Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 09-024

Multi-Type Nearest Neighbor Queries Road Networks With Time Window Constraints

Xiaobin Ma, Shashi Shekhar, and Hui Xiong

September 14, 2009

Multi-Type Nearest Neighbor Queries Road Networks With Time Window Constraints

Xiaobin Ma
Teradata Corporation
xiaobin@cs.umn.edu

Shashi Shekhar
University of Minnesota
shekhar@cs.umn.edu

Hui Xiong
Rutgers University
hui@rbs.rutgers.edu

ABSTRACT

A multi-type nearest neighbor(MTNN) query finds the shortest tour for a given query point and different types of spatial features such that only one instance of each feature is visited during the tour. In a real life MTNN query a user normally needs an answer with specific start time and turn-by-turn route for specific period of time on road networks, which requires considerations of spatial and temporal features of the road network when designing algorithms. In this paper, we propose a label correcting algorithm that is based on a time aggregated multi-type graph, a special case of a time aggregated encoded path view. This algorithm gives the best start time, a turn-by-turn route and shortest path in terms of least travel time for a given query. Experimental results are provided to show the strength of our proposed algorithm and design decisions related to performance tuning.

1. INTRODUCTION

Widespread use of spatial search engines such as Google Maps and MapQuest is leading to an increasing interest in developing intelligent spatial-temporal query techniques. For example, a traveler may be interested in finding a shortest route in terms of least travel time with the best start time between 9:00 am and 11:00 am from his house through one grocery store (with a stay of 1 1/2 hours), one Best Buy store (1 hour stay) and one post office (arriving before 4:00 pm; 1/2 hour stay) and returning home before 8:00 pm. This query illustrates some important properties. First, the traveler is trying to find a route with instances from different feature types that are a grocery store, an electric appliances store and a post office. This kind of query is called a multi-type nearest neighbor (MTNN) query in [10]. Second, the route to be found is a closed route from the query point back to the query point. Third, the traveler is interested in not only the route but also the best start time. Considering the variability of traffic patterns on different times on road networks, this best start time could differ for different time windows. Therefore, the query asks for answers containing not only spatial features like the route but also temporal features. Forth, the query itself contains spatial and temporal features. For example, the query point and different interested locations are spatial features. The best start time between 9:00 am and 11:00 am and length of stay at each location are temporal features.

In this paper, we study the spatial-temporal MTNN query problem on spatial-temporal road networks.

Related Work Vehicle routing and scheduling problems have been extensively studied in Operational Research. According to a taxonomy given by Bodin *et al.* [1], vehicle routing involves the traversing of a sequence of points in order. Vehicle scheduling involves traversing a sequence of points with an associated set of departure and arrival times. If a vehicle must traverse a sequence of points with time window and/or precedence relationships, the problem is a combined vehicle routing and scheduling problem. Numerous computational methods to solve such problems have been developed. Laporte *et al.* presented a summary of exact and approximate algorithms for vehicle routing and scheduling problems [7]. He also summarized classical and modern heuristics for solving such problems [8]. However, in all of these works, the sequence of points to be visited in a query was specified in advance. Thus no solution required that the point space be searched for points that would be visited in the query, a key difference between previous works and our MTNN query problem.

In order to quickly find answers to spatial queries, researchers normally model road networks as a graph. Huang *et al.*[5] precomputed all-pair shortest paths and stored them in a spatial database. This precomputed graph is called an Encoded Path View (EPV). Later, Huang *et al.*[6] extended EPV to large road networks and proposed a Hierarchical Encoded Path View (HEPV). Referencing a HEPV makes it possible to answer a nearest neighbor query on road network very efficiently.

Recently, George *et al.* [4] proposed a Time-Aggregated Graph (TAG) to model a spatial-temporal network. Based on this model, spatial queries that have been studied for decades are answered along both spatial and temporal dimensions. For example, George showed that the SP-TAG algorithm computes the shortest path for a given start time in a small time-dependent network. In a related study, Ding [3] proposed a time-dependent graph and studied how to find the best departure time in terms of least travel time from one place to another over a large road network.

Meanwhile, the queries related to multiple feature types

.

attracted attentions from different database research groups. X. Ma *et al.* [10] formalized a MTNN query problem and proposed a Page Level Upper Bound (PLUB) based algorithm to find an optimal route for the MTNN query. Sharifzadeh *et al.* [11] recently proposed an Optimal Sequenced Route (OSR) query problem and provided three optimal solutions: Dijkstra-based, LORD and R-LORD. Essentially, the OSR problem is a special case of the MTNN problem. Since it fixes the visiting order of feature types, it can be thought of as imposing a spatial constraint on the MTNN problem. Sharifzadeh *et al.* [12] extended the OSR work to road network by using Voronoi diagrams. Basically the algorithm of this extension precomputes Voronoi diagrams for every possible partial route and finds the optimal sequenced route very efficiently. However, it does not consider the variation in traffic patterns that occurs at different times and thus ignores the temporal dimension of road networks. It also does not give a turn-by-turn route for the query on road networks. Another issue is that with Euclidean distance (i.e. L_2 norm) as metric the cell edges in Voronoi diagram may become hyperbolic curves, which makes the determination of whether or not a point is inside a cell very difficult.

Another recently published work [9] proposed a number of fast approximate algorithms to give sub-optimal solutions in metric space for Trip Planning Queries (TPQ). This work focused on efficient algorithms but could not guarantee finding the shortest path.

In this paper, we extend MTNN query in the temporal dimension. We formalize a common query in real life as a spatial-temporal MTNN query with time window constraints and answer the query based on our extension of the encoded path view, which we call the Time Aggregated Multi-Type Graph (TAMTG), a special case of the time aggregated encoded path view (TAEPV) of road networks.

Our Contributions. We formalize a BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) query problem on spatial-temporal road networks by extending the encoded path view from spatial-only to spatial-temporal road networks. By identifying the special properties of BESTMTNN query that lead to our spatial-temporal partial route growth approach we propose a label-correcting based algorithm to solve the BESTMTNN query problem. This algorithm prioritizes the spatial-temporal partial routes with current least travel time. It takes a user-specified query that involves in spatial-temporal features such as query time window sizes for all features and planned stay time interval at a location and gives a turn-by-turn route and the best start time in terms of least travel time. Our experiments show our algorithm can answer normal user BESTMTNN queries in a reasonable time.

Overview. The remainder of this paper is organized as follows. Section 2 formalizes the BESTMTNN problem. In section 3 we identify the special properties related to a BESTMTNN query and describes the Time-Aggregated Multi-Type Graph (TAMTG); we then present our partial route growth approach designed to accommodate the BESTMTNN query as well as a TAMTG-based label-correcting algorithm that finds the optimal solution for the BESTMTNN problem. Section 4 gives the experimental setup and experimental results. Finally, in Section 5, we conclude our discussion and suggest further work.

2. BASIC CONCEPTS AND PROBLEM FORMULATION

In this section, we introduce some basic concepts, explain some symbols used in the remainder of the paper and give a formal statement of the BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) query problem.

Let $< P_{1,1'}, P_{2,2'}, ..., P_{k,k'} >$ be an ordered point sequence and let $P_{1,1'}, P_{2,2'}, ..., P_{k,k'}$ be from k different (feature) types of data sets. P_{i,i',t_i} is a point P'_i of feature type i at time t_i. A spatial-temporal *Partial Route* $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{l,l',t_l})$ is a route from the query point q at time t_0 through points from different feature types but not back to the original query point q. A complete closed route $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, \ldots, P_{k,k',t_k}, q_{t_{k+1}})$ is a route that goes from the query point q at time t_0 through point $P_{1,1'}$, goes from point $P_{1,1'}$ at time t_1 through $P_{2,2'}, \ldots$, and returns to query point q at time t_{k+1} from $P_{k,k'}$ at time t_k. $t(R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t_{k+1}}))$ represents the travel time through the route $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t_{k+1}})$.

A BESTMTNN is defined to be an ordered point sequence $< q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0} >$ such that $t(R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0}))$ is minimum among all possible routes for all possible start time points and all qualified time window. A BESTMTNN query is a query finding a BESTMTNN that includes a best start time, a turn-by-turn route, and a least travel time in given spatial and temporal data sets. Thus, $< q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0} >$ is a BESTMTNN query result.

An Encoded Path View (EPV) stores all pairs of shortest distance paths in a spatial database. A Time Aggregate EPV (TAEPV) records such spatial information but extends the EPV to include temporal information. In other words, a TAEPV stores all pairs of shortest distance for all time points in terms of least travel time. With a TAEPV on road networks, the results of a BESTMTNN query consolidate the location information of interested points with different traffic patterns that vary by times. A Time Aggregate Multi-Type Graph (TAMTG) is a special case of a TAEPV. In a TAMTG, only least travel times among all points of interest (POI) are stored. In a BESTMTNN query, the POIs are the given k data sets from k different feature types.

The following is a formal definition of the BEst Start Time MTNN (BESTMTNN) query problem. In the BESTMTNN query problem, we use road distance since we are searching for BESTMTNN on spatial-temporal road networks. The spatial-temporal road networks are represented by a TAEPV and a TAMTG that store least travel times among points. These least travel times are calculated in advance because routing and scheduling are extremely time-consuming. Considering POIs could number from hundreds to thousands and even more, it is infeasible to do an ad-hoc routing and scheduling for every BESTMTNN query. In this problem, we also consider time window constraints as user-specified parameters indicating what time intervals qualify for this query and how long the user is going to stay at a location (planned stay period). Time window constraints are meaningful parameters in daily life. For example, a post office can be visited only certain hours in a day, and a movie-goer will want to arrive at a cinema before a movie's specific start time and stay at the cinema as long as the movie is scheduled to last. Given all these input parameters, our objective is to minimum the travel time on the road networks. In this

problem, the planned stay period at any location is not considered as part of total travel time. What interested us is the travel time on the road.

Problem: The BESTMTNN Query
Given:

- A query point
- Distance metric - road distance
- k different types of points of interest
- Spatial-temporal road networks represented by Time Aggregated Encoded Path View (TAEPV) and Time Aggregated Multi-Type Graph (TAMTG)
- Time window constraints

Find:

- Least travel time, turn-by-turn route and BEst Start Time
  Multi-type Nearest Neighbor (BESTMTNN)

Objective:

- Minimize the travel time on the road networks from the query point covering an instance of each feature type and then back to the query point

3. BESTMTNN ALGORITHM

Here we examine in more detail that a typical BESTMTNN query from daily life that was introduced earlier. "Find a route with the best start time between 9:00 am and 11:00 am from my house through one Cub Food store (stay for about 1 1/2 hours), one Best Buy store (stay for about 1 hour) and one post office (arrive before 4:00 pm, stay about half an hour) and return to my house before 8:00 pm." From this typical query, we can find some important properties that could be used to guide the algorithm design for a BESTMTNN query. First, normally a traveler queries the best start time within a defined time window (e.g., "between 9:00 am and 11:00 am"). By contrast, he's willing to return home at any time as long as it's before his latest time (e.g., "Return to my house before 8:00 pm"). We can see therefore that the window containing possible start times is much smaller than the "start - return" time window. This suggests that a forward search, beginning from the starting query point, may be preferred for most cases. Second, the query starts from a query point (e.g., "my house") and ends at the same query point (e.g., "return to my house"). This basically says that the traveler asks for a closed travel route, which is different from the queries studied in previous works [9, 10, 11, 12]. In those studies, the requested travel routes do not include routes returning to the query point. This means that some properties identified for designing previous algorithms may not always hold. More specifically, the property 2, which guarantees the correctness of the LORD and RLORD algorithms in [11], is not always correct, and thus new properties should be identified in the design of algorithms for closed route queries. Third, the traveler asks for a specific turn-by-turn route. Here we can differentiate the two levels of routes. The first level is the route through only the points of interest (POIs) without routing and scheduling on points that are not in the set of POIs. The second level of route is the route between two POIs. There may be multiple routes between these two points on road networks so there is a routing and scheduling issue. The BESTMTNN algorithm and corresponding data structure used in the algorithm should support both levels of routing and scheduling. Finally, a BESTMTNN query is a temporal query; embedded in the query's search for the best start time and route is the assumption that traffic patterns change over time. There are three kinds of temporal patterns: long-term trends used in long-term forecasts, short-term information available when starting travel and used in short-term forecasts, and dynamic perturbation available only when arriving at a destination, and representing any unforeseen events encountered during travel. It is known that traffic volumes exhibit typical long-term temporal patterns. That is, traffic volumes vary at different time points known in advance. This kind of information can be used to do long-term forecasts. Due to the complexity of BESTMTNN queries, we do not consider short-term forecasts or dynamic perturbation in this paper.

As stated above, the BESTMTNN query differs from previously studied queries and thus new properties need to be identified and existing data structures extended to support solutions to this query.

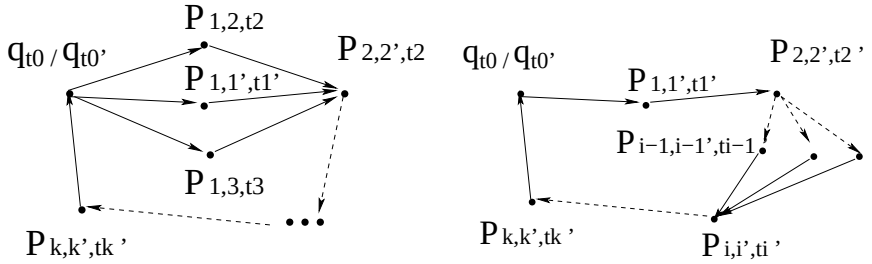
3.1 BESTMTNN-Related Properties

The BESTMTNN algorithm enumerates all permutations of all feature types. For each permutation, it starts with partial route only containing the query point and grows a partial route by adding points from next feature to a current partial route. In this part, we discuss some BESTMTNN-related properties that guarantee the corretness of partial route growth procedure.

Property 1 If a route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{k,k',tk}, q_{t0'})$ is the optimal route, then the travel time of route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}) = t(R(q0, P_{1,1',t1})) + t(R(P_{1,1',t1}, P_{2,2',t2}))$, is shortest among all possible routes from the query point q through any point from feature type 1 at any time and then reaches point $P_{2'}$ from feature type 2 at time $t2$.

In this property statement $P_{i,i',ti}$ represents point $P_{i'}$ of feature type i starting at time ti if the point $P_{i'}$ is not an end point on a route. If the point $P_{i'}$ is an end point on the route, $P_{i,i',ti}$ represents point $P_{i'}$ of feature type i with arrival time ti. Figure 1 (a) illustrates property 1. In this figure, the route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2})$ from the query point q_{t0}, through point $P_{1,1',t1}$ from feature type 1 and reaching point $P_{2,2',t2}$ from feature type 2 has the least travel time. However, when growing a partial route from the query point q to feature type 1, we don't know which specific partial route $R(q_{t0}, P_{1,1i,t1i})$ will lead to a shortest path from q_{t0} and reaching $P_{2,2',t2}$. Therefore, in order to grow partial routes from the query point q through a feature type 1 point to a feature type 2 point, we need to store the least travel time from the query point q through all feature type 1 points for all qualified time points.

This property is important and necessary because the BESTMTNN query route is a closed route that starts and ends with the same query point. Based on this property, we know that beginning a BESTMTNN query requires storing all the partial routes from the query point to all the points in the first feature type for all time points. This property also makes it possible to use a forward search in order to find a closed route.



(a) Property 1 (b) Property 2

Figure 1: Properties Related to BESTMTNN Query

Property 2 If a partial route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{i-1,(i-1)',t(i-1)}, P_{i,i',ti})$ is part of an optimal route, then the travel time of this partial route is least among all partial routes starting with q and ending with $P_{i,i',ti}$ for specific arrival time point t_i.

In partial route calculations, it is possible to have multiple partial routes ending with the same point at the same time point. Property 2 guarantees that only the partial route with the least travel time for a specific end point and time point needs to be stored. It also indicates that a stored partial route can be identified by its end point and arrival time point. According to property 2, it is enough to store information for the specific partial route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{i-1,(i-1)',t(i-1)}, P_{i,i',ti})$. Figure 1 (b) illustrates Property 2. In the figure, the partial routes ending with points from feature type $i-1$ are either from a different location or from a different time point. Before growing the partial route to feature type i, it is unknown which partial route ending with a point from feature type $i-1$ will be on the optimal route. However, according to property 2, it is known that the partial route must have a least cost (travel time) if it is on the optimal route. Therefore, in order to grow partial routes from feature type $i-1$ to feature type i it is enough to store those least cost partial routes ending with different points from feature type $i-1$ on different time points. After growing from all these partial routes to a specific point $P_{i,i',ti'}$ of feature type i at specific time ti' there are still multiple partial routes. All these newly grown partial routes could be represented as $R(q_{t0}, \ldots, P_{i-1,(i-1)',t(i-1)'}, P_{i,i',ti'})$, $R(q_{t0}, \ldots, P_{i-1,(i-1)'',t(i-1)''}, P_{i,i',ti'})$ etc. At this time, we know if the partial route ending with $P_{i,i',ti'}$ is on an optimal route it must be shortest. So, we only need to store one partial route ending with $P_{i,i',ti'}$ that has least cost.

3.2 Time Aggregate Multi-Type Graph (TAMTG)

A TAEPV stores all pairs of least travel times among all points, the start time of the route with the least travel time, and the next hop on the route at all time points on a graph. However, not all of this information is needed for partial route growth in a BESTMTNN query. Of interest in partial route growth are the least travel time among the points of interest (POIs) from different feature types at all time points and the start time of the route with the least travel time. In other words, there is no need to store the least travel time involved in a point of non-interest or if the least travel time is among the POIs from the same feature type. We call the graph that captures this more relevant set of least travel times a Time Aggregate Multi-Type Graph (TAMTG).

As an illustration, the "initial" part of the Figure 2 shows part of TAMTG. Indicated on the graph is the least travel time from the query point q to points $r1$ and $r2$ from feature type r for time points 2, 3 and 4 and the least travel time from points $r1$ and $r2$ to point $b1$ from feature type b for time points 4 to 14. It is worth noting that there may not be a direct link between points. Instead in the TAMTG graph, the least travel time between two points has been calculated in advance. In addition, for simplicity the graph does not show all the calculated least travel times among points. For example, the least travel time from $b1$ to $r1$ and $r2$ is not given.

3.3 Partial Route Growth

A point on a partial route contains not only the point itself but also a specific time point. For example, $P_{i,j,tj}$ represents the point P_j on the partial route from feature type i at time point tj. Both spatial and temporal data are required because traffic volume varies over time, that is, the time dimension plays an important role in the modeling of spatial-temporal road networks. The current partial route may come from different previous partial routes for different time points. In other words, to answer a BESTMTNN query it is not enough to store location (point) information. What is needed is point information for all qualified time points. Due to the time window constraints, not all time points may qualify when searching for a best route. As discussed in the BESTMTNN properties section, a partial route can be identified exclusively by its end point and the arrival time at the end point because only partial routes with specific end points and specific time points are needed to grow the partial route to the next feature type. In summary, identifying a partial route requires storing a location and a time point along with a least travel time.

In daily life, when traffic volume is very high, drivers often choose to stay at their location instead of trying to drive at that moment. Common sense tells them that driving during high volume traffic will not expedite their travel or will do so just a little. If staying extra time at a location leads to shorter or equal travel time later, it is reasonable to choose staying extra time units at a location. In partial route calculation of the BESTMTNN query algorithm, it is possible to add extra time units to a location, but this extra time needs to be counted as part of travel time since it is not a planned stay. Therefore, there are two categories of cost at a partial route. One is directly generated from the growth of previous partial routes to the current partial route. In Figure 2, for

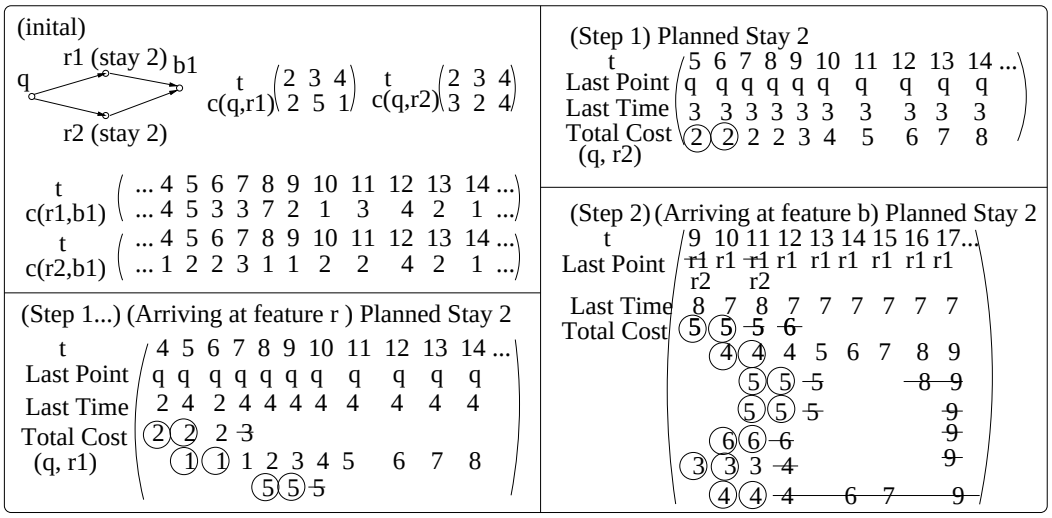


Figure 2: Partial Route Growth

example, the cost 4 of partial route $R(q_4, r_{1,7}, b_{1,10})$ is calculated from the growth of partial route $R(q_4, r_{1,7})$. Since there is a planned stay of 2 units at feature type b, the partial routes $R(q_4, r_{1,7}, b_{1,10})$ and $R(q_4, r_{1,7}, b_{1,11})$ cannot be grown at times 10 and 11. The number 4 inside a circle in "Step 1" and "'Step 2" indicates that this cost cannot be used for next partial route growth. For the same 2-unit planned stay, the cost of partial route $R(q_4, r_{1,7}, b_{1,12})$ is kept as 4. The other cost arises from any extra stay on the partial route that ended with the same point. For example, the cost 5 of partial route $R(q_4, r_{1,7}, b_{1,13})$ results from 1 extra time unit stay at point $b1$ at time 12.

When updating an existing partial route or generating a new partial route, it is necessary to update or generate the route for all qualified time points from all previous partial routes. The implication is again that a partial route in a BESTMTNN query is identified by both the end point on the route and a qualified time point. Here a qualified time point means a time point within the query time window for a currently visited feature.

Figure 2 illustrates the partial route calculation procedure. In the figure, t represents a time point. *Last Point* stores the last point on the partial route before reaching the current feature type. *Last Time* stores the start travel time from *Last Point* to the current point. *Total Cost* stores the least travel time for this partial route so far. The *Total Cost* number inside the circle means the time point with this cost is still within the planned stay period. So, this cost cannot be used to grow the partial route to next feature. For example, if a traveler arrived at a Best Buy store at 3:00 pm with a cost 9 and planned to stay 1 hour, the traveler will not depart within the period from 3:00 pm to 4:00 pm. In our algorithm, no time point within this window can be used to grow the partial route. If all costs at a time point calculated from different previous partial routes are within the stay period, the traveler cannot depart from this time point. In other words, it is impossible to grow a partial route from this time point. A number with a crossed out line means this cost is larger than or equal to the previously calculated cost from a different previous partial route. Due to space limitations, we don't display all the crossed out *Last Point* and *Last Time*. In real computation, if updating a cost for a partial route, the corresponding *Last Point* and *Last Time* also need to be updated.

As shown in the example of Figure 2, we are searching the BESTMTNN for the feature type sequence $< r, b >$. Point q is the query point. There are two points r_1 and r_2 from feature type r and one point $b1$ from feature type b. The planned stay at a point of feature type r is time unit 2 and will not be counted as travel cost. The query is asking for the best start time between 2 and 4. The TAMTG shown in the initial step stores the least travel time of point pairs for all time points for the sequence $< q, r >$ from query point q to feature type r and feature type sequence $< r, b >$ from feature type r to feature type b. Step 1 shows growing the partial route from the route containing only the query point q to feature type r. Step 2 illustrates growing the partial route from the route of $< q, r >$ to feature type b. The rule for growing the partial route in both steps is the same. The following example shows how partial route $R(q_4, r_{1,7})$ is grown to become partial route $R(q_4, r_{1,7}, b_{1,10})$. More specifically, we explain how the total cost 4 is calculated on the partial route $R(q_4, r_{1,7}, b_{1,10})$ that is for the route $R(q, r1, b1)$ at time point 10 in step 2. In step 1, after staying for 2 time units at time point 5 at point $r1$ on partial route $R(q_2, r_{1,5})$, time point 5 becomes 7. Please note that time unit 2 is a stay planned in advance so it is not counted as part of travel cost. Then we find the cost (least travel time) from $r1$ at time point 7 to $b1$ in the initial setup, which is 3. The arrival time point at $b1$ is $7+3=10$ and the total cost is $1+3=4$. So the total cost of arriving $b1$ at time point 10 is 4. The partial route for this calculation is $R(q_4, b_{1,7}, r_{1,10})$. The cost 5 in the same row at time 13 is the cost after staying at $b1$ for one extra time unit. This calculation continues for all possible partial routes. For a specific point at a specific time point, the least travel time is stored as the *Total Cost* (least travel time) for this partial route.

3.4 BESTMTNN Algorithm

Figure 3 illustrates the BESTMTNN algorithm. Briefly, the algorithm proceeds by gradually growing partial routes until finally a complete closed route is found. According to our analysis of this query's properties, a forward search strategy is preferred. In the following description of the

```
Input : Query point q, Time Window Constraints (TW),
        k: number of feature types, Distance metrics
        TAMTG : σ_vu(t) - cost from v to u at time t
Output : BESTMTNN route
BESTMTNN
 1.   Initialize : Add two fake new features of q as
 2.   first (feature 0) and last feature (feature k+1)
 3.   Find greedy route and get Current Search Bound
 4.   While there is permutation left
 5.        Clear Q and enqueue q into Q with cost 0
 6.        While priority Queue Q not empty
 7.             v = Dequeue(Q)
 8.             if (v is q and q is back-home query point
 9.             OR
10.             Minimum Total Cost >= Current
11.             Search Bound)
12.                  search in next permutation
13.             i = NextFeature(v)
14.             for (each node u in feature i)
15.                  for (every entry t_{i-1} within
16.                  time window of feature i − 1)
17.                  if (WithinTW(t_{i-1} + σ_vu[t_{i-1}], TW)
18.                  AND
19.                  ((C_u[t_{i-1} + σ_vu[t_{i-1}]] > σ_vu[t_{i-1}]+
20.                  C_v[t_{i-1}] OR i == 1))
21.                       C_u[t_{i-1} + σ_vu[t_{i-1}] =
22.                       σ_vu[t_{i-1}] + C_v[t_{i-1}]
23.                       Update related information
24.                  if (i has not been visited
25.                  AND
26.                  C_u[t_{i-1} + σ_vu[t_{i-1}]]+
27.                  σ_uq[t_{i-1} + σ_vu[t_{i-1}]) >
28.                  Current Search Bound
29.                       Enqueue(u, Q)
30.                  Maintain priority queue Q by moving
31.                  u forward in Q according to Minimum
32.                  Total Cost comparisons
33.   Report current route as BESTMTNN route and the
34.   starting time of BESTMTNN route as best starting
35.   time
```

Figure 3: BESTMTNN algorithm

BESTMTNN algorithm, assume a search order defined by $< F_1, F_2, F_3, \ldots, F_k >$. F_i represents feature type i.

In the priority Q, every node is attributed with a time series among which the i^{th} entry represents the partial route ending with the node at time point i and containing information about *Last Point*, *Last Time*, and *Total Cost* for this partial route as discussed in section 3.3. Thus a node in Q represents partial routes that end with the node for all qualified time points. For example, node u in the queue Q represents all the partial routes from query point q to u. When visiting a new feature type, a node from the new feature is added to the currently examined partial route to form a new partial route that is then enqueued into priority queue Q at the end for all time points within the time window of u. When visiting a feature type that has already been visited, the algorithm uses a label correcting approach [2] to modify the entries in a node according to the following conditions:

$C_u[t_{i-1} + \sigma_{vu}[t_{i-1}]]$ = minimum($\sigma_{vu}[t_{i-1}]$ $+C_v[t_{i-1}]$, $C_u[t_{i-1} + \sigma_{vu}[t_{i-1}]]$) where

$C_u[t]$:Least travel time of partial route from query point to u arriving at time t

$\sigma_{vu}[t]$:Least travel time from v to u starting at time t

The algorithm maintains a list of partial routes in the priority queue Q. The priority query is ordered by the *Minimum Total Cost* of all partial routes that end with the same node at all time points. The *Total Cost* of the partial route at a time point is the least travel time spent on the partial route at a time point. Then a *Minimum Total Cost* is the minimum of *Total Cost* at all time points. After a new partial route is formed or an existing partial route is updated the partial route can be moved forward if its *Minimum Total Cost* is smaller than that of the prior partial route in the queue. This condition guarantees that the following partial routes in the queue cannot have smaller least travel times even if these partial routes could be updated from prior partial routes in the queue. For example, assume the queue contains partial routes $< R(q, r2)$, $R(q, r1, b1)$, $R(q, r1, b2) >$ ordered by *Minimum Total Cost*. (For simplicity, time tag has been ignored.) A new partial route $R(q, r2, b1)$ is grown from the partial route $R(q, r2)$. The partial route $R(q, r1, b1)$ could be updated to $R(q, r2, b1)$ if $R(q, r2, b1)$ has smaller *Minimum Total Cost*. However, the *Minimum Total Cost* of the newly updated partial route $R(q, r2, b1)$ would still be bigger than that of $R(q, r2)$. So, if the *Minimum Total Cost* of $R(q, r2)$ is bigger than *Current Search Bound* or the length of time series it is safe to stop the search in the current permutation.

More specifically, the first step of the algorithm after initialization is to find a greedy route quickly and use its cost as first *Current Search Bound*. In a greedy route search, first a random point of feature type F_1 is picked, then the cost of travelling from query point q at the first qualified time point to this point is used as the current *Total Cost*. Then, a random point from feature type F_2 is picked to grow the partial route. This procedure continues until the search returns to the query point. It is possible that this approach cannot find a qualified greedy route. In this case, the length of the time series is used as the *Current Search Bound*.

The next step keeps all qualified partial routes $R(q_{tj}, P_{1,i,ti})$ as the first partial route set and enqueue all the partial routes into a priority queue Q that is ordered by *minimum Total Cost* of $C(q_{tj}, P_{1,i,ti})$. Here the point $P_{1,i,ti}$ is any point P_i of feature type F_1 at specific time ti and $C(q_{tj}, P_{1,i,ti})$ is the cost (least travel time) from q at time tj to point P_i of feature type 1 arriving at time ti.

In the following step the partial route at the head of priority queue Q is removed from the queue to become the current partial route. Assume this partial route is $R(q_{tj}, \ldots, P_{i-1,g})$ starting at time tj from q. On this partial route $P_{i-1,g}$ actually represents all partial routes ending with point $P_{i-1,g}$ for all qualified time points. If the next feature type F_i has not been visited, for every point $P_{i,l}$ in the feature type F_i, add the point $P_{i,l}$ to the current partial route, form a new partial routes $R(q_{tj}, \ldots, P_{i-1,g}, P_{i,l})$ and then calculate the new partial route costs $C(q_{tj}, \ldots, P_{i-1,g,tg}\ P_{i,l,tl})$ for all qualified time points among which $P_{i,l,tl}$ is the point P_l from the feature type F_i at time point tl. Finally the algorithm finds the *Minimum Total Cost* as minimum($C(q_{tj}$, $\ldots$, $P_{i-1,g,tg}\ P_{i,l,tl})$) for all qualified time points and enqueues the new partial route if the *Minimum Total Cost* is less than *Current Search Bound*. If the next feature type F_i has already been visited, there must be another partial route ending with $P_{i,l}$ in the queue Q. Look for this partial route in Q and compare the new least travel time on the new partial route to the previously calculated least travel

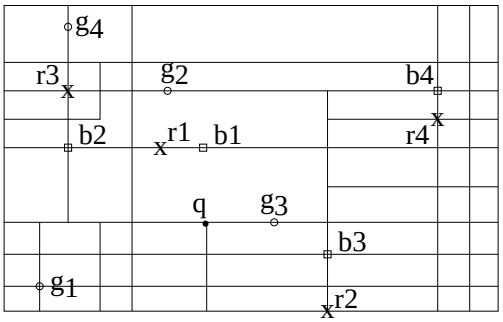


Figure 4: An Example of BESTMTNN

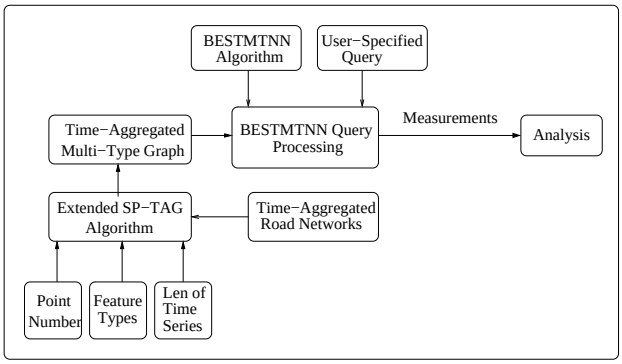


Figure 5: Experiment Setup and Design

time of the partial route ending with the same point $P_{i,l}$ for every qualified time point. If the new least travel time is less than the previous one at a time point, replace the previous partial route with the new partial route for this time point. Similar replacements should be done for all qualified time points.

In the last step in the iteration, the algorithm moves the partial route ending with the point $P_{i,l}$ forward in Q to keep the priority queue Q sorted by *Minimum Total Cost*.

This procedure will continue until a complete closed route is found for this permutation or the *Minimum Total Cost* from the current examined partial route is greater than the *Current Search Bound* or the length of the time series. At this time, it is possible that some partial routes remain in the priority queue. However, since the queue is sorted by *Minimum Total Cost*, it is impossible to find another complete closed route from the partial routes remaining in the queue with less travel time than the *Current Search Bound* or the length of the time series.

After searching all permutations, the BESTMTNN algorithm generates a complete closed route consisting of POIs with the best start time and a shortest travel time. The full turn-by-turn route can be found by simply checking with TAEPV that is used to generate TAMTG.

3.5 An Example of BESTMTNN Algorithm

Figure 4 illustrates how the BESTMTNN algorithm works on a spatial-temporal road network. For simplicity, we only show the algorithm for a specific permutation in the following. The full BESTMTNN algorithm works without predefined search order. In this example, q is the query point and there are three feature types r, b and g. Assume the current search sequence is $< r, b, g >$. First, the query point q is enqueued and then dequeued to calculate partial routes $R(q, r_1)$, $R(q, r_2)$, $R(q, r_3)$ and $R(q, r_4)$ for all qualified time points as described in section 3.3. (For simplicity, the time dimension of partial routes is not shown in this example.) $R(q, r_1)$ or $R(r_1)$ represents all partial routes ending with point r_1 for all time points. These partial routes are enqueued and sorted by *Minimum Total Cost* from all partial routes. Assume now the sorted partial routes in the priority queue are $< R(r_1), R(r_2), R(r_3), R(r_4) >$. Next, partial route $R(r_1)$, that is $R(q, r_1)$, is dequeued and grown by adding every point in feature type b to it. Four new partial routes $R(q, r_1, b_1)$, $R(q, r_1, b_2)$, $R(q, r_1, b_3)$ and $R(q, r_1, b_4)$ are generated and inserted into the queue such that the queue remains sorted. Assume the queue is $< R(r_2), R(b_1), R(b_2), R(r_3), R(r_4), R(b_4), R(b_3) >$ after sorting. At this time $R(r_2)$, that is $R(q, r_2)$, is dequeued and four new partial routes $R(q, r_2, b_1)$, $R(q, r_2, b_2)$, $R(q, r_2, b_3)$ and $R(q, r_2, b_4)$ are generated by adding every point in feature type b to it. Since feature type b was visited, every new partial route identified by its end point and arrival time at the end point is compared to the existing partial route if both of them share the same end point and time point. The partial route with shorter *Total Cost* is kept. After all the partial routes are grown and sorted, the partial route ending with point b_3 is moved forwarded to the head of the priority queue and the priority queue becomes $< R(b_3), R(b_1), R(b_2), R(r_3), R(r_4), R(b_4) >$. This procedure continues until a complete closed route ending with the query point q becomes the head of the priority queue. The BESTMTNN for this permutation is the complete closed route with the smallest least travel time value among all complete closed routes. In this example, the route with *Minimum Total Cost* among all routes represented by $R(q, r_2, b_3, g_3)$ is the BESTMTNN route.

4. EXPERIMENTAL EVALUATIONS

In this section, we present the results of various experiments to evaluate whether our BESTMTNN algorithm is optimal in finding the shortest path in terms of least travel time for the BESTMTNN query in real road network data sets. Specifically, we demonstrate performance of the BESTMTNN algorithm with respect to execution time using road network data with different properties related to the number of feature types, number of points in each feature type, length of query windows and length of time series.

4.1 The Experimental Setup

Experiment Platform Our experiments were performed on a PC with two 2.83 GHz CPUs and 4 GByte memory running the GNU/Linux Ubuntu 8.04.2 operating system. All algorithms were implemented in the C programming language.

Experimental Data Sets We evaluated the performance of the BESTMTNN algorithm for the BESTMTNN query with real road network data. The data set represents the static digital road map from the area of 3 miles in downtown Minneapolis, Minnesota. In this data there are a total of 786 nodes and 2106 edges. We synthetically generated different lengths of travel time series to create different Time-aggregated Road Networks. For evaluation purposes, data points were randomly picked from the nodes on the road networks to represent the points of interest (POIs) from different feature types.

There were four different parameters in our experimental

setup.

- Feature Type (FT): Feature type numbers from 2 to 7 to show the scalability of the algorithm in terms of number of feature types.
- Number of Points (NOP): Number of data points from 20 to 120 in each feature type.
- Length of Time Windows (TW): Length of query time windows from 20 to 120 for each feature type.
- Length of Time Series (TS): The available number of time points from 50 to 300 representing the long-term traffic patterns.

Experiment Design Figure 5 describes the experimental setup to evaluate the impact of design decisions on the relative performance of the BESTMTNN algorithm for the BESTMTNN query. The SP-TAG algorithm described in [4] calculated the shortest path in terms of least travel time between a pair of spatial points for a given start time. An extended version of SP-TAG algorithm randomly picks up points from the Time-Aggregated road networks as POIs in different feature types and generates TAMTG with different parameters choices related to number of feature types, number of points in a data set belonging to a feature type and different lengths of time series, based on the Time-Aggregated Road Networks. The BESTMTNN query processing engine takes different TAMTGs and user-specified queries to generate the performance measurements that are analyzed to evaluate the performance of the BESTMTNN algorithm. The user-specified query gives the start time interval during which the algorithm will find the best start time, the length of query time windows defining the qualified visit time interval at all feature types, stay time intervals at all feature types and the returning home time interval.

Our goal was to answer the following questions: (1) How do changes in number of feature types affect the scalability of the BESTMTNN algorithm? (2) How do differences in number of data points in each feature type affect the performance of the algorithm? (3) How do the lengths of query time windows at each feature type affect the performance of the algorithm? (4) How do the lengths of time series (number of time points) affect the performance of the algorithm?

4.2 Scalability of BESTMTNN with Respect to Feature Types

This section describes the scalability of BESTMTNN in terms of the number of feature types. In this experiment, the number of points in each feature type is 40, the length of query time windows for all feature types is 60, and the length of time series is 300 time units. The number of feature types changes from 2 to 7. Figure 6 shows that the algorithm runs less than 0.1 seconds when feature type number is 2,3 and 4 and runs for less than .5 seconds at feature type number 5. Run time increases to 3 seconds with 6 feature types and to a little less than 20 seconds for 7 feature types. The execution time increases dramatically when the number of feature types increases from 5 to 7 because the number of permutations increases dramatically, that is, the number of iterations required for searching BESTMTNN becomes significantly large. These results show that BESTMTNN is scalable with up to 7 feature types and for most daily life queries, BESTMTNN can give the query results quickly.

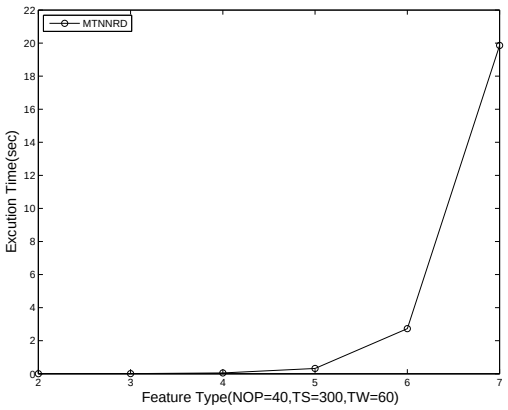


Figure 6: Scalability In Terms of Feature Type

4.3 The Effect of Number of Points in Feature Types on The Performance

In this section, we show how the POI density of the data sets affects the performance of the BESTMTNN algorithm. We tested BESTMTNN with feature type number 7, length of query time windows 60, time series units 300 and a changing number of points in each feature type from 20 to 120. The results shown in Figure 7 indicates the data density in the area covered by the experiment data set affects the BESTMTNN performance in a near linear fashion. When the data point number is 20, the running time is about 6 seconds. However, when the data point number reaches 120, that is, the total number of POI is $120 \times 7 = 840$, the running time is about 80 seconds. In our road network data set, the total node (point) number is 786, which indicates that there must be some POIs from different feature types that share the same location. This is a reasonable situation in daily life. For example, multiple business units may share the same building or mall. Meanwhile, a POI total of 840 means the data is extremely dense. It is probably necessary to partition this dense area into smaller areas in order to answer a BESTMTNN query faster.

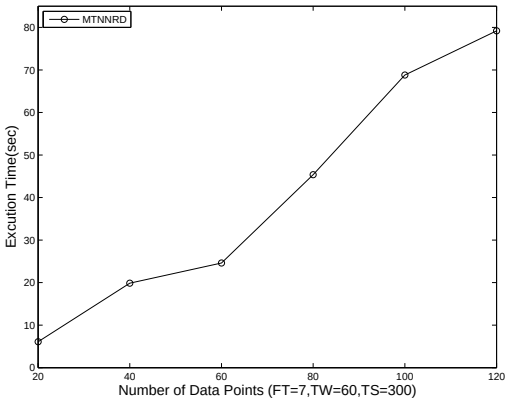


Figure 7: Effect of Number of Points

4.4 The Effect of Different Lengths of Query Time Windows on Performance

In this section, we illustrate the BESTMTNN performance under different lengths of query time windows at each feature type. We set the feature type number at 7, the number

of points in each feature type at 40 and the length of time series at 300. We took the same query windows size for all feature types and changed the lengths of query time windows from 20 to 120 units. Figure 8 shows that the BESTMTNN query is sensitive to the length of query window. Run times increase near linearly with increases of query time window. When the length of the time window is 20, the running time is less than 2 seconds. When the time window reaches at 120, the running time is just shy of 40 seconds. These results tell that it is beneficial for users to specify smaller query time windows. However, it is worth remembering that the chance of getting a BESTMTNN decreases as the size of the query time window becomes smaller.

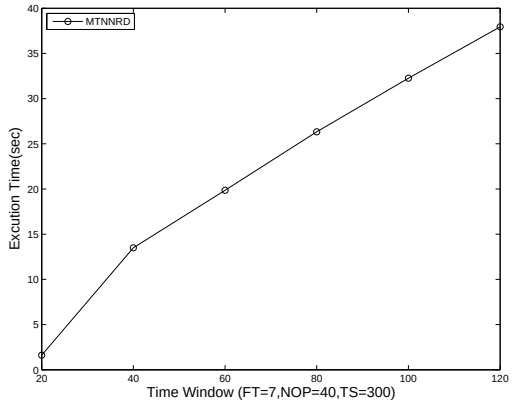


Figure 8: Performance Under Different Time Window Sizes

4.5 Effect of Different Lengths of Time Series on Performance

This experiment evaluates the effect of the total number of time series units in the spatial-temporal road network on the performance of BESTMTNN. Here we used number of feature types 7, number of points in each feature type 40 and the length of query time windows 50 or 60. The length of the time series was changed from 50 to 300. Please note the maximum meaningful length of query time window is 50 when the total available time series unit is 50. From the Figure 9, we can see that the total running times are between 16 and 21 seconds. The lengths of time series affect the BESTMTNN performance only a little and the changing pattern is not significant when the length of time series changes.

5. CONCLUSIONS AND FUTURE WORK

We identified the properties of a BESTMTNN query and formalized a BESTMTNN query problem on spatial-temporal road networks. We extended the EPV from spatial only to spatial-temporal road networks and utilized a special case of the extended EPV (TAEPV), TAMTG, in designing our BESTMTNN algorithm based on the label-correcting approach. In our experiment we evaluated the performance of the BESTMTNN algorithm in terms of number of feature types, number of points in each feature type, the length of query time windows and the length of time series of road networks.

In the future work, we plan to conduct the comparative experiments to characterize dominance zones of alternative

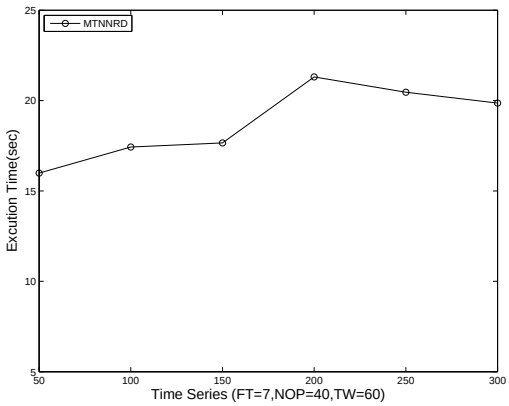


Figure 9: Effect of Time Series Length

choices for critical algorithm design decisions and extend the BESTMTNN algorithm to huge road networks by using a hierarchical TAMTG since it is extremely time-consuming to answer BESTMTNN query with current technologies without partitioning huge road networks.

6. REFERENCES

[1] L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 1981.

[2] B. Cherkassky, A. Goldberg, and T. Tadzik. Shortest paths algorithm: theory and experimental evaluation. *Mathematical Proggramming*, 1996.

[3] B. Ding, J. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, 2008.

[4] B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algotihms: a summary of results. In *SSTD*, 2007.

[5] Y.-W. Huang, N. Jing, and E. Rundensteiner. A semi-materialized view approach for route maintenance in Intelligent Vehicle Highway systems. In *ACM GIS*, 1994.

[6] Y.-W. Huang, N. Jing, and E. Rundensteiner. Hierarchical encoded path view for path query processing: an optimal model and its performance evaluation. *IEEE TKDE*, pages 409–432, May/June 1998.

[7] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992.

[8] G. Laporte, M. Gendreau, J. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *Intl. Transactions in Operational Research*, 2000.

[9] F. Li, D. Chen, and M. Hadjieleftherious. On trip planning queries in spatial databases. In *SSTD*, 2005.

[10] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting page level upper bound for multi-type nearest queries. In *ACM GIS*, 2006.

[11] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. In *VLDB Journal DOI 10.1007/s0078-006-0038-6*, 2007.

[12] M. Sharifzadeh and C. Shahabi. Processing optimal sequenced route queries using voronoi diagrams. In *Geoinformatica DOI 10.1007/s10707-007-0034-z*, 2008.