iRoad: A Framework For Scalable Predictive Query Processing On Road Networks *

Abdeltawab M. Hendawi Jie Bao Mohamed F. Mokbel

Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
{hendawi, baojie, mokbel}@cs.umn.edu

ABSTRACT

This demo presents the *iRoad* framework for evaluating predictive queries on moving objects for road networks. The main promise of the *iRoad* system is to support a variety of common predictive queries including *predictive point* query, *predictive range* query, *predictive KNN* query, and *predictive aggregate* query. The *iRoad* framework is equipped with a novel data structure, named *reachability tree*, employed to determine the reachable nodes for a moving object within a specified future time $\mathcal{T}$. In fact, the *reachability tree* prunes the space around each object in order to significantly reduce the computation time. So, *iRoad* is able to scale up to handle real road networks with millions of nodes, and it can process heavy workloads on large numbers of moving objects. During the demo, audience will be able to interact with *iRoad* through a well designed Graphical User Interface to issue different types of predictive queries on a real road network, to obtain the predictive heatmap of the area of interest, to follow the creation and the dynamic update of the reachability tree around a specific moving object, and finally to examine the system efficiency and scalability.

1. INTRODUCTION

The progression of GPS-enabled devices, e.g., in-car GPS, smart phones inspires a wide range of location-aware services, e.g., finding nearby facilities. An essential category of these services is based on objects future locations under the name of *predictive* queries [2, 3, 4, 5]. Predictive queries are concerned with the whereabouts of a set of moving objects in the near future. Primarily, numerous applications can benefit by considering the manipulation of predictive queries such as traffic management, aircraft management, routing, ride sharing, and advertising.

In this demo, we present the *iRoad* framework to support predictive query processing on moving objects for road networks. The basic query we address here is *predictive point* query, to find out the objects predicted to show up around a certain node within a given time units in the future. This fundamental query enables the prediction to be done on the lowest level, node, in the underlying road network. Standing on this building block query, *iRoad* also supports a wide variety of predictive queries within its framework including *predictive range* query, *predictive KNN* query, and *predictive aggregate* query. By the deployment of *iRoad* system, location-based services offered by many real applications can be improved, for example;

Traffic Management Systems. By employing *iRoad* framework, traffic management systems can benefit from running a bunch of predictive queries such as *predictive aggregate* query to find an estimate for the number of cars expected to be inside a certain region, e.g., down town, in the next time period, e.g., 20 minutes, from now. So they can take actions to handle possible congestion in advance. Users do not have to write *SQL* code to issue their queries, instead, through the usage of the nice graphical user interface of *iRoad*, users can simply draw a rectangle on a map to highlight the area of interest and enter a value for future prediction time period. Then the system returns the number of objects expected to be in the query rectangle within the specified time units. In addition, *iRoad* offers another smart tool named predictive heatmap. Through one quick look at the predictive heatmap, user can figure out which areas are expected to be over crowded.

Ride Sharing Systems. The main objective of ride sharing applications is to find the driver/rider closest to a rider/driver current location. By embedding *iRoad* framework, the ride sharing services can be enhanced by issuing *predictive range* query to find out the drivers expected to be nearby a rider's location in the near future. This helps users to better plan their trips and avoid wasting their times waiting for a driver to showup around their locations specially in uncomfortable conditions, e.g., bad weather, dangerous areas.

Location-based Advertising. By leveraging *iRoad* to run *predictive KNN* query, a store in a sale season can send electronic coupons to the K, e.g., seven, costumers that most likely to show up around its location within the next t time units, e.g., next 30 minutes. This paradigm allows location-based advertising to go beyond current closest customers to target possible closest ones in the near future. Sending coupons and promotions to those possible customers can encourage them to stop by the store which in turns increase the effectiveness of advertising for both business owner and consumer.

The main idea of *iRoad* system is to employ a novel data structure, named *reachability tree*, to hold the nodes reachable within a certain time frame $\mathcal{T}$ from an object current location. We assume that objects follow shortest paths during their travel from source

*This work is supported in part by the National Science Foundation under Grants IIS-0952977, IIS-1218168, and the Egyptian Ministry of Higher Education under Grant GM-887.

Proceedings of the VLDB Endowment, Vol. 6, No. 12

to destinations [6, 7]. So, we organize the nodes inside *reachability trees* according to the shortest path from the object start node which is the root of the tree. The time frame $\mathcal{T}$ is a controllable parameter used to determine the maximum prediction time *iRoad* can support. However, we use 30 minutes as a default value for $\mathcal{T}$, based on the finding that mean trip length for private cars is 19 minutes, NHTS [7]. By employing the reachability trees, *iRoad* is able to prune the space around each object which significantly shrinks the number of nodes to be considered for predicting object possible destinations. A probability value is assigned to each node according to its position in the tree, such that closer nodes have higher probabilities. The probability of a node n_i being a destination to the object *O* is equal to the probability of its parent node n_j divided by the number of children of n_j, where the probability of the object's current node is one. According to the underlying application requirements, the system can be adjusted to consider only objects with probabilities above a certain threshold $\mathcal{P}$ in the reported answers. Based on this technique, handling large number of objects can be done efficiently which guarantees the scalability of the *iRoad*. To control the depth of the reachable trees, *iRoad* offers another tunable parameter, ϵ, to compromise between the storage consumption and computation overhead. When ϵ is set to its minimum value, zero, a new reachability tree has to be obtained each time the object leaves its current node. This means, less storage needed to hold the trees but more computation for trees construction and pruning. On the other side, when ϵ is set to its maximum value, $\mathcal{T}$, one tree only will be used during the object whole trip, which means more storage to hold much bigger tree, but less computation overhead.

2. SYSTEM OVERVIEW

In this section, we define the basic query we support, and its assumptions and extensions, then we briefly describe the *iRoad* data structures and modules.

Query. Initially, we focus on addressing the *predictive point* query as our basic query on road networks. The *predictive point* query can be formalized as: "Given (1) a set of moving objects $\mathcal{O}$, (2) a road network graph *G* = (*V, E, W*), where *V* is the set of nodes, *E* is the set of edges, and *W* is the edges weights (i.e., travel times), (3) a maximum prediction time $\mathcal{T}$, and (4) a predictive point query *Q*(v, t), where $v \in V$, and t is a time period such that $t \leq \mathcal{T}$, we aim to find the set of objects $R \in \mathcal{O}$ expected to show up around the node v within the future time t. The returned result should identify the objects along with their probabilities to show up at the node of interest. For example, within the next 30 mins, object O_1 is expected to be at node v_3 with probability 0.8, so, the query result will be *R*(*Q*(v_3,30)) = {<O_1,0.8>}.

Assumptions. We assume that moving objects follow shortest paths in their routing trips. The intuitions behind this assumption is based on the fact that most of the moving objects, e.g., drivers, travel through shortest paths to the destinations [6, 7]. The $\mathcal{T}$ value is bounded by the finding that the average trip length for private cars is approximately 19 minutes, national household travel survey [7].

Extensions. We consider the *predictive point* query as a building block upon which *iRoad* can support other types of predictive queries including: (i) *Predictive range* query, where a user defines a query region that might contain more than one node and asks for the list of objects expected to be inside the boundaries of that region within a specified future time, (ii) *Predictive KNN* query to find out the K objects expected with the highest probability to be around the node of interest within a certain time period, and (iii) *Predictive aggregate* query to return the number of predicted objects to be inside a given region in the next time period.

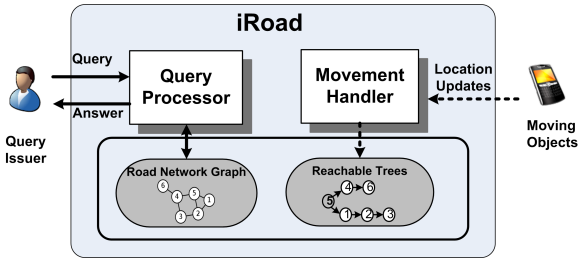

Figure 1: iRoad System Architecture

Data Structures. In *iRoad*, there are three basic data structures to maintain: (1) *Road Network Graph* contains a set of nodes *V* and edges *E*, the edges weights in terms of travel times. For each node v in the road network, we store the *predicted answers* to carry a list of objects predicted to show up around the node *v* with their probabilities in the future time *t*, determined in the query, e.g., 20-minute. (2) *Reachability Tree*, the core data structure inside *iRoad*. For each node v in the underlying road network graph *G*, we build a reachability tree with v as a root, to hold all nodes reachable to v within a prespecified time frame $\mathcal{T}$. Detailed discussion in Section 3. (3) *Trip History*, for each moving object, we maintain a buffer to hold the recent history of an object during its present trip.

System Architecture. Figure 1 gives the architecture of the *iRoad* framework, which consists of two main modules, namely, the *movement handler* module and the *query processor* module. A brief illustration will be provided in Section 4 and 5 respectively.

3. REACHABILITY TREES

Our proposed system *iRoad* intelligently employs a novel data structure named *reachability tree* to prune the road network around each moving object. Yet, only nodes reachable to an object start location within a specified time limit $\mathcal{T}$ are obtained and organized in a tree structure rooted by the object start node. The tree organization is based on the shortest path from the root node to the rest of nodes in the *reachability tree*, meaning that, traversing a tree from the root node to a leaf node gives the shortest path from the root to this leaf.

By pruning the space around each moving object, we significantly reduce the computation overhead required to compute and update the predicted objects along with their probabilities at each node in the underlying road network. The usage of *reachability tree* facilitates the computation of the probability that the object will be at each reachable node within a certain time units. Simply, the probability of an object O_i to be at a certain node v after t time units is equal to one divided by the number of nodes in the sub-tree underneath the object current node u. Surely, the employment of *reachability trees* inside the *iRoad* framework improves the query processing efficiency and guarantees the system scalability. The reason for that is because the possible destinations of the prediction become limited. Thus, we only need to consider a limited number of nodes for predication computation, instead of millions of nodes in a real road network. Therefore, *iRoad* can efficiently scale up to support large number of objects over real sized road networks.

The main idea to construct a *reachability tree* is to use a best-first expansion algorithm, similar to incremental network expansion algorithm INE [8], to visit the nodes and edges on the road network that are reachable through shortest path traversing. During the construction, we consider two parameters, the prediction time $\mathcal{T}$ which determines the maximum prediction time *iRoad* can support, and the time buffer ϵ which takes values from zero to $\mathcal{T}$ to decide the

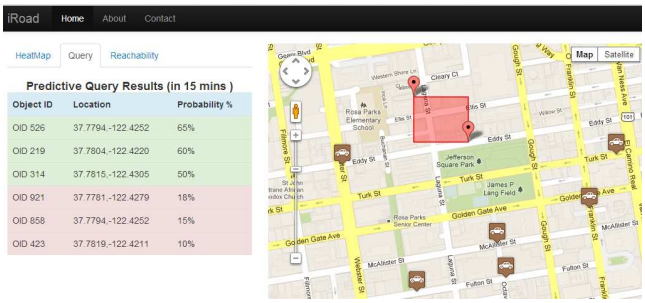

Figure 2: iRoad GUI for Predictive Range Query

Figure 3: iRoad GUI for Predictive HeatMap

depth of the *reachability tree*. The basic *reachability tree* is able to hold nodes reachable to an object in a limited time boundary, if the object travels over that time boundary, we need to obtain a new reachability tree. We can load a new reachability tree if is precomputed and saved on disk in advance or instantly compute it during the run time. In all cases, it is time consuming to obtain a new reachability tree with each single movement of an object. On the other side, we can obtain one huge reachability tree that allows the handling of the whole object trip without the need to load other trees. As a side consequence, the system will overwhelm the available storage. In between the two extremes, we provide a controllable parameter ϵ to allow the system to decide a buffer time for each reachability tree as a tradeoff between the query processing cost and the storage overhead. When ϵ is equal to zero, this means consuming less storage but with more computation overhead, while the vise versa will occur with ϵ equal to $\mathcal{T}$.

4. MOVEMENT HANDLER

This module is triggered when there is an object starting a new trip, a new movement for existing object, or an object ends its current trip. The idea of the *movement handler* module is to limit the updates caused by an object movement by limiting the space around each object. This is done by setting the maximum prediction time we can support to a specific time limit $\mathcal{T}$, e.g., 30 minutes, in the future. This space framing significantly reduces the cost consumed to update all nodes in the underlying road network graph by limiting update to those nodes inside the in-hand *reachability tree*. Initially, when an object starts a new trip, the *movement handler* constructs a *reachability tree* to hold the object limited space. Then, the list of *predicted answers* associated with each node in this tree is updated by inserting a new record carrying object identifier and object probability. Once, the object leaves its current node to a new one, we cascade deleting the object record from the *predicted answers* in all nodes no longer on the shortest paths from the object new node. At this moment, the sub-tree under the object new node is expected to be much smaller, yet, the probability to visit any of the nodes in this sub-tree increases. Therefore, we traverse the sub-tree rooted by the object new node to reflect the new probability on the *predicted answers*.

It is worthy to mention here that *iRoad* offers another adjustable probability threshold $\mathcal{P}$. By tuning $\mathcal{P}$, the *movement handler* module can be controlled to consider only the movement that causes the probability to be above a certain value $\mathcal{P}$, otherwise, it is ignored. For example, when an object O_1 starts its trip, it is expected to have many reachable nodes, e.g., 200 nodes, from the start location. The probability of O_1 to be at each of these nodes will be very small, e.g., 0.05, and not significant in some applications, e.g., traffic management. So, the system can be controlled to ignore updating the predicted results with this probability until it becomes larger than a specified $\mathcal{P}$ value, e.g., 0.10, which intuitively saves proportion of the computation overhead.

5. QUERY PROCESSOR

The main idea of processing predictive queries in *iRoad* is to have the predicted objects at each node precomputed in advance by the *movement handler* module, so for coming queries, the *query processor* module fetches those results, adapts them according to the type of received query and returns the answer in a very fast response time.

To evaluate a *predictive point* query, the *query processor* module initially finds the node of interest for which the query is asking about its predictable objects. Then, it retrieves the precomputed *predicted answer* saved with this node. For a *predictive range* query, where the user asks for the prediction inside a region that might contain many nodes rather than single node, the *query processor* combines the answers at those nodes of interest into a single basket by taking the union of the *predicted answer* lists associated with them. This will get ride of redundant objects. To unify the probabilities for object that appears in the result of more than one node of interest, we use the maximum probability among its occurrences. Finally, according the query type, (predictive range, KNN, aggregate), we adjust the in hand results. For example if the combined predicted results is $\{<O_1,0.75>, <O_2,0.25>, <O_3,0.35>, <O_4,0.65>\}$, the final answer for predictive aggregate query will indicate that two objects are expected to show up at the nodes of interest, while for predictive KNN query with K = 3, the answer will contain the three objects with highest probabilities, $\{O_1, O_4, O_3\}$, and for predictive range query, the four objects will be listed in the returned final query result.

6. DEMO SCENARIOS

This section presents the demo scenarios of the *iRoad* framework to illustrate its main functionalities through nicely designed set of graphical user interfaces. During this demo, we will be wearing two hats, the first is the users' one to describe how they can issue predictive queries and the formats of the results they receive, while the second is the system one to give an insight on its internal operations and depict the hidden processing required to answer users' queries. The demo is based on a large set of synthetic data of moving objects generated using the Brinkhoff's generator [1] on a real road network map extracted from the shape files of different counties in USA. The *iRoad* server is written in Java while the interfaces are web-enabled implemented using a combination of java scripts, HTML, and CSS. Four different scenarios will be provided during the demonstration venue, described as follows.

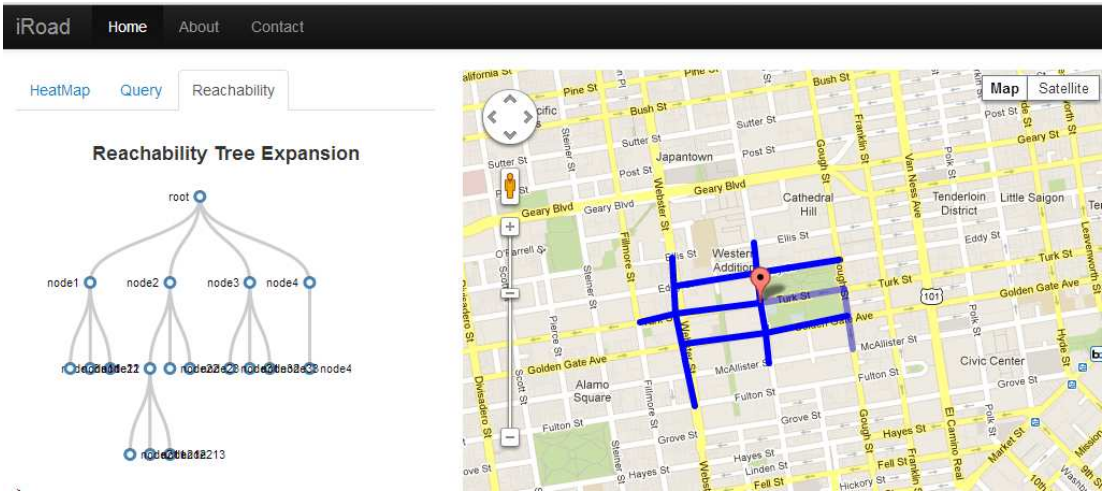

Figure 4: Reachability Tree Dynamics in iRoad Framework

6.1 Scenario 1: Issuing Predictive Queries

Initially, the audience can interact with the *iRoad* system by issuing different types of predictive queries including predictive point, range, KNN, and aggregate queries. Figure 2 gives a screen shot for a predictive range query where a user specifies a rectangular region that contains a number nodes on the map of San Francisco USA, and 15 minutes as the prediction time frame. Then the system responds by the list of objects predicted to show up inside the boundaries of the query region within the 15 minutes. The returned answer has three columns, objectId as identifier, latitude and longitude of the current location of the predicted objects, and the probability of how likely this object will be there within the determined time period. The resulted objects are plotted as car icons on the map. For predictive KNN query, the user can select a value for K to list only the K objects with the highest probability to appear in the query region.

6.2 Scenario 2: Predictive HeatMap

Figure 3 provides a screen shot for predictive heatmap which can be seen as a set of predictive point queries that cover every single node in a wide area of the map. Using the predictive heat map, audience can monitor the predicted objects in the area of interest and watch the updates on the predicted result using a nice color scheme. Heat map colors the zone around each node in the underlying map according to the number of predicted objects to be within it. In this example, red color refers to areas more likely to be crowded than the blued areas. The audience will be able to deal with the predictive heat map interactively by changing the query area, the prediction time, color schemas, and the opacity.

6.3 Scenario 3: Tree Dynamics

In this scenario we provide an eye on the *iRoad* internal gears in general, with more focus on the reachability tree in specific, as it is the core data structure. The map on the right hand side in Figure 4 illustrates the nodes reachable within 10 minutes, thick blued lines, from an object current location, water icon on the map, assuming the object follows the shortest path in its movements. The panel on the left hand side of Figure 4 provides the equivalent reachability tree where the root is the object present node. The audience will be able to notice the dynamic changes happen to the reachability tree as a reflection for the object movements. These dynamics include shrinking the in-hand tree when the object moves further to a different node, uploading new reachability tree when the trip goes over the preset maximum prediction time $\mathcal{T}$, or pruning the yet loaded tree by cutting out some branches based on the recent trip history. The audience also can control the system behavior by changing the ϵ value to decide the depth of reachability trees.

6.4 Scenario 4: Stress Test

This scenario is a stress test for the *iRoad* framework. Its objective is to give a glance on the system efficiency and scalability by running the system on heavy workloads. This is done by executing batches of queries, rather than a single query, and by using large road networks that contain millions of nodes, instead of small county or city, and large numbers of moving objects prepared using the Brinkhoff generator. The audience will be able to examine the size of the used data sets and choose between different workloads of road networks, queries, and moving objects. Then they can set the system parameters such as maximum prediction time $\mathcal{T}$, the reachability tree buffer ϵ, and the probability threshold $\mathcal{P}$. In addition, audience will be able to assess the overall system performance through inspection of the generated charts and graphs that show the CPU and memory costs for executing each workload.

7. REFERENCES

[1] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[2] A. M. Hendawi and M. F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *ACM SIGSPATIAL GIS*, California, USA, Nov. 2012.

[3] A. M. Hendawi and M. F. Mokbel. Predictive Spatio-Temporal Queries: A Comprehensive Survey and Future Directions. In *MobiGIS*, California, USA, Nov. 2012.

[4] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *ICDE*, pages 70–79, Cancn, Mxico, Apr. 2008.

[5] H. Jeung, M. L. Yiu, X. Zhou, and C. S. Jensen. Path Prediction and Predictive Range Querying in Road Network Databases. *VLDB Journal*, 19(4):585–602, Aug. 2010.

[6] J. Krumm. Real Time Destination Prediction Based on Efficient Routes. In *SAE*, Michigan, USA, Apr. 2006.

[7] J. Krumm, R. Gruen, and D. Delling. From Destiantion Prediction To Route Prediction. *Technical Report. Microsoft Research*, 2011.

[8] D. Papadias, J. Zhang, and N. Mamoulis. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, Berlin, Germany, 2003.