

# Storage and Retrieval of Moving Objects <sup>\*</sup>

Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi

University of California, Santa Barbara  
Santa Barbara, CA 93106, USA  
{hdchon, agrawal, amr}@cs.ucsb.edu

**Abstract.** We investigate the problem and provide a data model storing, indexing, and retrieving future locations of moving objects in an efficient manner. Each moving object has four independent variables which allow us to predict its future location: a starting location, a destination, a starting time, and an initial velocity. To understand the underlying complexity of the problem, we investigate and categorize the configurations where two variables can vary. Based on that understanding, we choose a configuration which is to some extent restrictive, but still can be used in a wide variety of realistic settings. A performance study shows that our model has much less overhead in processing range queries compared to other proposed approaches.

## 1 Introduction

Imagine you are driving a car equipped with an intelligent computer system that can communicate with a control center via a wireless network. While you are on a highway approaching a city area, you decide to stop for a cup of coffee and your car computer tells you that there are several upcoming exits on the road which have a coffee shop nearby. You ask the computer which one to choose to avoid traffic congestion in such a way that if you stop at the one after passing the city area, then you would avoid the upcoming congestion or if you stop at the one before the city, then existing congestion would disappear during your stopover. Alternatively, imagine that you have a flight scheduled at 11 a.m. at the airport two hours from your home and there is a highway control center which guarantees you the two hours travel time if you notify the control center of your schedule in advance. Then you do not have to leave your home earlier than necessary. There has been significant research in the area of Advanced Transportation Systems [4, 7] in the past decade and the next generation of wireless communication will soon be able to make these scenarios real [3, 1].

In order to develop enabling technologies to realize the above scenarios, a crucial component is to maintain up-to-date information about the location of moving vehicles on the road. There could be thousands of cars on a small segment of a highway at any given time of a day. And of course, they are moving continuously unless there is an accident or heavy traffic congestion on their paths.

---

<sup>\*</sup> This research was partially supported by the NSF under grant numbers EIA98-18320, IIS98-17432 and IIS99-70700.

Updating their current locations once per second causes thousands of update transactions per second, not to mention the query transactions. Therefore, keeping track of each and every car's current location in real time is very hard to achieve if not impossible. However, instead of updating continuously, if we have a way of predicting the current locations of moving objects using some static information they provide either when they enter into the highway system or when they decide to get on the highway sometime in the future, then answering queries such as the above may become feasible.

Although we use cars on a highway as an example, any objects with some intelligent equipment such as airplanes or cellular phone users fall into the same category so that we can apply this technology to other applications. In this paper, we analyze the problem and propose a data model which abstracts moving objects as line segments and develop an efficient indexing technique explicitly designed for moving objects. The paper is organized as follows: Section 2 formalizes and reviews related work. In Section 3, we categorize all possible configurations so that we can pinpoint the best possible configuration on which we can build an efficient index structure. In Section 4, we give the performance results. Lastly, Section 5 gives conclusions and future work.

## 2 Problem Statement and Related Work

Objects such as automobiles, cellular phone users, and air planes change their locations continuously; we will use the term *moving objects* to refer to such entities. The problem we analyze and solve in this paper is the following: how to model moving objects so that we can store, index, and retrieve their future locations in an efficient manner. In particular, we are interested in answering a typical range query such as:  $Q = \text{return all moving objects which would be in a section of a highway at some time in the future}$ . Each moving object in a one dimensional space has four independent variables which allow us to predict its future location: a starting location( $s$ ), a destination( $e$ ), a starting time( $t_s$ ), and an initial velocity( $v_0$ ). We assume that a moving object will notify the database system whenever it changes one of these four variables. Updating the database will be done by deleting first and inserting it again.

We can think of a moving object as a point in a four dimensional space. However, modeling a moving object this way is not appropriate for our purpose since first, they do not necessarily preserve the proximity in the real world meaning that two objects near each other in the real world may not be close in the four dimensional space. Second, indexing in a higher dimensional space is not as efficient as in a lower dimensional space.

Another way of modeling moving objects is to draw its expected trajectory in a two dimensional space with time and location as the two axes, and its trajectory becomes a line segment. Thus, we can also model a moving object as a line segment in two dimensional space. Then a typical query  $Q$  becomes a range query with range  $[t_1, t_2] \times [y_1, y_2]$ , in other words, find all moving objects that are between the location  $y_1$  and  $y_2$  during the time  $t_1$  and  $t_2$ . The query

can be stated as follows: from a line segments database, find all line segments which intersect the query rectangle  $R([t_1, t_2] \times [y_1, y_2])$ .

One critical assumption that affects the definition of the problem is the notion of registration (or reservation) meaning that before you actually get a service whether it is a highway or an airplane, you need to tell the system your schedule. Without registration, predicting congestion would be difficult. For instance, if we predict traffic congestion only based on the current information, then there would be no congestion whatsoever if the prediction is based on the information that exists in the database at 5 a.m. With the assumption of registration, we can use the result set of this query to predict traffic congestion or to guarantee the fixed travel time. Without it, we can predict at least how long and until which exit the congestion stays so that we can flexibly open or close exits in between.

Jagadish [10] considered the problem of indexing line segments that (i) go through a specified point or (ii) intersect a specified line segment. He first extends a line segment to an infinite line, gets the line equation, and uses the slope and  $y$ -intercept to transform a line into a point in a two dimensional space called the dual space. The transformation is known as Hough transformation [9]. It seems to have good storage utilization, however, when a line segment is transformed to a point in a dual space, half of the information is lost because a line segment has four parameters whereas an infinite line has only two parameters. Hoel and Samet [8] used PMR Quadrees [13] to store and index line segments. They focused on queries such as finding a nearest line segment from a specified point. In many cases, they have to store information for a line segment over and over again which leads to significant storage overhead.

In [16], Sistla et al. proposed a data model called Moving Objects Spatio-Temporal(MOST) for representing moving objects. In the MOST data model, the concept of dynamic attributes is introduced, which consists of three sub-attributes, update value, update time, and function. Each moving object is assigned a dynamic attribute which is used to get its current location. They also proposed Future Temporal Logic as the query language for the MOST data model which is built on top of existing DBMS and takes advantage of the dynamic attributes. Wolfson et al. [19] addressed the uncertainty issues which determine the frequency with which the database has to update the locations of moving objects to provide a bound on the error. Erwig et al. [6] described an approach to model moving and evolving spatial objects introducing data types for moving points and moving regions together with a set of operations on such entities. They do not propose a specific design of such types and operations or a formal definition of their semantics. Instead, they give an outline of the work that should be done. They also developed an SQL-like query language with some new types of operators. However, their work concentrates on the queries for the current and the past information. Pfooser and Jensen [14] assumed that the positions of moving objects are tracked by using Global Positioning System and discussed sampling errors and uncertainty. As proposed by Erwig et al. [6], they only handled current and past information.

To the best of our knowledge, [17] is the first work that addresses the issue of indexing moving objects to query their future positions. They present a method to index moving objects using the PMR quadtree which shares the same drawbacks as the method proposed by Hoel et al. [8]. Kollios et al. [11] have proposed two methods that index moving objects in one dimensional space, and they have developed extensions for indexing moving objects in two dimensional space. One of the two methods uses a point access method. The other is a query approximation algorithm that uses multiple  $B^+$ -Trees. We will be comparing the methods proposed in this paper with those in [11]. Recently, Saltenis et al. [15] proposed a variant of  $R^*$ -Tree [2], Time-Parameterized R-Tree(TPR-Tree) to index the current and future positions of moving objects. Minimum Bounding Rectangles(MBR) of TPR-Trees grow or shrink to enclose containing moving objects. A moving object is assigned to an MBR so that the size of the MBR is minimum in some point of time when most of the queries will arrive.

### 3 Models

As stated earlier, a moving object has four independent variables,  $\langle s, e, t_s, v_0 \rangle$ . In this section, we investigate and categorize the configurations where two variables can vary so that we can choose the best possible configuration on which we can build an efficient indexing structure. The configurations where only one variable can vary are quite simple and not realistic [5].

#### 3.1 Moving Objects with Two Degree of Freedom

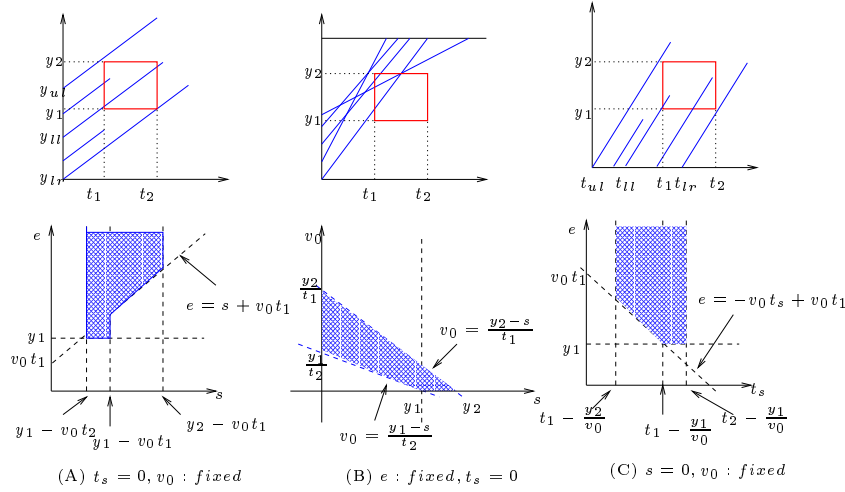
In this section, we consider scenarios in which two of the four parameters are constants. With four parameters, there are 6 different combinations to consider. Due to the space limit, we enumerate only three of them with the figures(Figure 1) in the original space as well as in the dual space. In the figure, the upper graph shows the line segments in the original space and the shaded area of the lower graph is the result set of the range query. The reader can find all the enumerations and the figures in [5]. Since the  $y$  location starts from zero, the constraint on  $s$  and  $e$  is that they should be greater than or equal to zero.

The first combination (A) is the case where  $s$  and  $e$  can vary, in other words, all moving objects start at the same time with the same velocity. First, check the conditions on  $s$ . Define  $y_{lr}$  as the  $y$ -intercept of the line which passes through  $(t_2, y_1)$  and  $y_{ul}$  as the  $y$ -intercept of the line which passes through  $(t_1, y_2)$ . Then for a line segment to intersect with the query rectangle  $R$ , the  $y$ -intercept of the line should be greater than or equal to  $y_{lr}$  and less than or equal to  $y_{ul}$ . The equation of the infinite line extended from the line segment which passes through  $(t_1, y_2)$  is  $y = v_0 t + y_{ul}$ . If we substitute  $(t_1, y_2)$  for  $(t, y)$ , we get  $y_{ul} = y_2 - v_0 t_1$ . In the same fashion, we can get  $y_{lr} = y_1 - v_0 t_2$ . The  $y$ -intercept of the line segment which passes  $(t_1, y_1)$ , say  $y_{ll}$ , is  $y_1 - v_0 t_1$ . Second, check the conditions on  $e$ . For the line segment which passes the horizontal line between  $[t_1, t_2]$ , the destination( $e$ ) being greater than or equal to  $y_1$  is enough to intersect with the

query rectangle  $R$ . However, that is not enough for one whose  $y$ -intercept is in between  $y_{lr}$  and  $y_{ul}$ . It should cross the vertical line at  $t_1$ . The  $e$  value of such a line at  $t_1$  is  $v_0 t_1 + s$  for a fixed  $s$ . Therefore, we have one more condition on  $e$ ,  $s + v_0 t_1 \leq e$ . See Figure 1(A). In summary, a moving object  $\langle s, e, t_s, v_0 \rangle$  where  $s$  and  $e$  can vary should satisfy the following conditions to be included in the result set for the query  $R([t_1, t_2] \times [y_1, y_2])$ :

$$y_1 - v_0 t_2 \leq s \leq y_2 - v_0 t_1, \quad (1)$$

$$e \geq \begin{cases} y_1 & \text{if } y_1 - v_0 t_2 \leq s \leq y_1 - v_0 t_1, \\ s + v_0 t_1 & \text{if } y_1 - v_0 t_1 < s \leq y_2 - v_0 t_1. \end{cases} \quad (2)$$



**Fig. 1.** Two parameters vary

The next combination (B) is the case where  $s$  and  $v_0$  can vary. In this combination, all moving objects start at the same time to the same destination, but with different starting locations and different velocities. Suppose  $s$  is fixed, and define  $v_{lr}$  as the slope of the line which passes through  $(t_2, y_1)$  and whose  $y$ -intercept is  $s$ . Likewise,  $v_{ul}$  as the slope of the line which passes through  $(t_1, y_2)$  and whose  $y$ -intercept is  $s$ . Then, for a line segment with  $s$  less than  $y_1$  to intersect with the query rectangle  $R$ , its slope should be greater than or equal to  $v_{lr}$ , and less than or equal to  $v_{ul}$ . The equation of the line which passes a fixed  $s$  is  $y = v_0 t + s$ , and the slope of such line which passes through  $(t_1, y_2)$  is  $\frac{y_2 - s}{t_1}$ , and the slope of such line which passes through  $(t_2, y_1)$  is  $\frac{y_1 - s}{t_2}$ . Therefore, we have  $v_{lr} = \frac{y_1 - s}{t_2}$  and  $v_{ul} = \frac{y_2 - s}{t_1}$ . For  $s$  greater than  $y_1$ , the slope being less than or equal to  $\frac{y_2 - s}{t_1}$  is enough to intersect with the query rectangle  $R$ , assuming the

slope should be greater than zero. See Figure 1(B). In summary, a moving object  $\langle s, e, t_s, v_0 \rangle$  where  $s$  and  $v_0$  can vary should satisfy the following conditions to be included in the result set for the query  $R([t_1, t_2] \times [y_1, y_2])$ :

$$\frac{y_1 - s}{t_2} \leq v_0 \leq \frac{y_2 - s}{t_1} \text{ if } s \leq y_1, \quad (3)$$

$$v_0 \leq \frac{y_2 - s}{t_1} \text{ otherwise .} \quad (4)$$

In combination (C) where  $e$  and  $t_s$  can vary, all moving objects start from the same location with the same velocity. Define  $t_{ul}$  as the  $t$ -intercept of the line which passes through  $(t_1, y_2)$  and  $t_{lr}$  as the  $t$ -intercept of the line which passes through  $(t_2, y_1)$ . For a line segment to intersect with the query rectangle  $R$ , their  $t$ -intercept( $t_s$ ) should be greater than or equal to  $t_{ul}$  and less than or equal to  $t_{lr}$ . The equation of the infinite line extended from the line segment which passes through  $(t_1, y_2)$  is  $y - y_2 = v_0(t - t_1)$ . If we substitute  $(t_{ul}, 0)$  for  $(t, y)$ , we get  $t_{ul} = t_1 - \frac{y_2}{v_0}$ . Similarly, we can get the  $t$ -intercept of the line which passes through  $(t_2, y_1)$ ,  $t_{lr} = t_2 - \frac{y_1}{v_0}$ . Besides,  $t$ -intercept of the line which passes through  $(t_1, y_1)$  is  $t_1 - \frac{y_1}{v_0}$ , say  $t_{ll}$ . Now, let us consider the bounds on  $e$ . For a line segment whose  $t$ -intercept is greater than  $t_{ll}$  and less than  $t_{lr}$ , i.e. the line segment which passes the horizontal line between  $t_1$  and  $t_2$ ,  $e$  being greater than  $y_1$  is enough for it to intersect with the query rectangle  $R$ . However, the line segment for which  $t$ -intercept is in between  $t_{ul}$  and  $t_{ll}$  should cross the vertical line at  $t_1$ . The equation of the infinite line which passes through a point  $(t_1, e)$  is  $y - e = v_0(t - t_1)$ . If we substitute  $(t_s, 0)$  for  $(t, y)$ , we get the  $e$  value of the line at  $t_1$ ,  $e = -v_0 t_s + v_0 t_1$ . See Figure 1(C). In summary, a moving object  $\langle s, e, t_s, v_0 \rangle$  where  $e$  and  $t_s$  can vary should satisfy the following conditions to be included in the result set for the query  $R([t_1, t_2] \times [y_1, y_2])$ :

$$t_1 - \frac{y_2}{v_0} \leq t_s \leq t_2 - \frac{y_1}{v_0}, \quad (5)$$

$$e \geq \begin{cases} -v_0 t_s + v_0 t_1 & \text{if } t_1 - \frac{y_2}{v_0} \leq t_s \leq t_1 - \frac{y_1}{v_0}, \\ y_1 & \text{if } t_1 - \frac{y_1}{v_0} < t_s \leq t_2 - \frac{y_1}{v_0}. \end{cases} \quad (6)$$

We have considered three of six possible combinations and summarized them in Figure 1. An interesting thing to note is that in the dual space the shapes of the result sets of the range query differ from each other, and the shapes are irregular, meaning that they are not rectangular or circular. The reason we care about the shape being rectangular or circular is that most popular multidimensional index structures such as  $R^*$ -Trees [2] or SS-Trees [18] are based on the fact that the shape of the range queries is either rectangular or circular.

### 3.2 SV Model

In this section, we analyze each of the parameters( $s, e, t_s$ , and  $v_0$ ) and choose the most suitable configuration for building an efficient index structure and realistic combination of the six possible cases as our data model.

We start by considering a simple transformation that can be used to generalize the problem from one with fixed starting location  $s$  to one with variable  $s$ . We can transform an object with variable nonzero  $s$  and  $t_s$  into one with  $s = 0$  and new  $t_s$  as follows,

$$newT_s = -s/v_0 + t_s . \quad (7)$$

With Transformation (7), we represent a moving object with any starting point as one that starts from the origin. Therefore, solving the limited problem where  $s = 0$  is the same as solving the general problem where  $s$  could be anywhere, meaning that we can have constant  $s$  and variable  $t_s$  without restricting the problem. If we visualize the conversion, it corresponds to extending a line segment to the  $t$  axis and getting the new  $t_s$  coordinate. This conversion introduces some overhead if we ask a query about the past. For example, if a moving object,  $m$ , starts to move from  $s = 10$  at time  $t_s = 10$  with velocity  $v_0 = 1$ , then the  $newT_s$  by Transformation (7) is 0. Next, if we ask a query about the vehicles in the time period between 3 and 5,  $m$  would come up as a result, although  $m$  does not exist at that time. This is an example of the overhead that we could have avoided if we do not convert  $t_s$  to  $newT_s$ . However, this is not a significant problem since we need the filtering when we query the past. In this case, quick response time is not as important as in the real time system.

Next, consider the destination parameter  $e$ . When  $e$  varies, there are two ways to convert the variable  $e$  to some fixed  $e$ . One is to set it to some constant and apply some filtering technique at the end of every query, which could introduce significant overheads. The other is to move the line segment so that the  $y$  coordinate of the right end point( $e$ ) of the line segment becomes a fixed constant. In this case, two other parameters( $s$  and  $t_s$ ) change correspondingly, which could make the problem more complex. In any case, converting variable  $e$  to some fixed constant is not helpful to solve the problem. Therefore,  $e$  should be a variable, which leaves us no choice except  $v_0$  being constant. However, requiring constant  $v_0$  is actually not a significant constraint. In real world highways with vehicles equipped with discrete speedometers, the number of different velocities is actually not so large. We can therefore easily create multiple instances of fixed velocity models for any given stretch of a highway under consideration. The more instances we have, the more precision we would gain. However, for a moving object whose velocity does not match one of the instances, we can insert it into the one which has the closest velocity and make it update more often than other objects whose velocity match exactly an instance.

Among the six combinations, (C) fits in with our purpose of choosing a configuration that has the most realistic assumptions and at the same time captures the abstraction of the problem well. Since (C) assumes  $s = 0$ , we can apply Transformation(7) without restricting the problem. Secondly, as we have discussed above, (C) assumes variable  $e$  and constant  $v_0$ . We call this combination SV model(fixed  $s, v$ ) and claim that this model captures the abstraction of the problem well. In the next section, we support this claim by considering an experimental evaluation with realistic database workloads.

## 4 Performance

In this section, we introduce the SS-Tree [18] as the underlying index structure, explain experimental settings, and show the result of a performance study.

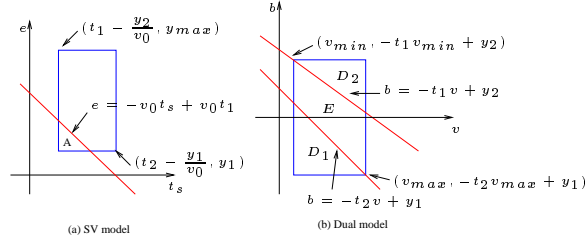
We use the SS-Tree [18] as the underlying index structure. The SS-Tree is a high-dimensional search tree proposed by White and Jain [18] for similarity searches such as k-nearest neighbor query. It has been shown to outperform the  $R^*$ -Tree [2] in high-dimensional k-nearest neighbor query. We expect the SS-Tree to perform better also for range queries than the  $R^*$ -Tree due to the same reasons.

As discussed earlier, we can cluster moving objects with the same velocity by associating them with an instance of the SS-Tree. Another benefit of using the SV model is that the shape of the result set in the SV model is close to rectangular (as shown in Figure 1(C)) which is easier to handle since many index structures (e.g.,  $R^*$ -Tree variants) use it as a result set of a range query.

The SV model requires an index structure for a fixed velocity and we assume that there are only 5 different velocities, therefore we need 5 SS-Trees for the SV model. Each SS-Tree for the SV model is associated with a fixed velocity,  $v = 0.8, 0.9, \dots, 1.2$ . The Dual transform model does not require multiple instances of an index structure, therefore, only one SS-Tree is used.

### 4.1 Experimental Setting and Uniform Data Distribution

In this section, we describe the experimental setting, the data sets and the metrics used. We first generate  $N$  objects, where  $N = 10000, 20000, \dots, 80000$ . Each object has 5 variables: a unique ID ( $id$ ), a starting location ( $s$ ), a destination ( $e$ ), a starting time ( $t_s$ ), and a velocity ( $v_0$ ). We assign a sequence number to each object as its ID and set the starting location to zero. After randomly generating two numbers in the range  $[0, 500]$ , we assign one to the destination and the other to the starting time. We maintain two lists, one sorted based on  $t_s$  and the other sorted on the time when a moving object reaches the destination and is supposed to remove itself from the system. The time to remove is  $t_e = \frac{e-s}{v_0} + t_s$ . The system starts the clock running from 0 to 500. At each tick, it inserts the objects which are supposed to start at that time, and deletes the objects which are supposed to reach their destinations by that time. To start with, the system is run for the first 20% of the time. A new object is inserted into the SV model by inserting into the SS-Tree corresponding to the same velocity as the object's together with  $(id, t_s, e)$ . In the Dual transform model, the object is inserted into the SS-Tree with  $id$ , the velocity,  $v_0$ , and the  $y$ -intercept,  $-v_0 \times t_s$ . After that, at every other tick, we run 5 random queries  $([t_1, t_2] \times [y_1, y_2])$  against the two models and collect the results. If the width  $(t_2 - t_1)$  is 5, we call the query a 1% query and if it is 50, a 10% query.  $t_1$  should be greater than the current tick value. For a given query, the SV model first converts the rectangle to another rectangle as follows (See Figure 2(a)):  $(t_1, t_2) \rightarrow (t_1 - \frac{y_2}{v_0}, t_2 - \frac{y_1}{v_0})$ ,  $(y_1, y_2) \rightarrow (y_1, y_{max})$ . With the converted rectangle, the SV model runs the query against the 5 SS-Trees and merges all the results. The Dual transform model also converts the



**Fig. 2.** Two Models

rectangle to another one as follows (See Figure 2(b)):  $(t_1, t_2) \rightarrow (v_{min}, v_{max})$ ,  $(y_1, y_2) \rightarrow (-t_2 v_{max} + y_1, -t_1 v_{min} + y_2)$ .

The overhead of the SV model is the number of objects for which  $(t_s, e)$  satisfies  $e < -v_0 t_s + v_0 t_1$ , where  $v_0$  is the velocity of the corresponding SS-Tree, i.e., the number of objects which are in the area  $A$  in Figure 2(a). We refer to this as the *SV overhead*. The rest of the objects are the exact answers.

Before we identify different overheads in the Dual transform model, we need to take a closer look at [11] in which Kollios et al. propose a query approximation algorithm using multiple  $B^+$ -Trees. Kollios et al. noticed that depending on where the origin is considered, the  $y$ -intercept of the corresponding line segment can be different, resulting in the areas  $D_1$  and  $D_2$  in Figure 2(b). Multiple  $B^+$ -Trees are kept, which contain the same information about the moving objects but use different origin values. When a query arrives, the sizes of the area  $D_1 + D_2$  are computed from all indices to choose the one with the minimal size, and process the query against that index structure. In this fashion, the overhead associated with the areas  $D_1$  and  $D_2$  is minimized. We call this the *index overhead* which is the number of objects that would not have been retrieved if the rectangle is not used to retrieve the answer set. Since we use one multidimensional index structure the size of the areas  $D_1$  and  $D_2$  that is generated is an upper bound of what is incurred in [11]. For the comparison, we define another kind of overhead that cannot be avoided even if the size of the area  $D_1 + D_2$  is reduced to 0 by using multiple  $B^+$ -Trees. That overhead is the number of objects that are inside  $E$  but not the exact answer. We call this the *dual transformation overhead*. This *dual transformation overhead* is introduced because the Dual transform model loses half of the information of the object when it transforms the original space to the dual space.

We now compare the percentage of the three kinds of overheads as we vary the size of the objects or the width of the query rectangle. As observed in Figure 3(a), the percentage of the *SV overhead* is less than 1% in any size of the data when the width is 5(1% query), and less than 4% in any size of the data when the width is 50(10% query). The *dual transformation overhead* is around 25% in 1% query and around 22% in 10% query.

Figure 3(b) shows that as the width gets larger, the *SV overhead* becomes bigger and the overheads due to the Dual transform model become smaller. The

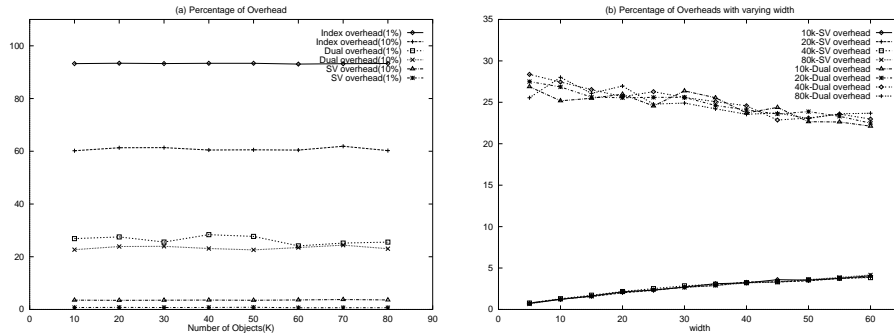


Fig. 3. Percentage of overheads with varying data size

reason is that as the width grows, the number of objects in the exact answer set also grows, so does the size of the area  $A$ . The result set of the Dual transform model also grows as the width grows, however, it already includes too many false hits to start with so that the rate at which it introduces the new false hits to the growing result set is a lot less than the rate at which previous overhead becomes exact answers. Even if we have an ideal algorithm that returns the objects in the area  $E$  without any extra work in terms of the I/O, the Dual transform model would still have 20% overheads compared to less than 4% as in the case of the SV model. In any case, the SV model outperforms the Dual transform model in terms of additional I/O.

## 4.2 Simulated Realistic Data

In 1990, the Santa Barbara County Association of Governments published the document [12] that is a study of Route 101 from the Hollister exit to the county line in the Santa Barbara area which is about 30 miles long. The purpose of the study is to review existing conditions and programmed improvements and to forecast travel for the year 2000. Directly from the document are the name of the exits, the lengths between exits, the annual growth of the sections until the year 2000, approximate traffic volume per lane per hour, and the number of lanes in each section. From that information, we calculate the projected traffic volume at the year 2000 and multiply the number of lanes and get the numbers for the traffic per segment per hour at peak time. The average speeds are roughly around 30 miles per hour. What we need for simulating the SV model are each car's starting time and destination along with its velocity. We also need to take its starting location into account although the SV model assumes that it is zero. Therefore, we have to use our knowledge about the neighborhoods and highway exits of the area. First, we assign a *getoff* and a *geton* rate to each exit. It is a combination of our experience and the data from [12]. If the traffic becomes less at an exit, then more cars get off than get on the exit and vice versa. We distribute the cars which get on an entry to the upcoming exits based on the *getoff* rates of each exit. And we make some number of cars get on an exit based

on the *geton* rates and randomly choose the starting times of each car from one hour interval. After applying this scheme, the numbers we derive is similar in the sense that when the exact numbers change, they also change with similar rate. The readers can find the numbers in [5]. The test setup is identical to the previous subsection. Figure 4 shows the results. Although the exact numbers are a little bit different from the uniformly distributed data, the general trend of the results complies with earlier findings.

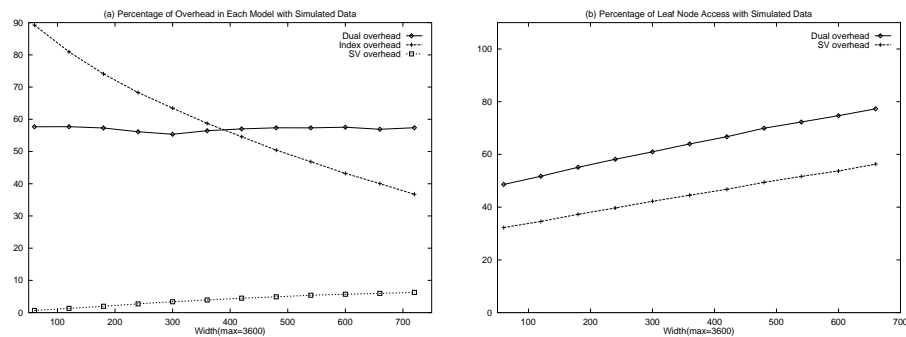


Fig. 4. Performance with realistic data distribution

## 5 Conclusion and Future Work

A moving object moves with four independent parameters,  $\langle s, e, t_s, v_0 \rangle$ . We considered the scenarios where not all parameters are variables. There are 6 different combinations when two parameters can vary, and we analyzed all of them. Among them, we chose the SV model where  $s = 0$  and  $v_0$  is fixed and  $t_s, e$  can vary because it offers the most realistic setting for moving objects and we can build an efficient multidimensional index structure on it. The performance test showed that the SV model has significantly lower overhead when compared to the Dual transform model [10, 11]. Although we used the SS-Tree as the underlying index structure, any other index structure would work because the SV model does not depend on the underlying index structure.

## References

1. I. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu, and W. Wang. Mobility Management in Current and Future Communications Networks. *IEEE Network*, pages 39–49, 1998.
2. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The  $R^*$ -Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

3. K. Buchanan, R. Fudge, D. McFarlane, T. Phillips, A. Sasaki, and H. Xia. IMT-2000: Service Provider's Perspective. *IEEE Personal Communications*, 1997.
4. CalTrans. Advanced Transportation Systems program Plan. <http://www.dot.ca.gov/hq/newtech>, 1996.
5. H. Chon, D. Agrawal, and A. El-Abbadi. Storage and Retrieval of Moving Objects. Technical Report TRCS00-21, Department of Computer Science, University of California, Santa Barbara, 2000.
6. M. Erwig, R. H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. In *GeoInformatica*, volume 3, 1999.
7. B. Foreman. A Survey of Wireless Communications Technologies for Automated Vehicle Control. In *Future Transportation Technology Conf. and Exposition*, Costa Mesa, CA, 1995.
8. E. Hoel and H. Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *Advances in Spatial Database - 2nd Symposium*, 1991.
9. P.V.C. Hough. Method and Means for Recognizing Complex Patterns. *U.S. Patent No. 3069654*, 1962.
10. H. V. Jagadish. On Indexing Line Segments. In *Proceedings of the Int. Conf. on Very Large Data Bases*, Brisbane, Australia, 1990.
11. G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Moving Objects. In *Proceedings of ACM Symp. on Principles of Database Systems*, 1999.
12. G. Lorden, J. Kemp, D. Daskal, A. Terry, and N. Ramirez. *South Coast route 101 corridor study*. Santa Barbara County Association of Governments, 1990.
13. R. Nelson and H. Samet. A Consistent Hierarchical Representation for Vector Data. In *ACM SIGGRAPH*, pages 197–206, 1986.
14. D. Pfoser and C. S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. of the SSDBM Conf.*, pages 111–132, 1999.
15. S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 331–342, Dallas, Texas, 2000.
16. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and Querying Moving Objects. In *Proceedings of the Int. Conf. on Data Engineering*, 1997.
17. J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method. In *Proceedings of ACM Symp. on Principles of Database Systems*, 1998.
18. D. White and R. Jain. Similarity Indexing with the SS-Tree. In *Proceedings of the Int. Conf. on Data Engineering*, 1996.
19. O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, 1998.