 119–134, 1999.

[12] G. Kollios, D. Gunopulos, and V. Tsotras. On Indexing Mobile Objects. *In Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS)*, pages 261–272, June 1999.

[13] H. Mokhtar, J. Su, and O.H. Ibarra. On moving object queries. In *In Proc. of the 21st ACM Symp. on Principles of Database Systems (PODS)*, pages 188–198, 2002.

[14] D. Papadopoulos, G. Kollios, D. Gunopulos, and V.J. Tsotras. Indexing mobile objects on the plane. In *In Proc. 5th International Workshop on Mobility in Databases and Distributed Systems (MDDS) 2002 (to appear)*, 2002.

[15] William H. Press. *Numerical Recipes in C: The Art Of Scientific Computing*. Cambridge University Press, 1992.

[16] David F. Rogers. *Procedural Elements for Computer Graphics*. McGraw Hill, 1985.

[17] S. Saltenis and C. Jensen. Indexing of Moving Objects for Location-Based Services. *To Appear in Proc. of IEEE ICDE*, 2002.

[18] S. Saltenis, C. Jensen, S. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. *In Proceedings of the ACM SIGMOD*, pages 331–342, May 2000.

[19] A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. *In Proceedings of the 13th ICDE, Birmingham, U.K*, pages 422–432, April 1997.

[20] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *ICDE 2003 (to appear)*, 2003.

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

[22] B. R. Vatti. A generic solution for polygon clipping. pages 56–63, July 1992.

[23] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *ICDE*, pages 588–596, 1998.

[24] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. *In Proc. of 11th Int. Conf. on SSDBMs*, pages 111–122, July 1998.

[25] Z.Song and N.Roussopoulos. K-nearest neighbor search for moving query point. In *In Proc. SSTD 2001*, pages 79–96, 2001.