Storage and Retrieval of Moving Objects

Hae Don Chon, Divy Agrawal, Amr El Abbadi
Department of Computer Science
University of California, Santa Barbara
{hdchon,agrawal,amr}@cs.ucsb.edu

Abstract

Information management for advanced transportation systems is significantly complex, since it requires maintaining dynamically changing information associated with moving objects. Updating current locations of moving objects every time they move is not a realistic solution. In this paper, we investigate the problem and provide a data model with which we can store, index, and retrieve 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 only one or 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. After developing a data model, we conduct performance study and show 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 and enjoy a cup of cappuccino 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. These are not scenarios which appear in science fiction anymore. There has been significant research in the area of Advanced Transportation Systems [5, 8] in the past decade and the next generation of wireless communication will soon be able to make these scenarios real [4, 2].

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. Updating their current locations once per second causes thousands of update transactions per second, not to mention the query transactions, which could cripple the server at the control center. 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 to illustrate the problem of maintaining information about moving objects, any device 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 involving moving objects. 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 3.3 we propose a data model and compare it with a traditional model. And in Section 4, we give the performance results. 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 = *return all moving objects which would be in a section of a highway at some time in the future.* We can view a moving object as an (id, location) pair. Each moving object has four independent variables which allow us to predict its future location: they are a starting location(s), a destination(e), a starting time(t_s), and an initial velocity(v_0). We assume that moving objects will notify the database system whenever they change one of these four variables. 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 representing the moving object becomes a line segment in time and location space. 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 $[t_1, t_2] \times [y_1, y_2]$ as shown in Figure 1, i.e., 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 (thereby, moving objects) which intersect the query rectangle $R([t_1, t_2] \times [y_1, y_2])$. The query rectangle R we refer to throughout the paper is the one with two points (t_1, y_1) and (t_2, y_2) as its diagonal points in the lower left and the upper right corners, respectively.

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, an airplane, or a cell phone, you need to tell the system your schedule. Without registration, predicting congestion would be difficult. For instance, if nobody books a flight but goes to the airport to fly, then an airline company may presume that no one is scheduled to fly, and hence, could cancel the flight. Or, if we predict traffic congestion only based on the current information, then there would be no congestion

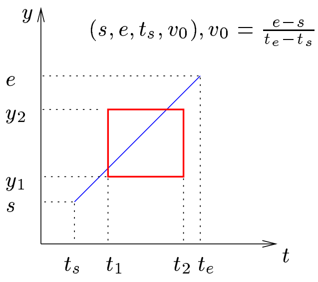


Figure 1: Typical Query

whatsoever if the prediction is based on the information that exists in the database at 5 a.m. or 6 a.m. Unfortunately, this does not account for the new objects that enter the database during the morning rush hour. Although this assumption is important from the perspective of application of moving object databases, it does not affect the development of our data model. 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.

Several researchers have investigated the problem of indexing line segments [12, 10, 1] in the field of line segments databases. Although they did not consider their work in the context of moving objects, their results are closely related to moving objects databases. Jagadish [12] 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 [11]. 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. Thus, additional filtering technique must be used at the end of every query to filter out redundant information. Hoel and Samet [10] used PMR Quadtrees [15] 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 [18], Sistla et al. proposed a data model called MOST(Moving Objects Spatio-Temporal) for representing moving objects in database systems. 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(FTL) 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. [21] 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. For instance, the difference between the actual location of a moving object and the location in the database increases as time elapses so that to set the maximum difference(e.g., 1 mile), each object has to update its location periodically(e.g., every 10 minutes). 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. Pfoser and Jensen [16] assumed that the positions of moving objects are tracked by using Global Positioning System(GPS) and discussed sampling errors and uncertainty. They provided a representation for moving objects using relational database. As proposed by Erwig et al. [6], they only handled current and past information.

To the best of our knowledge, [19] 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. [10]. Kollios et al. [13] 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 such as hB^{π}-Tree [7]. The other is a query approximation algorithm that uses multiple B^{+}-Trees. They indicate that for many applications, the relative positions of the moving objects do not change often so that they restrict their queries where the trajectory lines in the time-space plane do not cross. We will be comparing the methods proposed in this paper with those in [13]. Recently, Saltenis et al. [17] proposed a variant of R^{*}-Tree [3], 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 the most of queries will arrive. To utilize existing DBMS(relational or object-oriented), Forlizzi et al. [9] proposed a data model and data structures. They provided two example algorithms to illustrate the use of their model and structures.

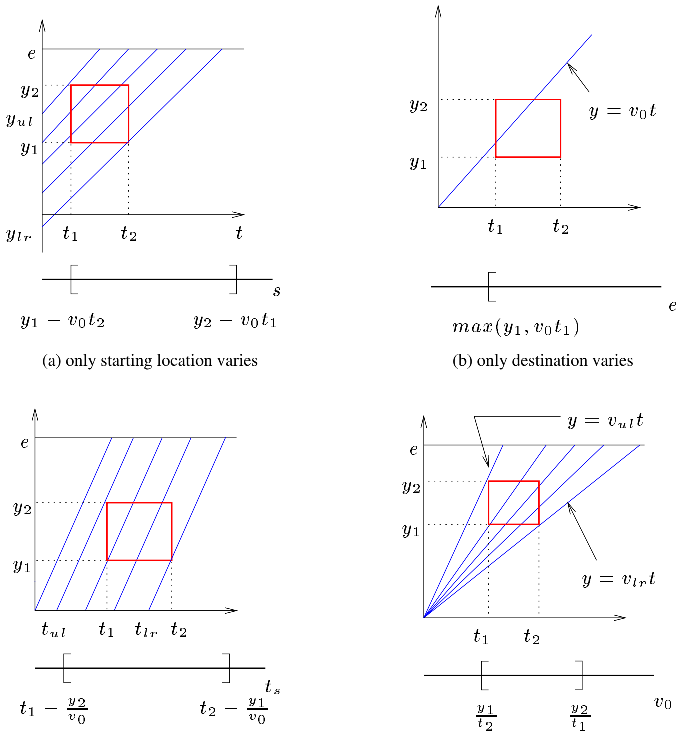
3 Models

As stated earlier, a moving object has four independent variables, $\langle s, e, t_s, v_0 \rangle$. What if only one variable can vary? What if two? To answer these questions, we investigate and categorize the configurations where only one variable can vary or two can vary so that we can choose the best possible configuration on which we can build an efficient indexing structure.

3.1 Moving Objects with Minimum Degree of Freedom

In order to understand the complexity of modeling and maintaining information about moving objects, we start by restricting the problem where the moving objects have only one degree of freedom. Assuming that a moving object is modeled as $\langle s, e, t_s, v_0 \rangle$, we examined the ramifications where only one of the variables is allowed to be varied. The main idea is to develop a systematic understanding of the underlying complexity of the problem. Figure 2 illustrates the variable configurations: (a) only s varies, (b) only e varies, (c) only t_s varies, and (d) only v_0 varies. We consider the query rectangle $R([t_1, t_2] \times [y_1, y_2])$.

In case (a), we assume that all moving objects start at the same time with the same velocity, and they have the same destination. They only differ in their starting locations(s). The line segments in the two dimensional space are all parallel to each other, the starting time t_s is 0, but their y-intercepts which are the starting location of the corresponding moving objects are different. 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) as shown in Figure 2(a). For a line segment to intersect with the query rectangle R, its y-intercept(s) should be greater than or equal to y_{lr} and less than or equal to y_{ul}. See Figure 2(a). The equation of the infinite line extended from the line segment which passes through



(a) only starting location varies

(b) only destination varies

(c) only starting time varies

(d) only velocity varies

Figure 2: Only one parameter varies

(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 $y_{lr} = y_1 - v_0 t_2$. As a result, s should be in between $y_1 - v_0 t_2$ and $y_2 - v_0 t_1$. However, in the case where either y_{lr} or both of y_{lr} and y_{ul} are less than 0, we take 0 as their value. Therefore, a moving object $\langle s, e, t_s, v_0 \rangle$ where s is the only degree of freedom is in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$ iff

$$max(0, y_1 - v_0 t_2) \leq s \leq max(0, y_2 - v_0 t_1) \ . \tag{1}$$

In case (b), all moving objects start from the same starting location at the same time with the same velocity. They only have different destinations(e). Therefore for a line segment to intersect with the query rectangle R, its destination should be greater than or equal to y_1 when the lower left corner of the query rectangle R is in the left side of the line $y = v_0 t$. However, if the lower left corner of the query rectangle R is in the right side as in Figure 2 (b), the line segment should cross the vertical line at t_1 to intersect with the query rectangle R, and the y coordinate of the line at t_1 is $v_0 t_1$. Therefore, a moving object $\langle s, e, t_s, v_0 \rangle$ where e is the only degree of freedom is in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$ iff

$$max(y_1, v_0 t_1) \leq e \ . \tag{2}$$

In case (c), all moving objects start from the same location with the same velocity to the same destination, but they start at different times(t_s). 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) as shown in Figure 2(c). 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}. See Figure 2(c). 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), $t_{lr} = t_2 - \frac{y_1}{v_0}$. Thus, a moving object $\langle s, e, t_s, v_0 \rangle$ where t_s is the only degree of freedom is in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$ iff

$$t_1 - \frac{y_2}{v_0} \le t_s \le t_2 - \frac{y_1}{v_0} \ . \tag{3}$$

Finally in case (d), all moving objects start at the same time from the same location to the same destination, but their velocities(v_0) are different. Define v_{ul} as the slope of the line which passes through (t_1, y_2) and v_{lr} as the slope of the line which passes through (t_2, y_1) as shown in Figure 2(d). For a line segment to intersect with the query rectangle R, its slope(v_0) should be greater than or equal to v_{lr} and less than or equal to v_{ul}. If we substitute (t_1, y_2) for (t, y) in the equation $y = v_{ul}t$, we get the $v_{ul} = \frac{y_2}{t_1}$. Likewise, we get $v_{lr} = \frac{y_1}{t_2}$. Therefore, a moving object $\langle s, e, t_s, v_0 \rangle$ where v_0 is the only degree of freedom is in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$ iff

$$\frac{y_1}{t_2} \le v_0 \le \frac{y_2}{t_1} \ . \tag{4}$$

As we have shown, all of the above four cases have intervals in one dimensional space as their result set of the range query. We can use any existing relational database system, and the B^+-Trees which seem to be the best possible solution for indexing in one dimensional space with a total ordering.

3.2 Moving Objects with Two Degree of Freedom

In this section, we consider scenarios in which two of the four parameters are constants. These are more realistic assumptions than the ones discussed in the previous subsection. With four parameters, there are 6 different combinations to consider. We enumerate all of them with the figures in the original space as well as in the dual space. Figure 3 illustrates all the possible combinations together with the result set of the range query. Since the y location starts from zero, the conditions on s and e should be greater than or equal to zero.

The first combination (A) is the combination of the cases (a) and (b) of the previous section where s and e can vary, in other words, all moving objects start at the same time with the same velocity. First, let us check the conditions on s. Define y_{lr} and y_{ul} as in Section 3.1. 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}. As we calculated in the previous subsection, $y_{lr} = y_1 - v_0 t_2$ and $y_{ul} = y_2 - v_0 t_1$. The y-intercept of the line segment which passes (t_1, y_1), say y_{ll}, is $y_1 - v_0 t_1$. Second, let us 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_{ll}. 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, $s + v_0 t_1 \le e$. See Figure 3(A). In the figure, the left graph shows the line segments in the original space and the shaded area of the right graph is the result set of the range query. In summary, a moving object $\langle s, e, t_s, v_0 \rangle$ where s and e can vary should satisfy the following

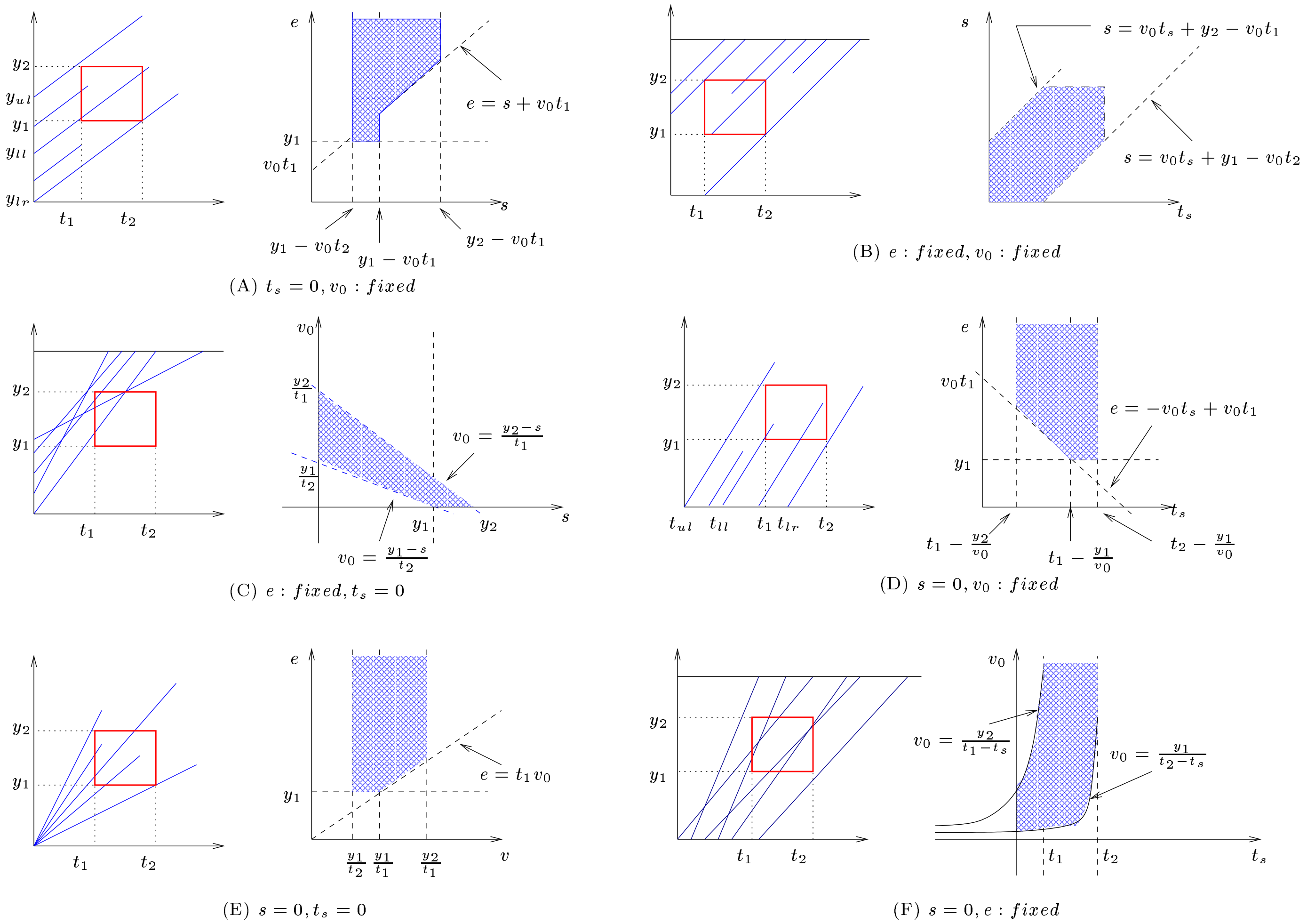


Figure 3: Two parameters vary

conditions to be in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$:

$$y_1 - v_0 t_2 \leq s \leq y_2 - v_0 t_1, \tag{5}$$

$$e \geq \begin{cases} y_1 & \text{if } y_1 - v_0 t_2 \leq s \leq y_1 - v_0 t_1, \\ s + v_0 t_1 & \text{if } y_1 - v_0 t_1 < s \leq y_2 - v_0 t_1 \ . \end{cases} \tag{6}$$

In the combination (B), s and t_s can vary, which is the combination of the cases (a) and (c) of Section 3.1. In other words, all moving objects have the same velocity and the same destination. For a line segment to intersect with the query rectangle R, the starting point(lower left point) of the line segment should be in between the line passing (t_1, y_2) and the one passing (t_2, y_1). The two new conditions are that the starting location(s) of a moving object should be less than y_2 and the starting time(t_s) should be less than t_2. See Figure 3(B). In summary, a moving object $\langle s, e, t_s, v_0 \rangle$ where s and t_s can vary should satisfy the following conditions to be in the result set for the query

$R([t_1, t_2] \times [y_1, y_2])$:

$$v_0 t_s + y_1 - v_0 t_2 \le s \le v_0 t_s + y_2 - v_0 t_1, \tag{7}$$
$$s \le y_2, \tag{8}$$
$$t_s \le t_2 \ . \tag{9}$$

Let us now consider the combination (C) which is the combination of the cases (a) and (d) of Section 3.1 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 similar to the case (d), 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 3(C). 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 in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$:

$$\frac{y_1 - s}{t_2} \le v_0 \le \frac{y_2 - s}{t_1} \text{ if } s \le y_1, \tag{10}$$
$$v_0 \le \frac{y_2 - s}{t_1} \text{ otherwise } . \tag{11}$$

In the combination (D) which is the combination of the cases (b) and (c) where e and t_s can vary, all moving objects start from the same location with the same velocity. As we did in Section 3.1, we can get the lower and the upper bound on t_s which are $t_{ul} = t_1 - \frac{y_2}{v_0}$ and $t_{lr} = t_2 - \frac{y_1}{v_0}$, respectively. 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 3(D). 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 in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$:

$$t_1 - \frac{y_2}{v_0} \le t_s \le t_2 - \frac{y_1}{v_0}, \tag{12}$$

$$e \ge \begin{cases} -v_0 t_s + v_0 t_1 & \text{if } t_1 - \frac{y_2}{v_0} \le t_s \le t_1 - \frac{y_1}{v_0}, \\ y_1 & \text{if } t_1 - \frac{y_1}{v_0} < t_s \le t_2 - \frac{y_1}{v_0} \ . \end{cases} \tag{13}$$

In the combination (E) which is the combination of the cases (b) and (d) where e and v_0 can vary, all moving objects start from the same location at the same time. As in the case (d), define v_{lr} and v_{ul} as the slopes of the lines which pass through (t_2, y_1) and (t_1, y_2), respectively. Then, for a line segment to intersect with the query rectangle R, its velocity(v_0) should be greater than or equal to

$v_{lr} = \frac{y_1}{t_2}$ and less than or equal to $v_{ul} = \frac{y_2}{t_1}$. The slope of the line which passes through (t_1, y_1) is $\frac{y_1}{t_1}$, say v_{ll}. Now, consider the bounds on the destination(e). For the line segment whose slope is greater than or equal to v_{lr} and less than or equal to v_{ll}, i.e. the line segment which passes the horizontal line between t_1 and t_2, its destination being greater than y_1 is enough for it to intersect with the query rectangle R. However, that is not the case for one with the slope greater than y_{ll}. Such line segment should cross the vertical line at t_1. e value of such line at t_1 is $v_0 t_1$. Therefore, we have one more bound on e, $e \geq t_1 v_0$. See Figure 3(E). In summary, a moving object $\langle s, e, t_s, v_0 \rangle$ where e and v_0 can vary should satisfy the following conditions to be in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$:

$$\frac{y_1}{t_2} \leq v_0 \leq \frac{y_2}{t_1}, \tag{14}$$

$$e \geq \begin{cases} y_1 & \text{if } \frac{y_1}{t_2} \leq v_0 \leq \frac{y_1}{t_1}, \\ t_1 v_0 & \text{if } \frac{y_1}{t_1} < v_0 \leq \frac{y_2}{t_1} \end{cases}. \tag{15}$$

In the last combination (F) which is the combination of the cases (c) and (d), all moving objects have the same starting location and the same destination. Suppose t_s is fixed and less than t_1, and define v_{lr} as the slope of the line which passes through (t_2, y_1) and whose t-intercept is t_s. In the same fashion, define v_{ul} as the slope of the line which passes through (t_1, y_2) and whose t-intercept is t_s. Then, for a line segment with t_s less then t_1 to intersect with the query rectangle R, the slope of the line should be greater than or equal to v_{lr} and less than or equal to v_{ul}. Similar to the case (C), we have $v_{lr} = \frac{y_1}{t_2 - t_s}$ and $v_{ul} = \frac{y_2}{t_1 - t_s}$. For t_s greater than t_1, the slope being greater than or equal to $\frac{y_1}{t_2 - t_s}$ is enough to intersect with the query rectangle R. See Figure 3(F). In summary, a moving object $\langle s, e, t_s, v_0 \rangle$ where t_s and v_0 can vary should satisfy the following conditions to be in the result set for the query $R([t_1, t_2] \times [y_1, y_2])$:

$$\frac{y_1}{t_2 - t_s} \leq v_0 \leq \frac{y_2}{t_1 - t_s} \text{ if } t_s \leq t_1, \tag{16}$$

$$\frac{y_1}{t_2 - t_s} \leq v_0 \text{ otherwise } . \tag{17}$$

We have considered all six possible combinations and summarized them in Figure 3. As can be seen in the figure, varying two parameters out of four is already complex enough to handle and therefore, in this paper, we do not investigate the cases where three or four parameters can vary. We leave that as future work. 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 [3] or SS-Trees [20] are based on the fact that the shape of the range queries is either rectangular or circular.

3.3 SV Model

We started by considering the four cases where only one parameter can vary, these cases solve only a very limited number of problems. We then went further to the cases where two out of the four parameters can vary. There are six such combinations and we investigated all of them and Figure 3 summarized our investigation. In this section, we analyze each of the parameters(s, e, t_s, and v_0) and choose the most suitable 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,

$$newT_s = -s/v_0 + t_s \ . \tag{18}$$

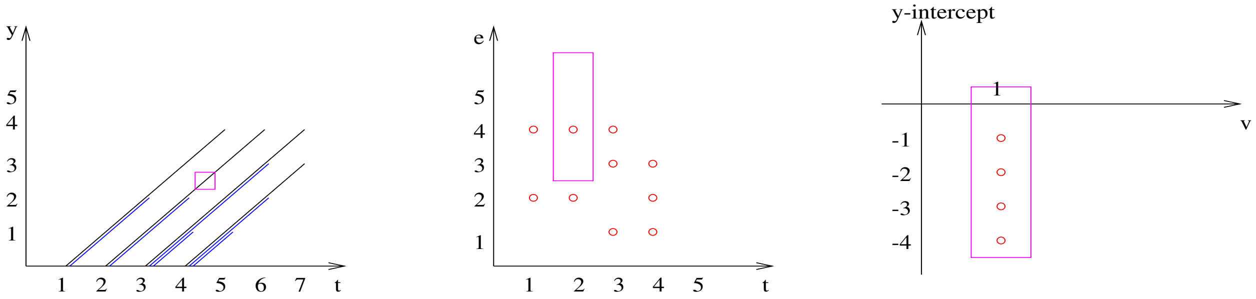
With Transformation (18), we represent a moving object with any starting point as one that starts from the origin. Therefore, solving the limited problem where $s = 0$ is the same as solving the general problem where s could be anywhere, meaning that we can have s being constant and t_s being variable 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 (18) 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 $newT_S$. To get the exact answers, we have to filter the answers at the end of every query. 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 where we need an answer to a query (about the present and the future) right away. In this paper, we are concerned about answering queries in realtime, therefore, we feel that the overheads introduced by this conversion are not so significant since they are incurred in answering queries about the past.

Next, let us consider the destination parameter e. When e varies, there are two ways to convert the variable e to some fixed e. One is just 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 (for example, 40, 45, 50,..., 70, 75). 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, (D) 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 (D) assumes $s = 0$, we can apply Transformation(18) without restricting the problem. Secondly, as we have discussed above, (D) 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.

3.4 SV Model versus Dual Transform Model

We give an example to compare the SV model to the Dual transform model which is a traditional model that transforms a line segment to a point in a dual space using the Hough-X transformation [12, 13]. The Hough-X transformation [11] maps a (infinite) line written in the form $y = ax + b$ to a point



(a) original space (b) SV model (c) Dual model

Figure 4: Example

(a, b) in a transformed space(called dual space), where a is the slope and b is the y-intercept of the line. Consider an example of 10 line segments with different starting times and destinations but with the same starting location and velocity. See Figure 4(a). We then transform the line segments into points in the SV model (Figure 4(b)) and in the Dual transform model(Figure 4(c)). In the figure, each circle represents a moving object and as in Figure 4(c), all line segments with the same velocity maps into a point in the Dual transform model. Now, consider a range query, which is supposed to return one object as shown in Figure 4(a). The SV model returns one exact answer whereas the Dual transform model returns all the objects as an answer. Although this may appear to be an extreme example, we expect this superior behavior of the SV model also in the general cases. To support this claim, we run several performance tests and report our findings in the next section.

4 Performance

In the previous subsection, we demonstrated how the SV model captures the problem well. In this section, we introduce the SS-Tree [20] as the underlying index structure, explain experimental settings, and show the result of a performance study.

4.1 SS-Tree

We use the SS-Tree [20] as the underlying index structure. The SS-Tree is a high-dimensional search tree proposed by White and Jain [20] for similarity searches such as k-nearest neighbor query. It has been shown to outperform the R^*-Tree [3] 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. The structure of the SS-Tree is basically the same as the R^*-Tree except for the fact that the SS-Tree uses a bounding sphere to enclose elements in a node rather than a bounding box. Using a bounding sphere reduces overlapping areas between index nodes, which is the main reason for the performance gain. Since moving objects can enter into and leave a system anytime, an index structure for the system should be able to dynamically insert and delete an element. The SS-Tree meets this criteria.

As discussed earlier, we can cluster moving objects with the same velocity, associate 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 3(D)) which is easier to handle since many index structures(e.g., R^*-Tree variants) use it as a result set of a range query.

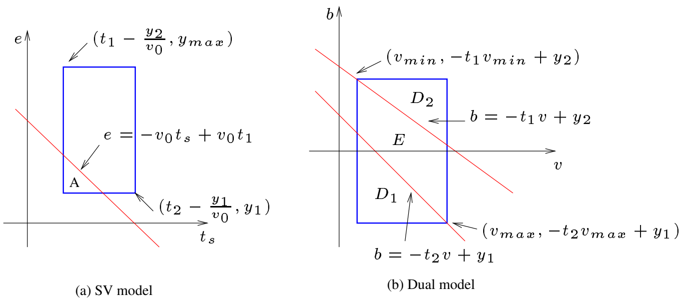


Figure 5: Two Models

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, $v = 0.8, 0.9, ..., 1.2$. The Dual transform model does not require multiple instances of an index structure, therefore, only one SS-Tree is used.

4.2 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 $N = 10000, 20000, ..., 80000$. 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($insertList$) sorted based on t_s and the other($deleteList$) sorted on the time when a moving object reaches the destination and is supposed to remove itself from the system. We calculate the time(t_e) as follows:

$$t_e = \frac{e-s}{v_0} + t_s \quad .$$

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, we let the system run for the first 20% of the time which is 100 ticks in this test. When it inserts an object into the SV model, it finds the SS-Tree corresponding to the same velocity as the object's and inserts it into the tree together with (id, t_s, e). And when it inserts it into the Dual transform model, it just does it together with the 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 against the two models and collect the results at the end of the test. To generate a random query, we generate a rectangle $[t_1, t_2] \times [y_1, y_2]$ with some width. If the width 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. We run the same query against the SV model and the Dual model. For a given query, the SV model first converts the rectangle to another rectangle with the following format(See Figure 5(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 rectangle to another one, but with a different format(See Figure 5(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 5(a). The rest of the objects are the exact answers. We refer to this the *SV overhead* .

Before we identify different overheads in the Dual transform model, we need to take a closer look at [13] in which Kollios et al. propose a query approximation algorithm using multiple B^+-Trees to index moving objects in the Dual transform model. They noticed that depending on where you consider the origin is, the b coordinate(the y-intercept of the corresponding line segment) can be different, resulting in the the areas of D_1 and D_2 in Figure 5(b). They keep multiple B^+-Trees which contain the same information about the moving objects but use different origin values. When a query arrives, they first compute the sizes of the area $D_1 + D_2$ from all indices, choose the one with the minimal size, and process the query against that index structure. In this fashion, they minimize the overhead associated with the areas D_1 and D_2. Note that test data set in [13] is uniformly distributed over the original space, therefore the minimum size of D_1 and D_2 implies minimal number of objects in that area. We call this the *index overhead* which is the number of objects that would not have been retrieved if we do not use the rectangle as a measurement of the answer.

Since we use one multidimensional index structure whereas Kollios et al. [13] use multiple instances of one dimensional B^+-Tree, the size of the areas D_1 and D_2 that we generate is an upper bound of what is incurred in [13] implementation. 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 6, 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). On the other hand, the *index overhead* in the Dual transform model is greater than 90% in 1% query and around 60% in 10% query. The *dual transformation overhead* in the Dual transform model is around 25% in 1% query and around 22% in 10% query.

Figure 7 shows that as the width gets larger, the *SV overhead* becomes bigger and the overheads due to the Dual transform model become smaller. The 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 in Figure 5. 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.

Figure 8 (a) shows the percentage of the objects retrieved. The number of objects retrieved in both models grows as the query width increases. Although in the SV model the percentage grows faster than that of the Dual transform model, the difference remains large due to the fact that the Dual transform model incurs large overheads. However, Figure 8 (b) reveals a somewhat different result: the difference between the two models in terms of the leaf node I/O is not that large, unlike

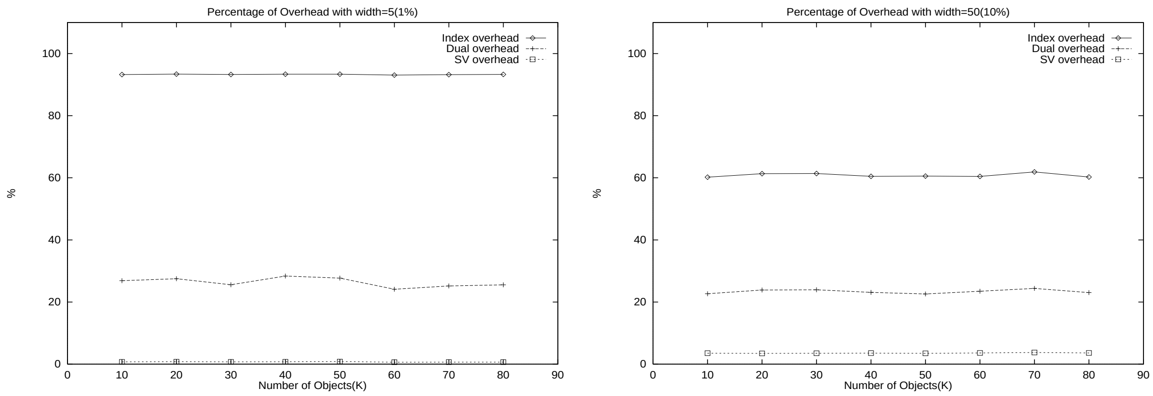


Figure 6: Percentage of overheads with varying data size

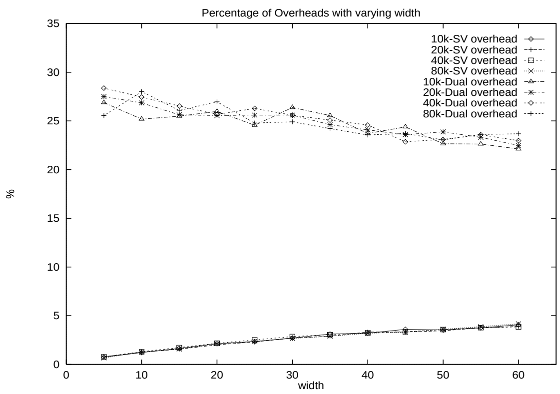


Figure 7: Percentage of overheads with varying width

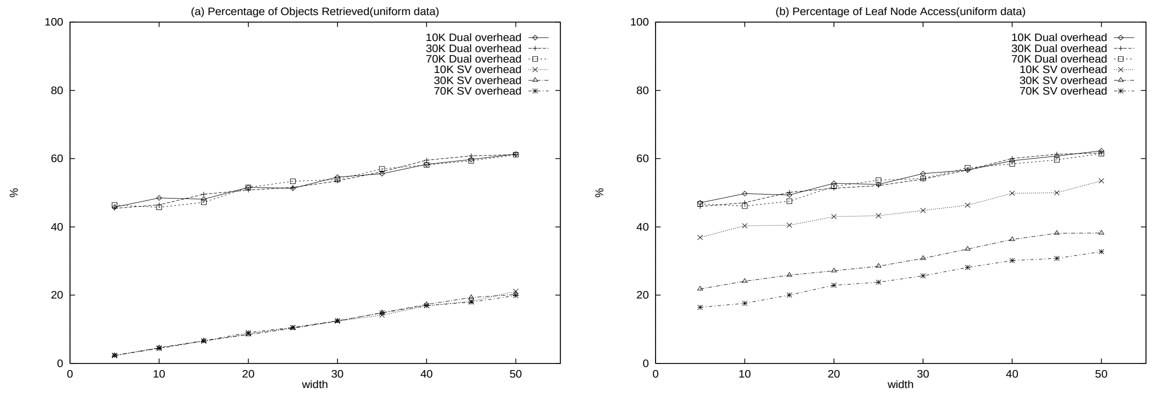


Figure 8: I/O Performance with uniform data distribution

any other cases we have seen so far. Further analysis indicates that since we have used 5 SS-Trees for the SV model, the number of leaf nodes in each tree are roughly one fifth of that for the Dual transform model, which implies that the density of the leaf nodes in the SV model is sparser than that of the Dual transform model so that it may visit more nodes when the trees are queried in the SV model. What we would like to emphasize here is that we are comparing two models so that the comparison of the percentage of the overheads are meaningful.

4.3 Simulated Realistic Data

We start by describing how we generate the realistic data. In 1990, the Santa Barbara County Association of Governments published the document [14] 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 this table, 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(Table 1). The average speeds are different from each section but they are roughly around 30 miles per hour. That is all the direct information we can derive from the document. However, 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 [14]. 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. We use 1 as the average speed. We have to multiply 120 to the distances between exits since 30 miles per hour is the average speed and we use seconds as the unit of time. After applying this scheme, the numbers we derive is not exactly same as [14]. However, it is similar in the sense that when the exact numbers change, they also change with similar rate. The total of the cars we generate is 24,182.

The test setup is identical to the previous subsection. We use 5 SS-Trees for the SV model and

Exit Name	Miles From	Annual Growth	per lane per hour	lane	2000 plph	per segment per hour	simulated
1 Hollister	0	0	0	0	0	0	0
2 Storke	2.1	2.1	1050	4	1319	5278	5278
3 Los Carneros	1.1	2.1	1500	4	1885	7541	7442
4 Fairview	1.2	2.1	1750	4	2199	8797	8633
5 Route 217	1.1	2.1	1300	6	1633	9803	9584
6 Turnpike	1.3	2.1	1700	6	2136	12819	12316
7 State	1.7	2.1	1800	6	2262	13573	12995
8 La Cumbre	0.7	2.1	1850	6	2325	13951	13323
9 Las Positas	1.2	2.2	1950	6	2477	14864	14192
10 Mission	0.7	2.1	1900	6	2388	14328	13983
11 Carrillo	1.0	2.8	1800	6	2438	14633	14337
12 Castillo	0.6	3.3	1700	6	2429	14578	14414
13 Milpas	1.4	4.1	1650	6	2567	15402	14995
14 Cabrillo	1.4	2.7	2300	4	3083	12332	12309
15 Olive Mill	0.9	2.6	2200	4	2917	11670	11520
16 San Ysidro	0.5	2.6	2200	4	2917	11670	11470
17 Sheffield	1.0	2.2	2000	4	2540	10163	10396
18 Evans	0.7	2.2	1850	4	2350	9401	9889
19 Padaro N.	1.2	2.2	1800	4	2286	9147	9752
20 Padaro S.	2.0	2.3	1750	4	2247	8989	9712
21 Santa Monica	1.4	2.5	1700	4	2230	8922	9773,
22 Linden	0.7	3.1	1650	4	2308	9233	9981
23 Casitas Pass	0.4	3.4	1650	4	2383	9533	10294
24 Bailard	1.0	3.5	1600	4	2335	9343	10155
25 Route 150	0.4	3.2	1000	6	1414	8484	9367
26 County Line	1.1	3.1	950	6	1329	7974	0

Table 1: Route 101

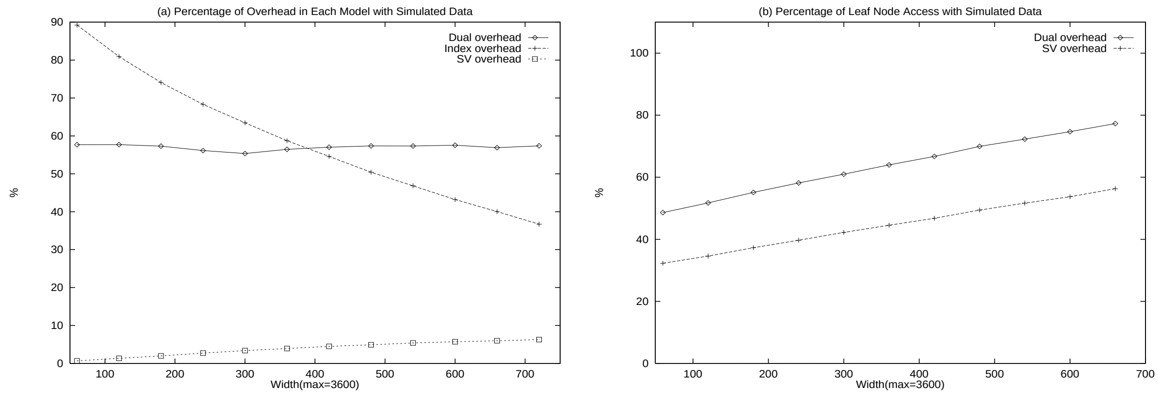


Figure 9: Performance with realistic data distribution

one for the Dual transform model and let the system run 20% of the time, 720 seconds in this case. Figure 9 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 an earlier findings.

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, an 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 [12, 13]. 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. Actually, searching for an efficient index structure for the SV model will be the one of our future work. We also plan to build a prototype for mobile databases based on the proposed approach.

References

[1] Pankaj Agarwal, Lars Arge, Jeff Erickson, Paolo Franciosa, and Jeffrey Vitter. Efficient Searching with Linear Constraints. In *Proceedings of ACM Symp. on Principles of Database Systems*, Seattle, Washington, 1998.

[2] I. Akyildiz, J. McNair, J. Ho, H. Uzunalioglu, and W. Wang. Mobility Management in Current and Future Communications Networks. *IEEE Network*, pages 39–49, July 1998.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard 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.

[4] Ken Buchanan, Rodger Fudge, David McFarlane, Tim Phillips, Akio Sasaki, and Howard Xia. IMT-2000: Service Provider's Perspective. *IEEE Personal Communications*, August 1997.

[5] CalTrans. Advanced Transportation Systems program Plan. *http://www.dot.ca.gov/hq/newtech*, 1996.

[6] Martin Erwig, Ralf Hartmut Güting, Markus Schneider, and Michalis Vazirgiannis. Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. In *GeoInformatica*, volume 3, 1999.

[7] G. Evangelidis, D. Lomet, and B. Salzberg. The hB^{π}-tree: A Modified hB-tree Supporting Concurrency, Recovery and Node Consolidation. In *Proceedings of ACM Symp. on Principles of Database Systems*, 1993.

[8] Bret Foreman. A Survey of Wireless Communications Technologies for Automated Vehicle Control. In *Future Transportation Technology Conf. and Exposition*, Costa Mesa, CA, August 1995.

[9] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A Data Model and Data Structures for Moving Objects Databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 319–330, Dallas, Texas, May 2000.

[10] Erik G. Hoel and Hanan Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *Advances in Spatial Database - 2nd Symposium*, 1991.

[11] P.V.C. Hough. Method and Means for Recognizing Complex Patterns. *U.S. Patent No. 3069654*, 1962.

[12] H. V. Jagadish. On Indexing Line Segments. In *Proceedings of the Int. Conf. on Very Large Data Bases*, Brisbane, Australia, 1990.

[13] George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On Indexing Moving Objects. In *Proceedings of ACM Symp. on Principles of Database Systems*, 1999.

[14] Gerald Lorden, James Kemp, David Daskal, Ann Terry, and Nannette Ramirez. *South Coast route 101 corridor study.* Santa Barbara County Association of Governments, June 1990.

[15] Randal C. Nelson and Hanan Samet. A Consistent Hierarchical Representation for Vector Data. In *ACM SIGGRAPH*, pages 197–206, August 1986.

[16] Dieter Pfoser and Christian S. Jensen. Capturing the Uncertainty of Moving-Object Representations. In *Proc. of the SSDBM Conf.*, pages 111–132, 1999.

[17] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 331–342, Dallas, Texas, May 2000.

[18] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and Querying Moving Objects. In *Proceedings of the Int. Conf. on Data Engineering*, April 1997.

[19] Jamel Tayeb, Ozgur Ulusoy, and Ouri Wolfson. A Quadtree Based Dynamic Attribute Indexing Method. In *Proceedings of ACM Symp. on Principles of Database Systems*, 1998.

[20] David A. White and Ramesh Jain. Similarity Indexing with the SS-Tree. In *Proceedings of the Int. Conf. on Data Engineering*, 1996.

[21] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving Objects Databases: Issues and Solutions. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, July 1998.