Processing Proximity Relations in Road Networks

Zhengdao Xu
University of Toronto
zhengdao@cs.toronto.edu

Hans-Arno Jacobsen
University of Toronto
jacobsen@eecg.toronto.edu

ABSTRACT

Applications ranging from location-based services to multi-player online gaming require continuous query support to monitor, track, and detect events of interest among sets of moving objects. Examples are alerting capabilities for detecting whether the distance, the travel cost, or the travel time among a set of moving objects exceeds a threshold. These types of queries are driven by continuous streams of location updates, simultaneously evaluated over many queries.

In this paper, we define three types of proximity relations that induce location constraints to model continuous spatio-temporal queries among sets of moving objects in road networks. Our focus lies on evaluating a large number of continuous queries simultaneously. We introduce a novel moving object indexing technique that together with a novel road network partitioning scheme restricts computations within the partial road network. These techniques reduce query processing overhead by more than 95%. Experiments over real-world data sets show that our approach is twenty times faster than a baseline algorithm.

Categories and Subject Descriptors

H. [**Information Systems**]; H.2.8 [**Database Applications**]: Spatial databases and GIS; H.3.3 [**Information Search and Retrieval**]: Information Filtering; H.4 [**Information Systems Applications**]: Location-based Services

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Location-based Services, Road Networks, Location Constraint, Location Query, Constraint Processing, Publish/Subscribe

1. INTRODUCTION

Motivation: Many emerging applications of moving objects require sophisticated query processing capabilities. Examples include the correlation of location position data for several continuously moving objects over time, the tracking of proximity relations among sets of moving objects, and the tracking of cost constraints among several sets of moving objects, such as time, distance, or budget. With *set of moving objects* we refer to the objects that are considered together in a single query. In this paper, unless stated otherwise, P denotes the set of moving objects for a single query. P comprises n moving objects, $P = \{p_1, p_2, ..., p_n\}$, where p_i designates moving object i.

In our approach, we develop query support that specifically targets the processing of *many sets of moving objects* simultaneously. For example, a location-based application may require that different *groups* of individuals are tracked at the same time. A dispatcher may need immediate notification, if the time or cost to assemble the individuals of given groups exceeds a threshold. Similarly, each individual may register to receive a warning, if the distance among the members of the group exceeds a threshold. An individual may be part of several different *sets of objects* that are tracked simultaneously.

The goals of this paper are to develop novel query processing capabilities to address these scenarios in road networks, where the *network distance*, defined as the distance between two points in the network, is the predominant distance measure. The specific measure could be a hop count or a cost function over edge weights. In particular, we are interested in processing large numbers of simultaneously executing queries over dynamically changing location data, rather than the processing of one single query over historic data.

The system architecture underlying our approach is depicted in Fig. 1. Location position updates are streamed into the system and trigger the evaluation of queries. Queries in this sense are standing and continuous queries that given a location update, continuously produced the desired output. The output is conveyed to the moving objects or to third party tracking applications built around the query processor.

Prime applications of this model are location-based services. Possible queries include the tracking of distances among sets of moving objects, such as the continuous monitoring of the distances among vehicles of a fleet, a convoy, or members of a workforce, and the computation of the closest or cheapest meeting point based on the underlying road network graph. For example, a service team may want to get notified if there is a location in the road network such that the total travel distance for the team to that location is below a given threshold (or if it deviates from a set "radius"), and where this location is. Similarly, individual team members may be bound to opportunistically optimize their own travel budgets and require that the maximum

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

travel distance to the meeting point does not deviate from a given cost. The challenge is to develop spatio-temporal query support for these scenarios.

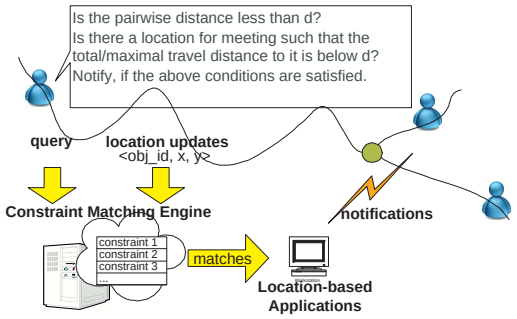

Figure 1: Data Flow and System Architecture

Other applications can be found in the online multi-player gaming sector, such as Doom 3, Quake or WorldofWarcraft, an emerging multi-billion dollar industry. Today, WorldofWarcraft alone is "inhabited" by a total of about 9 million people from around the globe. These games define virtual spaces where thousands of players interact simultaneously. The virtual spaces are divided into zones and areas connected by a network of pathways. Quake, for instance, offers a labyrinth of tunnels players traverse to move about the virtual spaces. Navigation is based on maps, defining a road network graph among the various spaces. In these games players define buddy lists to track who is online or who is in close proximity. Conceivable features to aid the players, are warning or alerting capabilities issuing alarms, if team members propagate too far apart to convene to take joint action, for example. The complexity of these games is daunting, features are required that advises cooperating players of ad hoc meeting places that minimize a given global cost, such as time, distance, or cost to travel through the virtual spaces. Alternatively, individual players of a group may desire to minimize their own cost function. The question of cost is extremely relevant in these games, due to the monetary value-based nature of these games resulting in entire eco-systems in and of themselves. The challenges for the underlying query processor is to simultaneously track thousands of queries for hundreds of thousands of online players.

Problem Statement: The above scenarios outline applications that require new types of spatio-temporal query support for query processing in road networks, including the monitoring of spatial relations, the formulation of queries to represent these relations, and the efficient, simultaneous evaluation of queries for large numbers of concurrently moving objects.

The objectives of this paper are to address these problems. First, we introduce three *distance measures* (i.e., the *pairwise-distance*, the *sum-distance*, and the *max-distance* defined in detail below) to define *proximity relations* for query processing in spatio-temporal applications such as the above. These measures operate over the location constellation of the moving objects by applying a network distance to all objects tracked. Each measure induces a *proximity relation* that minimizes or maximizes a property over the underlying network graph for the given measure and the given set of moving objects, P. We develop query processing algorithms for computing the value of the proximity relation and the point on the road network that yields the relation's value under the continuously changing constellation of objects in P. Furthermore, we develop query processing techniques that allow us to monitor the proximity relations of different sets of moving objects simultaneously. We refer to the monitoring objective of constraining a set of objects to maintain the value of a proximity relation below or above a threshold as a *location constraint* for the set of objects. Finally, we demonstrate how object movement histories and projected movement trajectories can be used to assess past matches and predict future matches, respectively.

Existing query types, such as the k nearest neighbor (kNN), continuous nearest neighbor (CNN), range query, closest pairs query and e-Distance joins [3, 8, 11, 6, 10] do not address our requirements. These approaches do not offer algorithms to detect and to monitor proximity relations of the kinds illustrated above. Proximity relations differ from the nearest neighbor query and range query, as proximity relations operate over, constrain, and monitor the *constellation* among *sets* of moving objects. Also, our approach is not based on static query points (like the kNN query) or fixed query ranges (like the CNN or range queries). A proximity relation offers different expressiveness from those other queries.

There is work on matching location constraints in Euclidean space [17, 16]. These approaches are based on indexing the space with spatial data structures, such as KD-trees and quadtrees. However, these solutions do not apply in a context where the space is defined by a road network graph. Moreover, these approaches do not address how to determine points on the road network graph (or points in the Euclidean space) that optimize a given cost measure, which is central to supporting the applications advocated in this paper.

Contributions and Organization: The contributions of this paper are five-fold: (1) The definition of three novel types of location constraints to represent different kinds of proximity relations among sets of moving objects in a road network with applications ranging from location-based services to multi-player online gaming (Sec. 2). (2) The development of efficient query processing algorithms to compute the three proximity relations and evaluate the location constraints (Sec. 3 and Sec. 5). (3) A novel graph partitioning technique to index the moving objects and optimize location constraint evaluation (Sec. 4). (4) An extension of the approach to evaluate constraints over past movement histories and projected movement trajectories (Sec. 6). (5) An extensive experimental evaluation of our approach on synthesized and real-world data sets collected from Google Ride Finder and Cabspotting (Sec. 7) shows the applicability of our approach to the above outlined scenarios. Related work is presented in Sec. 8.

2. NOTATION AND DEFINITIONS

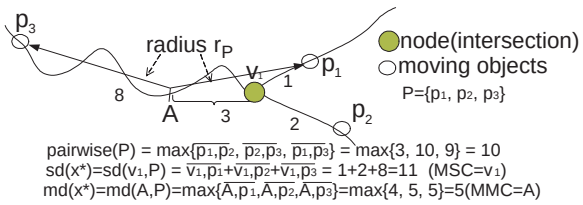

Figure 2: Pairwise, Sum and Max Distances

In this section we introduce and illustrate our notation, define

the distance measures underlying the proximity relations, and define the location constraints.

We use the common interpretation of a road network as a graph G consisting of a set of nodes and a set of edges connecting the nodes. We adopt the standard assumption that the road network structure is static [8, 3, 11, 6, 10]. Each edge of the graph has a non-negative weight. In practice the edge weight can model the length of the edge, the travel time, or travel cost, for example. The distance between moving objects on the road network is the network distance rather than the Euclidean distance. The *network distance* between two objects on the road network is the length of the shortest path connecting the objects.

With respect to graph, G, we refer to the *shortest path* and its *length* with reference to the path's start and end points. For example, $\overline{p_i, p_j}$ denotes both the shortest path from p_i to p_j and its length. A single edge in the graph with start and end nodes v_a, v_b, and its length are denoted by $[v_a, v_b]$. Furthermore, the shortest path between p_i and p_j via the nodes v_a and v_b is denoted by $\overline{p_i, v_a, v_b, p_j}$. Based on this notation, we now define three distance measures, which underly the proximity relations defined in this paper.

Given a set P of moving objects on the road network graph, G, the *pairwise-distance* measure of two objects p_i and p_j in P is defined as follows:

$$pd(p_i, p_j) \quad = \quad \overline{p_i, p_j}. \tag{1}$$

For example, for location-based advertising, the measure can capture the distance between a moving object and a point of interest on the road network. The objects could be close in Euclidean space but still have a large *pairwise-distance* in the network space. They may for instance be separated by a river, while the next bridge is far away.

Given a point, x, on the road network graph, G, the *sum-distance* measure of the set P w.r.t. x is defined as follows:

$$sd(x, P) \quad = \quad \sum_{i=1}^{n} \overline{x, p_i}. \tag{2}$$

$sd(x, P)$ is the sum of the length of the shortest paths between x and all locations of objects from P. For example, in Quake, *sum-distance* could model the cost for a group of players to assemble at location x.

The *max-distance* measure of set P w.r.t x is defined as follows:

$$md(x, P) \quad = \quad max_{i=1}^{n} \overline{x, p_i}. \tag{3}$$

$md(x, P)$ is the maximal length of all shortest paths between x and all locations of objects from P. For example, in a military context, Command & Control can use *max-distance* to monitor the maximal distance of all mobile units to a mobile supply truck.

We abbreviate *sum-distance* and *max-distance* by $sd(x)$ and $md(x)$, respectively, if P is clear from the context. Also, to emphasize that x is a point on a certain edge e, we sometimes write *sum-distance* as $sd_e(x)$. Likewise, the *max-distance* is represented as $md_e(x)$ given x is on edge e.

Each of the above distance measures induces a *proximity relation* as follows. The *max-pairwise-distance* relation for object set P is the maximal *pairwise-distance* of any pair of objects in P. It is denoted by $mpd(P)$ and defined as follows:

$$mpd(P) \quad = \quad max_{p_i, p_j \in P}\, pd(p_i, p_j). \tag{4}$$

The *min-sum-distance* relation for object set P is the smallest *sum-distance* for all x in the road network G.[1] Informally, it is the point on the road network from where the aggregated cost of reaching the points in P is lowest, provided shortest path are taken. Formally, the *min-sum-distance* is defined as follows:

$$msd(P) \quad = \quad min_{x \in G}\, sd(x). \tag{5}$$

A point, x^*, in G is called the *min-sum-center* (MSC) of object set P, if $sd(x^*) = min_{x \in G} sd(x)$. x^* is the point in the road network where the optimal cost is achieved for the current location of the moving objects represented in P.

The *min-max-distance* relation for object set P is the smallest possible *max-distance* for all x in the road network G. Informally, it is the point in the network from where any point in P can be reached with the lowest cost. Formally, the *min-max-distance* is defined as:

$$mmd(P) \quad = \quad min_{x \in G}\, md(x). \tag{6}$$

A point, x^*, in G is called the *min-max-center* (MMC) of object set P, if $md(x^*) = min_{x \in G} md(x)$. $r_P = md(x^*)$ is called the *radius* of P, since x^* can be reached from P within r_P.

To illustrate the above definitions, Fig. 2 shows three moving objects on a simple road network with three roads meeting at the intersection, v_1. The *max-pairwise-distance* of object set $P = \{p_1, p_2, p_3\}$ is $mpd(P) = max(\overline{p_1, p_2}, \overline{p_2, p_3}, \overline{p_1, p_3}) = max(3, 10, 9) = 10$. The *sum-distance* from point A to moving objects in P is $sd(A) = \overline{A, p_1} + \overline{A, p_2} + \overline{A, p_3} = 4 + 5 + 5 = 14$. Note, $sd(A)$ is not the minimum of the *sum-distance* to P. For instance, $sd(v_1) = \overline{v_1, p_1} + \overline{v_1, p_2} + \overline{v_1, p_3} = 1 + 2 + 8 = 11$, is smaller than $sd(A)$. v_1 is actually the *min-sum-center* of P ($MSC = v_1$). The *max-distance* from v_1 to P is $md(v_1) = max(\overline{v_1, p_1}, \overline{v_1, p_2}, \overline{v_1, p_3}) = max(1, 2, 8) = 8$. $md(v_1)$ is not the minimum of the *max-distance* to P. For instance, $md(A) = max(\overline{A, p_1}, \overline{A, p_2}, \overline{A, p_3}) = max(4, 5, 5) = 5$ is smaller than $md(v_1)$. A is the *min-max-center* of P ($MMC = A$) and the radius $r_P = md(A) = 5$.

Each of the above proximity relations induces a location constraint as follows, where d is referred to as the *alerting distance*:

1. The *pairwise constraint* is of the form $mpd(P) < d$. It is *satisfied* if the *max-pairwise-distance* of P is less than d.

2. The *min-sum constraint* is of the form $msd(P) < d$. It is *satisfied* if the *min-sum-distance* of P is less than d.

3. The *min-max constraint* is of the form $mmd(P) < d$. It is *satisfied* if the *min-max-distance* of P is less than d.

3. COMPUTING PROXIMITY RELATIONS

In this section, we develop query processing algorithms for computing the *max-pairwise-distance*, *min-sum-distance* and *min-max-distance* relations including the computation of optimal points MSC and MMC in the road network where the proximity relations assume these values.

We define the road network as a graph $G(V, E)$ with node set $V = \{v_1, v_2, ..., v_{|V|}\}$ and edge set $E = \{e_1, e_2, ..., e_{|E|}\}$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in the network. Objects can reside anywhere on an edge, not just at a node. Suppose object p_1 is on the edge $[v_1, v_2]$ and object p_2 is on the edge $[v_3, v_4]$. Since the shortest path $\overline{p_1, p_2}$ must go through exactly one node in $\{v_1, v_2\}$ and another node in $\{v_3, v_4\}$, the distance between p_1 and p_2 equals the smallest distance among $\overline{p_1, v_1, v_3, p_2}$, $\overline{p_1, v_1, v_4, p_2}$, $\overline{p_1, v_2, v_3, p_2}$, and $\overline{p_1, v_2, v_4, p_2}$, which represent four routes from p_1 to p_2.

[1] x could be anywhere on G, not just on nodes. The counterpart of the *min-sum-distance* in the Euclidean space is called geometric median (or 1-median).

In order to compute the pairwise node distance, we adopt the $network\ expansion$ algorithm [11].

We assume that each edge, e_i, with start node v_a and end node v_b, is associated with a cost distribution function $f_{e_i}(\theta)$ for $\theta \in [0,1]$. The function represents the cost of the movement between v_a and v_b. θ ($0 \leq \theta \leq 1$) represents a fraction of the distance traveled from the start node, v_a, w.r.t. the total length of the edge. For example, in function f_{e_i}, $f_{e_i}(0)$ represents the cost distribution at point v_a and $f_{e_i}(1)$ represents the cost distribution at point v_b. If θ_1 and θ_2 ($\theta_1 < \theta_2$) are the start and end positions of the trip on this edge, the cost of the trip is $F_{\theta_1,\theta_2} = \int_{\theta_1}^{\theta_2} f_{e_i}(\theta)d\theta$. The cost could be interpreted as travel time, travel expense, or delay on the road, which may not be proportional to the travel distance.

For simplicity of presentation, we assume that the cost distribution function is a positive constant k_{e_i}, which equals the length of the edge e_i. The cost of the trip, $F_{\theta_1,\theta_2} = k_{e_i}(\theta_2 - \theta_1)$, equals the length of the traveled distance on the edge from θ_1 to θ_2. However, our algorithm does not require that the cost function be a constant; f could be any continuous function. Furthermore, we associate the position of p_i with θ_{p_i} in the cost function f. In the following, we describe the query algorithms to compute the *pairwise-distance*, *min-sum-distance* and *min-max-distance* relations, respectively. We also present a number of formal properties that influence the evaluation of the proximity relations. All the lemmas presented below are formally proven in [15].

3.1 Computation of Pairwise Distance

The evaluation of the *max-pairwise-distance* relation is straightforward: Each *pairwise-distance* measure is computed and the maximum is recorded. The distances among object pairs computed in prior iterations of the algorithm can be reused. The general rule is that if a chain of the distances of m pairs of objects $\overline{p_i, p_{k_1}}, \overline{p_{k_1}, p_{k_2}}$, ..., $\overline{p_{k_{m-1}}, p_j}$ is already computed, then due to the triangle inequality, $\overline{p_i, p_j}$ is bounded from above by the sum of these m distances. If this sum is less than the current maximum recorded, $\overline{p_i, p_j}$ can be safely pruned without explicit computation. The smallest upper bound is used for pruning if multiple chains exist. We also use the Euclidean distance as lower bound to prune the computation. This is based on the fact that any path between two objects has a length larger than the Euclidean distance between them. So if the Euclidean distance (lower bound) of two objects is already larger than the alerting distance, this constraint cannot be matched and no further distance check is needed. The Euclidean lower bound check is used in all algorithms presented below, which we do not re-emphasize each time. This reduces the processing overhead because unnecessary computation is avoided.

3.2 Computation of Min-Sum Distance

For computing the *min-sum-distance*, it is important to first determine the location of the *min-sum-center* (MSC). First, we consider how to determine the position on the edge $[v_a, v_b]$ that achieves the smallest *sum-distance*. Below, we define an *edge split*, which turns out to be the candidate position for the MSC.

Due to the triangle inequality, $\overline{p_i, v_a} + [v_a, v_b] \geq \overline{p_i, v_b}$ and $\overline{p_i, v_b} + [v_b, v_a] \geq \overline{p_i, v_a}$. Therefore, there exists a *split point* es_{p_i} on the edge $[v_a, v_b]$ such that $\overline{p_i, v_a} + [v_a, es_{p_i}] = \overline{p_i, v_b} + [es_{p_i}, v_b]$ (es_{p_i} is $\frac{[v_a,v_b]+\overline{p_i,v_b}-\overline{p_i,v_a}}{2}$ away from v_a.) See Fig. 3 for an illustration. Intuitively, the *split point* and p_i bisect the cycle that consists of edge $[v_a, v_b]$ and shortest paths $\overline{p_i, v_a}$ (from p_i to v_a) and $\overline{p_i, v_b}$ (from p_i to v_b). If a point x is inside $[v_a, es_{p_i}]$, the shortest path from x to p_i goes through v_a, otherwise (x is inside $\overline{es_{p_i}, v_b}$) it goes through v_b. We call this split point the *edge split* (or $e-$split, for short) w.r.t. p_i.

Let the position of x on the edge be θ (or be distance $\theta[v_a, v_b]$ away from v_a), then the shortest path from p_i to x is a piecewise linear function, $D_{p_i}(\theta)$, of θ, with split point $\theta_{es_{p_i}}$:

$$D_{p_i}(\theta) = \begin{cases} \overline{p_i, v_a} + F_{0,\theta} & \text{if } 0 \leq \theta < \theta_{es_{p_i}} \\ \overline{p_i, v_b} + F_{\theta,1} & \text{if } \theta_{es_{p_i}} \leq \theta \leq 1 \end{cases} \quad (7)$$

The *sum-distance* from object set P to x is a piecewise linear function: $sd(x) = \sum_{p_i \in P} D_{p_i}(\theta)$ and the *min-sum-distance* on the edge is $min_{0 \leq \theta \leq 1} sd(x)$. Since the cost function is piecewise linear with intervals delimited by points in $ES = \{0, \theta_{es_{p_1}}, ..., \theta_{es_{p_n}}, 1\}$, it achieves the minimum or maximum at the end points in ES.

Therefore, one potential position of the MSC coincides with the position of one point in ES. We emphasize that this is only one potential position of the MSC. In fact, the position of the MSC might not be unique. Consider the case of two objects, the MSC of the pair could be anywhere on the shortest path between the two objects, since the *sum-distance* to any point on the edges along the shortest path is constant. For a non-linear cost function, the optimal points are computed in each interval (not necessarily on the end point), since the extreme value could be achieved anywhere in the interval.

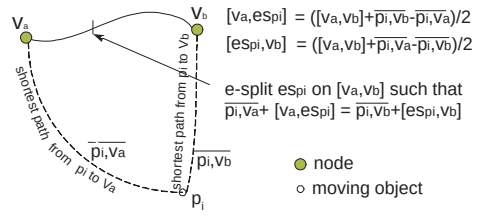

Figure 3: Edge Split

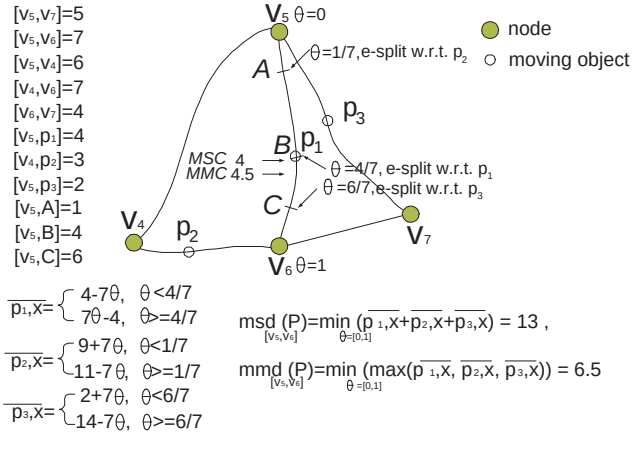

Figure 4: Computation of Min-Sum & Min-Max Distances

Fig. 4 shows part of the road network from Fig. 6, where we compute the *min-sum-distance* for the object set $P = \{p_1, p_2, p_3\}$ on the edge $[v_5, v_6]$. The search for the MSC can be restrained

inside this partial road network (called a *partition*) because the edges $[v_5, v_4]$ and $[v_5, v_7]$ are also the shortest paths between v_5 and v_4, and between v_5 and v_7, respectively. We further elaborate on the rationale behind this in Sec. 4.

We assume the start point for the cost distribution function of $[v_5, v_6]$ is v_5; that is at v_5, $\theta = 0$, and at v_6, $\theta = 1$. As shown in Fig. 4, the $e-$splits w.r.t. to p_1, p_2 and p_3 are $\theta_{es_{p_1}} = 4/7$ (point B), $\theta_{es_{p_2}} = 1/7$ (point A), and $\theta_{es_{p_3}} = 6/7$ (point C), respectively. Therefore, for any point x (corresponding to θ in the cost distribution function) on edge $[v_5, v_6]$, the length of the shortest path $\overline{p_1, x}$, $\overline{p_2, x}$ and $\overline{p_3, x}$ are piecewise linear functions shown on the bottom left of Fig. 4. Therefore, for $0 \leq \theta < 1/7$, $sd_{[v_5,v_6]}(x) = 15 + 7\theta$, the *sum-distance* is in the range [15, 16) (see the chart in the middle of Fig. 5). For $1/7 \leq \theta < 4/7$, $sd_{[v_5,v_6]}(x) = 17 - 7\theta$, the *sum-distance* is in the range [16, 13). For $4/7 \leq \theta < 6/7$, $sd_{[v_5,v_6]}(x) = 9 + 7\theta$, the *sum-distance* is in the range [13, 15). For $6/7 \leq \theta \leq 1$, $sd_{[v_5,v_6]}(x) = 21 - 7\theta$, the *sum-distance* is in the range [15, 14]. The *min-sum-distance* of 13 is achieved at the $e-$split with $\theta = 4/7$, which coincides with $\theta_{es_{p_1}}$. The *min-sum-distance* on each edge can be computed with the same algorithm, showing that the *min-sum-distance* on the edge $[v_5, v_6]$ is also the *min-sum-distance* of the whole network ($\theta_{MSC} = 4/7$ on the edge $[v_5, v_6]$). When the *sum-distance* is linear in each interval, only the *sum-distance* at the end points in ES needs to be considered. For non-linear functions, the extreme values in each interval need to be examined.

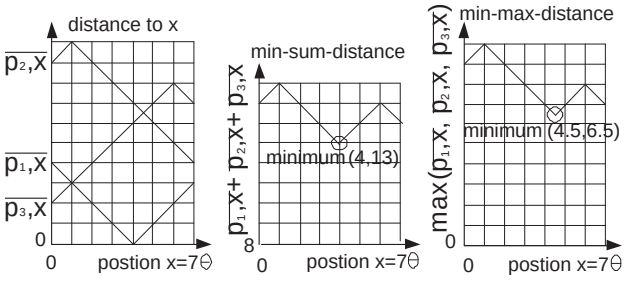

Figure 5: Min-Sum-Distance & Min-Max-Distance

The search space of $e-$splits can be pruned based on Lemma 1. To introduce Lemma 1, we define the notion of an $l-$vicinity. Let x be a point in tbe road network, G, we say that the network location y is in $l-$vicinity of x if $l \geq \overline{x, y} > 0$ (x itself is not considered to be in $l-$vicinity). We can always choose a small $l = \epsilon$ such that the $l-$vicinity of x contains neither any object from P nor any of their $e-$splits. Such an $l-$vicinity is called an $\epsilon-$vicinity. The boundary of the $\epsilon-$vicinity in the road network is a set of split points called the $\epsilon-$splits (see Fig. 6 for an illustration.) Nodes connecting to multiple edges can have many $\epsilon-$splits (e.g., v_3 in Fig. 6 has 3 $\epsilon-$splits).

Let the $\epsilon-$splits of point x be $\epsilon_x = \{\epsilon_x^1, \epsilon_x^2, ..., \epsilon_x^k\}$. Suppose that $P_{\epsilon_x^i}$ ($\subset P$) is the subset of P on ϵ_x^i's side. That is, ϵ_x^i is on the shortest path from x to any object in $P_{\epsilon_x^i}$. For the $\epsilon-$splits of MSC, $\epsilon_{MSC} = \{\epsilon_{MSC}^1, \epsilon_{MSC}^2, ..., \epsilon_{MSC}^k\}$, the following property holds.

LEMMA 1. $|P_{\epsilon_{MSC}^i}| \leq \sum_{j \neq i} |P_{\epsilon_{MSC}^j}|$.

Here $|P_{\epsilon_{MSC}^i}|$ represents the number of objects in the set $P_{\epsilon_{MSC}^i}$. Lemma 1 states that the number of shortest paths coming from

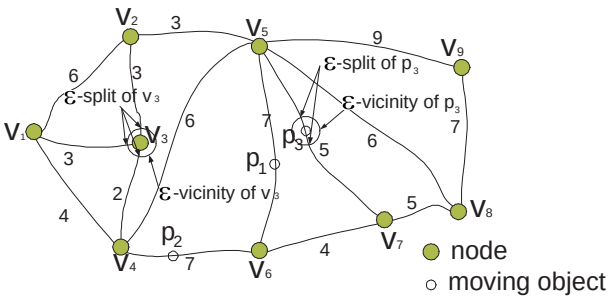

Figure 6: $\epsilon-$vicinity and $\epsilon-$split

one $\epsilon-$split is outnumbered by the number of shortest paths from all the other $\epsilon-$splits combined, because if it were not so, we could always reduce the *sum-distance* by moving the MSC towards ϵ_{MSC}^i (proven in [15]). This implies that the $e-$split, es, can not be the *min-sum-center*, if $|P_{\epsilon_{es}^i}| > \sum_{j \neq i} |P_{\epsilon_{es}^j}|$. As the search proceeds from one $e-$split to the next on the edge, the value of $|P_{\epsilon_{es}^i}| - \sum_{j \neq i} |P_{\epsilon_{es}^j}|$ can only change by 2, because the shortest path from only one object to the $e-$split may change its route. Therefore, we can safely move forward to the next $((|P_{\epsilon_{es}^i}| - \sum_{j \neq i} |P_{\epsilon_{es}^j}|)/2 + 1)$-th $e-$split, pruning all the $e-$splits in between. Lemma 1 suggests the following algorithm for computing the *min-sum-distance* and the position of the MSC. In Lines 6, 7, we see the algorithm skips a number of e-splits if the above condition is not true.

Algorithm MIN-SUM_DISTANCE(ObjSet P, RoadNetwork G)
1. **for** each edge $[v_a, v_b]$ in the road network G
2. $ES_{[v_a,v_b]}$ = $e-$splits w.r.t. $P \cup \{v_a, v_b\}$;
3. **for** each $ES_{[v_a,v_b]}$
4. **for** each point es in $ES_{[v_a,v_b]}$
5. find the ϵ-splits ϵ_{es} of es
6. **if** $|P_{\epsilon_{es}^i}| > \sum_{j \neq i} |P_{\epsilon_{es}^j}|$
7. skip the next $(|P_{\epsilon_{es}^i}| - \sum_{j \neq i} |P_{\epsilon_{es}^j}|)/2$ points in $ES_{[v_a,v_b]}$
8. **else** /* $|P_{\epsilon_{es}^i}| \leq \sum_{j \neq i} |P_{\epsilon_{es}^j}|$ */
9. **if** $minsum > sd(es)$
10. $minsum$ = $sd(es)$;
11. msc = es;
12. **return** [$minsum$,msc];

3.3 Computation of Min-Max Distance

In this section, we describe how the *min-max-distance* and *min-max-center* (MMC) of P are determined. Recall, $md(x)$ is the *max-distance* from point x to objects in P, and $r_P = md(x^*)$ is the radius of P (where x^* is the MMC.)

Assume the position of x on the edge is θ and the $e-$split w.r.t. p_i is es_{p_i}, then the shortest path from p_i to x is a piecewise linear function, $D_{p_i}(\theta)$, of θ, as in Eq. 7. The *max-distance* is $md(x) = \max_{p_i \in P} D_{p_i}(\theta)$. The *min-max-distance* on the edge is $min_{0 \leq \theta \leq 1} md(x)$, which is the lowest point in the upper envelop[2] of $D_{p_i}(\theta)$. In Fig. 4, the position of $e-$splits w.r.t. objects in P are $1/7, 4/7, 6/7$. For any point x (corresponding to θ in the cost distribution function) on edge $[v_5, v_6]$, if $0 \leq \theta < 1/7$, $md_{[v_5,v_6]}(x) = 9 + 7\theta$, then the *max-distance* is in range [9, 10) (see the right chart in Fig. 5). For $1/7 \leq \theta < 4/7$, $md_{[v_5,v_6]}(x) = 11 - 7\theta$, the *max-distance* is in the range [10, 7). For $4/7 \leq \theta < 6/7$, either $\overline{p_2, x}$, or $\overline{p_3, x}$ could be

[2]The upper envelop of functions f_1, f_2,...,f_n is max$\{f_1, f_2, ...f_n\}$.

the maximum. If $4/7 \leq \theta < 9/14$, $md_{[v_5,v_6]}(x) = 11 - 7\theta$ ($\overline{p_2,x}$ is the maximum), the *max-distance* is in range [7, 6.5). If $9/14 \leq \theta < 6/7$, $md_{[v_5,v_6]}(x) = 2 + 7\theta$ ($\overline{p_3,x}$ is the maximum), the *max-distance* is in range [6.5, 8). For $6/7 \leq \theta \leq 1$, $md_{[v_5,v_6]}(x) = 14 - 7\theta$, the *max-distance* is in range [8, 7]. The *min-max-distance* of 6.5 is achieved at $\theta = 9/14$. The *min-max-distance* on each edge is computed with the same algorithm showing that the *min-max-distance* on edge $[v_5, v_6]$ is also the *min-max-distance* of the whole network ($\theta_{MMC} = 9/14$ on edge $[v_5, v_6]$). Note the MMC does not have to coincide with any $e-$splits or the end points of the edge.

Lemma 2 below captures the intuition that the MMC must have the same distance, r_P (the radius as defined above), to at least two objects in P. If this were not the case, the *min-max-distance* could always be reduced by moving the MMC a bit closer towards the object that has the furthest distance.

LEMMA 2. *At least two objects in P have the distance r_P to the min-max-center of P.*

Lemma 2 shows that the MMC is like a center of P with equal distance to at least two objects. We now state Lemma 3 that shows the inequality relation between the *max-distances* of different locations on the road network. Lemma 2 and Lemma 3 lead to Theorem 1, which allows us to prune the computation by skipping entire edges.

LEMMA 3. *For any two points x and y on the road network and a set P of objects, $md(x, P) \leq md(y, P) + \overline{x, y}$.*

x and y are interchangeable in the above lemma. This shows that there is correlation between the *max-distances* to two points in the road network. One *max-distance* cannot deviate from the other one by too much (within distance $\overline{x, y}$). This gives rise to the following theorem to prune the search for the *min-max-center*.

THEOREM 1. *Let $p_i \in P$, then $md(p_i, P)/2 \leq r_P \leq md(p_i, P)$. If the min-max-center is on the edge $[v_a, v_b]$, r_P is bounded from below by $(md(v_a) + md(v_b) - [v_a, v_b])/2$ and bounded from above by the smaller of $md(v_a) + [v_a, v_b]$ and $md(v_b) + [v_a, v_b]$.*

Theorem 1 gives us the bounds of r_P when checking a specific edge. The proof is in [15]. This edge can be safely pruned if the lower bound of r_P is above the computed *min-max-distance*, because the *min-max-distance* on that edge must be larger than the *min-max-distance* recorded. Below, we show the algorithm MIN-MAX_DISTANCE for computing the *min-max-distance* and the MMC for object set P. In Line 4, the lower bound of r_P is used to prune the edge, as stated in Theorem 1.

Algorithm MIN-MAX_DISTANCE(ObjSet P, RoadNetwork G)
1. $ub = md(p_i)$;// for any $p_i \in P$
2. $mmc = p_i$;
3. **for** each edge $[v_a, v_b]$ in the road network G
4. **if** $(md(v_a) + md(v_b) - [v_a, v_b])/2 \leq ub$
5. $ES_{[v_a,v_b]}$ = $e-$splits w.r.t. $P \cup \{v_a, v_b\}$;
6. $md_{[v_a,v_b]}(x) = \max_{p_i \in P} D_{p_i}(\theta)$;
7. $minmax_{[v_a,v_b]} = min_{0 \leq \theta \leq 1} md_{[v_a,v_b]}(x)$;
8. $mmc_{[v_a,v_b]} = \{x | md_{[v_a,v_b]}(x) = minmax_{[v_a,v_b]}\}$;
9. **if** $minmax_{[v_a,v_b]} < minmax$
10. $minmax = minmax_{[v_a,v_b]}$;
11. $mmc = mmc_{[v_a,v_b]}$;
12. **if** $minmax < ub$
13. $ub = minmax$;
14. **return** $[minmax, mmc]$;

4. ROAD NETWORK PARTITIONING

The query processing algorithms for computing the proximity relations in Sec. 3 are expensive despite the pruning techniques. This is caused by the sequential search on every edge (e.g., searching MSC and MMC). In this section, we develop techniques based on road network partitioning to prune the search space and to improve the efficiency of the algorithms.

We define the *boundary* of the road network graph as a sequence of edges that forms a cycle bounding a planar embedding of the graph G (For a disconnected graph, each separate graph component has a *boundary*.) For example, in Fig. 7, the *boundary* of the graph is the cycle formed by $v_1 - v_2 - v_3 - v_4 - v_5 - v_6 - v_7 - v_1$. We call the shortest path connecting two nodes on the *boundary* of the graph a *cut*. Although the edges of G are un-directional, a cut is defined as a *directional path*, expressed as $\overrightarrow{v_a, v_b}$ if the path is from v_a to v_b. A cut divides G into two parts, defining the boundary between the left part, G^0, and the right part, G^1.

For example, the shortest path $\overrightarrow{v_5, v_2}$ (in Fig. 7) partitions the whole road network into two parts, the right part, G^1, enclosed by the nodes $v_2 - v_3 - v_4 - v_5 - v_2$, and the left part, G^0, enclosed by the nodes $v_2 - v_5 - v_6 - v_7 - v_1 - v_2$.

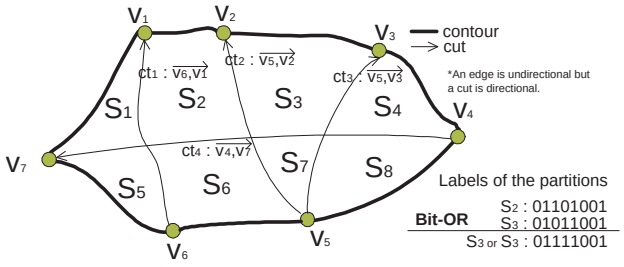

Figure 7: Partitioning a Road Network

In our approach, a set of k cuts, $CTS = \{ct_1, ct_2, ..., ct_k\}$, is chosen such that the cuts divide the graph into partitions of roughly the same size. To achieve this, the first cut is chosen such that it has end nodes on the boundary and it bisects the current space into halves of roughly equal size. In each subspace, a second cut is chosen in the same manner so that it bisects the sub-space. As long as the end nodes are selected from the boundary of the original graph (not from nodes on a cut), no cuts, generated in this manner, intersect with each other. This gives rise to a set of "parallel" (non-intersecting) cuts. We repeat this process to retrieve a second set of cuts. The two sets form a grid-like partitioning. This approach works well for "well-formed" city maps where most of the streets are parallel forming a grid-like pattern.

The number of cuts, k, depends on the size of the road network. The parameter k is proportional to the square root of the size of the road network (the number of edges in our implementation), because the number of partitions created is roughly proportional to the square of k. The determination of cuts for arbitrary graphs is deferred to future work. In our experiments, we pre-process cuts of arbitrary graphs to ensure the quality of the cuts determined. This is a one-time setup cost.

The set CTS uniquely describes a graph partitioning scheme. Each cut ct_i divides G into two parts, denoted by G_i^0 and G_i^1. Conceptually, ct_i belongs to both parts. The cuts in CTS divide the whole graph into partitions, each partition is bounded on one side by a cut or an edge of the *boundary*. A cut partitions not only the graph but also the shortest path between the objects in the same partition as stated in the following lemma.

LEMMA 4. *No shortest path between objects in the same partition goes through a cut from one partition to another.*
The intuition is that if this were not the case, the path outside the partition could be shortcut by the very cut it trespasses giving rise to an even shorter path between the objects. (The formal proof is in [15].)

A partition is uniquely described by a label (a string) comprised of k keys. Each key represents a cut with two bits. The i-th cut, ct_i, is expressed with the key in the $(2i-1)$-th and $2i$-th bits of the key string. The two bits are 01 if the partition is on the right side of the cut, 10 if the partition is on the left side of the cut, and 11 if the partition is on both sides of the cut. For example, in Fig. 7, partition S_2 is on the right of cut ct_1 and ct_4, and on the left of cut ct_2 and ct_3. Therefore, the label for partition S_2 is 01101001. Likewise, the label for partition S_3 is 01011001. If we combine the labels of a set of partitions with bitwise-OR, the result is the label of the smallest partition that encloses these partitions, as stated in Lemma 5 (for the proof see [15].)

LEMMA 5. *The bitwise-OR of the labels of a set of partitions S_i ($1 \leq i \leq n$) represents the label of the smallest partition ($S_{combined}$) enclosing all S_i.*

In Lemma 5, by smallest partition, we mean that any partition enclosing S_i ($1 \leq i \leq n$) must also contain $S_{combined}$. For instance, combining the labels for partitions S_2 and S_3 (01101001 OR 01011001), the result is 01111001, which is exactly the label for the partition with S_2 and S_3 combined (it is on the right side of ct_1 and ct_4, on both sides of ct_2, and on the left side of ct_3 (Fig. 7).)

We refer to the *unit partition* as a partition that is not divided by any cut. The moving objects can be indexed with unit partitions. An object on a cut is assumed to be inside either partition that bounds the cut. The partitions are structured as an array and each element of the array contains an attribute label representing the partition. Also each edge has a reference to one entry of the array to indicate which partition it is located in. Thus, the edge an object is on, the associated partition, and the relation to the cut are directly accessible. According to Lemma 4, the computation of pairwise distances can be restricted to a partition if all objects in P are inside the partition. No edge or node outside the partition need to be considered because a shortest path cannot traverse other partitions. For each partition S, we define its *diameter*, $dia(S)$, as the maximal length of the shortest path between any two points in S. This definition is slightly different from that in [13], yet the computation is similar. By definition, the diameter is the longest possible network distance in the partition. It can be used to prune the computation, as stated in Lemma 6.

LEMMA 6. *The pairwise distance computation is strictly restrained to the partition where both objects are located. If all the objects in set P are inside a partition S, then the max-pairwise-distance of P is no larger than $dia(S)$.*
For the same reason, the graph partition also restrains the search for MSC and MMC, as stated in Lemma 7 (Proof is given in [15].)

LEMMA 7. *If all the objects in set P are inside partition S, then MSC and MMC of P are also inside partition S.*
Lemma 7 establishes the fact that the location of the MSC and the MMC are relatively close to P (inside the same partition). This fact can be exploited to prune the search space. With road network partitioning, we first compute the smallest partition that contains all objects of P with bitwise-OR, as described above. For computing *max-pairwise-distance*, *min-sum-distance* and *min-max-distance*, we only consider the partial road network that is inside the partition rather than the whole road network. The improved query processing algorithm, PROXIMITY is sketched below. Note that it restricts the computation of the constraints to part of the road network, g, determined by the smallest partition enclosing all the objects involved (Lines 1, 2) rather than all the edges in the road network. The parameter T specifies the type of proximity relation evaluated. Our experiments show a 10-30% reduction in processing overhead under random object movement and a reduction by 80% under a clustered movement pattern (see Sec. 7.)

Algorithm PROXIMITY(ObjSet P, Type T)
1. combined = label(p_1)||label(p_2)||...label(p_n);
2. g = partition(combined); /*partial network g*/
3. **if** (T = PAIRWISE)
4. **return** `PAIRWISE_DISTANCE(`P`, `g`);`
5. **if** (T = MIN-SUM)
6. **return** `MIN-SUM_DISTANCE(`P`, `g`);`
7. **if** (T = MIN-MAX)
8. **return** `MIN-MAX_DISTANCE(`P`, `g`);`

5. CONSTRAINT EVALUATION

In this section, we show how to apply the partitioning technique developed in the previous section to the problem of matching location constraints. Recall that the three location constraints (defined at the end of Sec. 2) are *satisfied* if the *max-pairwise-distance*, *min-sum-distance*, and *min-max-distance* of a set P are below some alerting distance d.

Given the partition information of where objects are located, the results for some constraints can be directly determined. First, the labels of the partitions that contain the objects involved in constraints are identified. The labels are then combined with bitwise-OR to retrieve the smallest partition that contains all the objects involved. The partition-based matching of the constraints is based on the following observations: If the objects in P are located inside the n partitions S_1, S_2, ... S_n (not necessarily different) and S is the smallest partition that encloses P then:

1. The *max-pairwise-distance* of P is bounded from above by $dia(S)$. (This is the result of Lemma 6.)

2. The *min-sum-distance* of P is bounded from above by $(n-1)dia(S)$. (This upper bound can be attained by choosing any object as the MSC.)

3. The *min-max-distance* of P is bounded from above by $dia(S)$. (Within distance $dia(S)$, one can reach any point in S from any object in P.)

4. The *max-pairwise-distance* of P is bounded from below by $max(pdist(S_i, S_j))$ ($1 \leq i, j \leq n$) [3]. (The *pairwise-distance* of the objects is always bounded from below by the distance between the partitions where they are located.)

5. The *min-sum-distance* of P is bounded from below by $\sum_{1 \leq i \leq \lfloor n/2 \rfloor} pdist(S_{2i-1}, S_{2i})$ (for each pair of objects, one in partition S_{2i-1} and the other in partition S_{2i}, there exists a path between them by way of MSC. The length

[3] $pdist(A, B)$ returns the minimal distance between two partitions, A and B. That is $pdist(A, B) = min(\{\overline{x,y} | x \in A, y \in B\})$.

of the path must be no less than the distance between the partitions.)

6. The *min-max-distance* of P is bounded from below by $max(pdist(S_i, S_j))/2$. (the *min-max-distance* is at least half the distance between any two partitions.)

The partition-based matching is based on the fact that if the upper bound is already less than the alerting distance, d, then the constraint must be *satisfied*. If the lower bound is greater than the alerting distance, then the constraint must be *unsatisfied*.

The above observations help to prune the location constraints, since the matching results for many constraints can already be determined based on the partition information alone. The detailed algorithm for partition-based evaluation is as follows. We assume that the moving objects in the road network continuously send location information to the constraint matching engine (see Fig. 1). The Location_Update_Evaluation algorithm is triggered with each location update message. If the object, p, changes its partition, the Partition_Based_Evaluation Algorithm is called. This algorithm evaluates the constraints involving p based on the above observations. This results in some constraints being *satisfied* or *unsatisfied*, while others remain *uncertain* (i.e., cannot be determined based on partition information alone.) Only *uncertain* constraints are computed explicitly with Algorithm Match_Uncertain_Constraints. If the object p does not change its partition, then only the *uncertain* constraints have to be reevaluated, entirely pruning the constraints that are *satisfied* or *unsatisfied* based on the partition information. Match_Uncertain_Constraints calls PROXIMITY to restrict the computation to the partial network (as in Sec. 4), which reduces the processing overhead significantly. The algorithms are expressed in pseudo code below.

Algorithm Location_Update_Evaluation(*MobileObject p*)
1. $p.previous\_partition = p.current\_partition$;
2. $p.current\_partition$ = `Find_New_Partition`(p);
3. **if** $p.previous\_partition \neq p.current\_partition$
4. `Partition_Based_Evaluation`(p);
5. `Match_Uncertain_Constraints`(p);

Algorithm Partition_Based_Evaluation(*MobileObject p*)
1. **for** each Constraint c that p is associated with
2. **let** $P = \{p_1, p_2, ..., p_n\}$ be the set of bodies in c;
3. combined = label(p_1)||label(p_2)||...label(p_n);
4. $c.p\_b\_result$ = *uncertain*;
5. **if** (c.type=PAIRWISE)
6. **if** (dia(partition(combined)) $< c.d$)
7. $c.p\_b\_result$ = [*satisfied*,-];
8. **if** (max(pdist(partition(label(p_i)),partition(label (p_j)))) $> c.d$)
9. $c.p\_b\_result$ = [*unsatisfied*,-];
10. **if** (c.type=MIN-SUM)
11. **if** ($(n-1)$dia(partition(combined)) $< c.d$)
12. $c.p\_b\_result$ = [*satisfied*,p_i];/*any p_i*/
13. **if** ($\sum_{1 \leq i \leq \lfloor n/2 \rfloor}$pdist(partition(label($p_{2i-1}$)), partition(label(p_{2i})) $> c.d$)
14. $c.p\_b\_result$ = [*unsatisfied*,-];
15. **if** (c.type=MIN-MAX)
16. **if** (dia(partition(combined)) $< c.d$)
17. $c.p\_b\_result$ = [*satisfied*,any point in partition (combined)];
18. **if** (max(pdist(partition(label(p_i)),partition(label(p_j))))$/2 > c.d$)
19. $c.p\_b\_result$ = [*unsatisfied*,-];

Algorithm Match_Uncertain_Constraints(*MobileObject p*)

1. **for** each Constraint c that p is associated with
2. **if** $c.p\_b\_result$ = *satisfied*
3. $c.result$ = *satisfied*;
4. **if** $c.p\_b\_result$ = *unsatisfied*
5. $c.result$ = *unsatisfied*;
6. **if** $c.p\_b\_result$ = *uncertain*
7. **if** (c.type=PAIRWISE)
8. **if** (`PROXIMITY`($c.P$, `PAIRWISE`)$< c.d$)
9. $c.result$ = [*satisfied*,-];
10. **else**
11. $c.result$ = [*unsatisfied*,-];
12. **if** (c.type=MIN-SUM)
13. ms = `PROXIMITY`($c.P$, `MIN-SUM`);
14. **if** ($ms.minsum < c.d$)
15. $c.result$ = [*satisfied*,$ms.msc$];
16. **else**
17. $c.result$ = [*unsatisfied*,-];
18. **if** (c.type=MIN-MAX)
19. mm = `PROXIMITY`($c.P$, `MIN-MAX`);
20. **if** ($mm.minmax < c.d$)
21. $c.result$ = [*satisfied*,$mm.mmc$];
22. **else**
23. $c.result$ = [*unsatisfied*,-];

Scaling to large road networks: Besides processing increasing constraint loads, scaling to large road networks that go beyond a single city is of practical concern. In this case, the number of network distance computations and the number of location updates impose additional challenges.

To address this point, we introduce a system design parameter, d_{max}, which bounds the alerting distances of all constraints from above. For the targeted applications, we envision d_{max} to be small (on the order of kilometers or less). Objects farther apart than d_{max} can be safely ignored as their constraints wont match. This parameter can be used to segment the road network into smaller sub-graphs. For example, the road network of North America can be segmented into about 90 sub-graphs if roads longer than 200 *km* serve as thresholds for designating sub-graphs (objects farther apart wont be involved in matching constraints.) Each sub-graph is managed by a dedicated server (designated as road network server). A server manages several sub-graphs depending on the size of the road network portion it represents and the server's compute capacity.

Sub-graphs, represented by the minimum bounding rectangles (MBR), are indexed by an R-tree. With a fanout of 5, only 3 tree levels are required to manage the 90 MBRs representing the road network of North America. Given the coordinates of a moving object, the R-tree retrieves the MBR containing the object and dispatches the constraint processing to the road network server where the object is located. A constraint is processed only if all its objects are located on the same server. If the objects are spread over multiple MBRs, the constraints wont match since the distances among different MBRs is at least d_{max}. A gateway server maintains the R-tree which neither holds the detailed road network, nor does it evaluate any constraints. The gateway dispatches the processing of a constraint to one of the road network servers where all objects are located and suspends the evaluation of a constraint if the objects are spread over multiple MBRs. In this way large road networks with many moving objects can be managed. We evaluate the resulting scalability in Sec. 7.

6. MATCHING WITH MOTION PLAN

Often moving objects (trains, street cars, commuters, etc.) follow pre-determined routes. This information can be used to predictively compute location constraint matches in the future or to retrospectively determine historic matches. An example of a predictive constraint is "*When would the pairwise constraint*

$mpd(P) < d$ *be satisfied?*" and a historic constraint could be "*Has the min-sum constraint* $msd(P) < d$ *been matched in the past 20 minutes? If so, when did it match?*". In this section we briefly describe through examples how these queries can be processed with our approach.

We express the movement trajectory of an object as a vector $\langle route, velocity\rangle$, where $route$ is a sequence of nodes v_i the object visits ($route = \langle v_1, v_2, ..., \rangle$) and $velocity$ is the change in speed while moving ($velocity = \langle\langle t_1, s_1\rangle, \langle t_2, s_2\rangle, ...\rangle$.) The object moves with speed s_i in the time interval $[t_i, t_{i+1}]$ (t_i is a time stamp.) We refer to the vector, $\langle route, velocity\rangle$, as the *motion plan*. Given a motion plan and a start point on the road network, future constraint matches can be predicted, and historic matches can be computed. Notice that for a motion plan extending to the past, the time stamps in $velocity$ are smaller than the current time, indicating that the movement occurred before the time of reference. For clarity of presentation, we assume the current time (when the query is executed) is 0. We only discuss the predictive case where the motion plan has positive time stamps. The historic evaluation is analogous, except it deals with a motion plan with negative time stamps.

We first look at the computation of the *pairwise-distance* for two objects with given motion plans. Based on Fig. 4, assume that at time 0, p_1 is moving towards v_6 with speed[4] 0.5 ($\langle\langle v_6\rangle, \langle 0, 0.5\rangle\rangle$) and p_3 is moving towards v_7 with speed 0.5 ($\langle\langle v_7\rangle, \langle 0, 0.5\ \rangle\rangle$). Similar to the discussion in Sec. 3, the distance between p_1 and p_3 equals the smallest among $\overline{p_1, v_5, v_5, p_3}$, $\overline{p_1, v_5, v_7, p_3}$, $\overline{p_1, v_6, v_5, p_3}$ and $\overline{p_1, v_6, v_7, p_3}$. The length of $[p_1, v_5]$ is a function of time, t, $[p_1, v_5]_t = 4 + 0.5t$. Likewise, $[p_1, v_6]_t (= 3 - 0.5t)$, $[p_3, v_5]_t (= 2 + 0.5t)$ and $[p_3, v_7]_t (= 3 - 0.5t)$ are also functions of t. Since $\overline{v_5, v_5} (= 0)$, $\overline{v_6, v_7} (= 4)$, $\overline{v_5, v_7} (= 5)$ and $\overline{v_6, v_5} (= 7)$ are static, all four routes are functions of t, and $\overline{p_1, p_3}_t$ is the lower envelop [5] of these four functions (the solid line in Fig. 8), which is also a piecewise linear function. The prediction of the matching time for constraint $mpd(p_1, p_3) < 7$ are the time intervals [0,1] and [3,6], where the lower envelop is below the horizontal line $d = 7$. For clarity of presentation, we restrict the time range to a small value (6 in Fig. 8) such that the objects do not go beyond the current edge. When each pair is considered, this computation is extensible to solve constraints involving more than two objects. Also the movement of the objects does not have to be linear, in which case the *pairwise-distance* is the lower envelop of two non-linear functions.

We use the function MOTIONPLAN_DISTANCE(MotionPlan m_1, m_2, TimeRange T) to compute the distance between objects with motion plan m_1 and m_2 within the time range T. In our implementation, the query time range can be arbitrarily specified as long as the motion plan is defined within that range. An object can have a sequence of nodes and arbitrary speed changes as a designated motion plan, and it updates the server only if the motion plan changes.

If an object moves with constant speed, the $e-$split w.r.t. the object also moves with constant speed. If we replace the motion plan of one object with the motion plan of an $e-$split in MOTIONPLAN_DISTANCE, the dynamic *sum-distance* between the $e-$split and all moving objects can be computed and plotted as a piecewise function. In that manner, the predictive and historic min-sum constraint can also be evaluated. Again, based on

[4]Speed is expressed as unit length per unit of time.

[5]The lower envelop of function f_1,f_2,...,f_n is $\min\{f_1, f_2, ...f_n\}$.

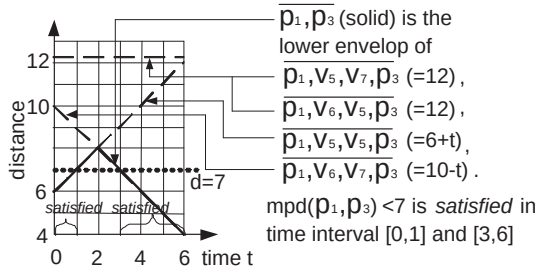

Figure 8: Pairwise Distance as a Function of Time

the example in Fig. 4, as p_3 is moving towards v_7 with speed 0.5, the $e-$split es_{p_3} (with start position $\theta_{es_{p_3}} = 6/7$) w.r.t. p_3 on $[v_5, v_6]$ is moving towards v_5 with the same speed. Therefore, $[es_{p_3}, v_6] = 1 + 0.5t$ and $[es_{p_3}, v_5] = 6 - 0.5t$. We show in Fig. 9 that $\overline{es_{p_3}, p_3} = 8$, $\overline{es_{p_3}, p_1} = 2 - t$ for $0 \le t \le 2$ and $\overline{es_{p_3}, p_1} = t - 2$ for $2 \le t \le 6$ (p_1 is moving towards v_6 with speed 0.5 and it goes beyond v_6 after time 6.) Now assuming p_2 also moves with speed 0.5 towards v_4, then $\overline{es_{p_3}, p_2} = 5 + t$ for $0 \le t \le 5$, $\overline{es_{p_3}, p_2} = 15 - t$ for $5 \le t \le 6$ (p_2 goes beyond v_4 after time 6.) The *sum-distance*, $sd(es_{p_3})$, is a piecewise function: $sd(es_{p_3}) = \overline{es_{p_3}, p_1} + \overline{es_{p_3}, p_2} + \overline{es_{p_3}, p_3} = 15$ for $0 \le t \le 2$, $sd(es_{p_3}) = 11 + 2t$ for $2 \le t \le 5$, and $sd(es_{p_3}) = 21$ for $5 \le t \le 6$, as shown in the right side of Fig. 9. The constraint $msd(p_1, p_2, p_3) < 19$ is *satisfied* in the time interval $[0, 4]$ given the MSC coincides with es_{p_3}. This computation also needs to be executed for the $e-$splits of p_1 and p_2. All the time intervals retrieved are then combined to form the final result.

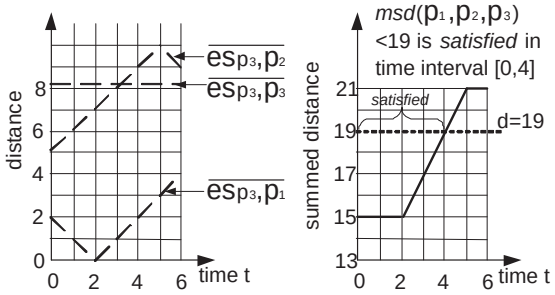

Figure 9: *Sum-distance* to $e-$split

Matches for the min-max constraint can be computed in a similar fashion. For example, with the above motion plan, with speed 0.5, es_{p_1} and es_{p_2} are both moving towards v_6 and es_{p_3} is moving towards v_5. If we denote the position of a point on the edge $[v_5, v_6]$ as the distance from v_5 to that point, the position of the $e-$splits can be described by functions of time: $es_{p_1}(t) : 4 + 0.5t$, $es_{p_2}(t) : 1 + 0.5t$, $es_{p_3}(t) : 6 - 0.5t$. They partition the edge into four intervals $[v_5, es_{p_2}]$, $[es_{p_2}, es_{p_1}]$, $[es_{p_1}, es_{p_3}]$ and $[es_{p_3}, v_6]$.

Within the time interval $[0, 2]$, the $e-$splits do not change their order on the edge $[v_5, v_6]$. Assuming x is a point on $[es_{p_2}, es_{p_1}]$, then $es_{p_2} \le x \le es_{p_1}$ (or $1 + 0.5t \le x \le 4 + 0.5t$ for $t \in [0, 2]$.) Since the positions of p_1, p_2 and p_3 are all described as functions of time, the distance from x to the objects can be described as: $\overline{x, p_1} = 4 + 0.5t - x$, $\overline{x, p_2} = 11 + 0.5t - x$ and

$\overline{x, p_3} = 2 + 0.5t + x$, each is a function with two variables (i.e., represents a plane in 3D space). It can be verified with linear algebra that the maximum among them is $\overline{x, p_2}$ and the minimum of $\overline{x, p_2}$ is 7, which is achieved when $x = 4 + 0.5t$. So the *min-max-distance* equals 7 in the time interval $[0, 2]$, if the MMC ($= 4 + 0.5t$) is in $[es_{p_2}, es_{p_1}]$.

Likewise, the *min-max-distance*, given the MMC is in $[v_5, es_{p_2}]$, $[es_{p_1}, es_{p_3}]$, $[es_{p_3}, v_6]$, is also functions of time, and the lower envelop of the *min-max-distance* on all intervals is the *min-max-distance* on that edge. As time extends, the $e-$splits may change their order on the edge, but the computation of the *min-max-distance* follows in the same fashion and the constraint match can be determined by combining results from all time ranges (each range is defined as the time period within which the order of $e-$splits does not change).

If we abstract the motion plan as object positions, then our graph partitioning scheme can be applied to prune the computation of predictive and historic constraints. For instance, for the parts of the motion plans that are inside a partition, the *max-pairwise-distance* of the objects is bounded from above by the diameter of the partition involving part of the plan in the defined time range.

7. EXPERIMENTS

For the experimental evaluation, we used data sets synthesized with the IBM City Simulator and Location Transponder released through IBM alphaWorks. We also extracted data sets from Google Ride Finder [2] and Cabspotting [1] (i.e., trace data that provides position updates of taxi cabs in major U.S. cities.) The street layout of these cities serves as road networks for our experiments. We evaluate a stream of position updates against location constraints stored in our system. The typical experiment setup for one city consists of a road network with 25,000 nodes and 100,000 edges, with the average edge length ranging from 0.2 to 3 kilometers. The location constraints are generated beforehand and each constraint is associated with n ([2∼5]) moving objects.

We track 50,000 objects moving on the road network with an average velocity of 10*m* per unit of time. In our experiments we varied the alerting distance from a few hundred meters to about 100*km*. For simplicity, the evaluation is carried out in rounds. In each round all objects move for one unit of time. Location updates trigger the evaluation of constraints. The results presented are the average over 40,000 rounds. The average matching time is the time to evaluate all constraints in one round. All experiments were conducted on a Pentium 4 with 2GB RAM running Ubuntu 8.04.

Efficiency of Distance Computation: In this experiment, we evaluate the efficiency of our algorithms based on the different pruning strategies introduced in Sec. 3 (e.g., reuse of the previously computed pairwise distances for *max-pairwise-distance* computation, e-split pruning based on Lemma 1 for *min-sum-distance* computation, and edge pruning based on Theorem 1 for *min-max-distance* computation). Table 1 compares the number of pairwise distance computations, the number of e-splits searched, and the number of edges searched for a workload with 20,000 constraints. To assess the effect of each optimization in isolation, the road network is not partitioned. We observe that the pruning strategies reduce the search space by about 39% for *max-pairwise-distance*, 75% for *min-sum-distance*, and 93% for *min-max-distance*. With road network partitioning, further reduction (at least 10% depending on the object movement patterns) is possible as shown below.

Table 1: Search Space Pruning for Distance Computations

type	w/o pruning	w/ pruning
# **pairwise distances (pairwise)**	270,202	165,243
# e-**split searched (min-sum)**	198,019,031	47,053,564
# **edges searched (min-max)**	30,040,937	2,821,920

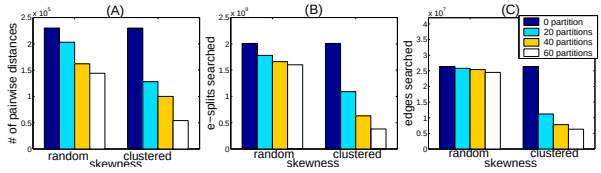

Figure 10: Partition-based Constraint Matching

Effect of Partitioning: This experiment evaluates the pruning effect of graph partitioning. Three partition schemes are generated with 20, 40 and 60 unit partitions, respectively. The cuts are selected such that all the partitions are roughly of the same size [6]. To demonstrate the pruning effect of graph partitioning alone, we applied none of the other pruning strategies from Sec. 3. In addition to the uniform distribution of the objects across the whole network, we also identify and utilize a skewed (clustered) data set where a high density of objects are clustered within a few designated partitions (e.g., hot spots in the city) in a specific period of time (e.g., during rush hours). The skewed data represents clustered movement of objects, a pattern often found in practice. First, with the uniform distribution, we observe a reduction of the search space as the number of unit partition increases. Fig. 10 shows that with 60 partitions, the search space reduction is around 30% for pairwise constraints (A) and 10% for min-sum (B) and min-max (C) constraints. This shows that a higher partition granularity increases the precision of partition-based evaluation and prunes more constraints. Fig. 10 also shows that the efficiency of pruning is influenced by the movement pattern of objects. Pruning is more efficient for clustered constellations of objects. For all partition schemes and all kinds of constraints, a significant reduction in the search space is observed when objects are clustered. For example, if the objects are clustered, pruning exhibits the best results. The search space is reduced to 15-20% of the original size for the case of 60 partitions.

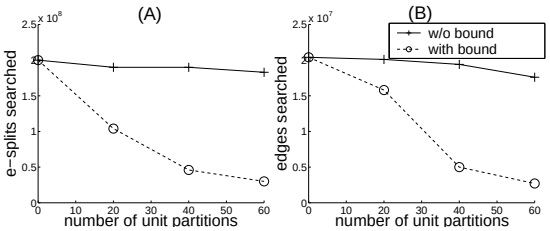

Figure 11: Effect of Pruning with Bound

[6]Partition size is defined as the number of edges in the partition.

Effect of Pruning with Bounds: This experiment demonstrates the effect of the six pruning rules based on lower and upper bounds from Sec. 5. Due to space limitation, Fig. 11 only shows the number of e-splits and edges searched for the computation of min-sum constraints (A) and min-max constraints (B) with and without bound pruning. We observe that the search space is reduced by more than 70% if bound-based pruning is applied for a partition scheme with over 40 unit partitions. This is a direct consequence of partition bounds, since the matching result of many constraints can be directly inferred from the partition information as partitioning is getting more fine-grained. We also find that very small or very large alerting distances have a much better pruning effect.

Effect of Network Distance Computation: An important factor that affects the performance of the aforementioned algorithms is the network distance computation. In our implementation, we use the *network expansion* algorithm [11] to compute the distances between nodes. The *network expansion* algorithm has a time complexity that theoretically depends on how widespread objects are apart. Since we modify the algorithm to stop expanding search if the path length exceeds the alerting distance, the time for the algorithm in our implementation is a constant depending on the alerting distance.

Predictive Constraint Evaluation: In this experiment, we measure the matching time for evaluating predictive constraints with various trajectory length (defined as the number of edges in a trajectory) and partition schemes. We use experimental data obtained from evaluating min-sum and min-max constraints to demonstrate the results. Fig. 12(A) shows that the matching time is proportional to the length of the trajectory. The more edges are processed, the more expensive the evaluation and the higher the matching time. However, with a fixed trajectory (40 edges), the matching time drops (to 20% for 60 partitions) as the number of partitions increases. This result is due to the pruning effect finer grained partitioning achieves (see Fig. 12(B)).

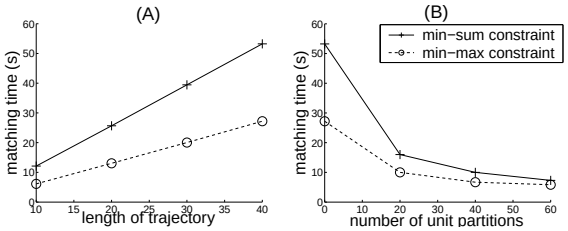

Figure 12: Predictive Constraint Processing

Scalability to Large Road Networks: In this experiment, we initially store the maps of 50 US cities in one server and subject the server to different constraint loads (10k, 15k, 20k). We then distribute the city maps across 5 and 10 road network servers, respectively. The results are shown in Fig. 13. When using multiple servers, the map and the constraints are evenly distributed among the servers. From the figure, we see that distributed processing greatly improves scalability. The average matching time is reduced close to 1/5th (for the 5 server case) and to 1/10th (for 10 server case) of the original load (see Fig. 13(A)). However, distributed processing requires the location update messages to be forwarded to the road network severs via the gateway server. This leads to a 30% increase in network bandwidth use. Not every location update message is dispatched by the gateway server due to pruning conducted by the gateway. (Fig. 13(B)). Some messages are discarded since the gateway knows that the objects are on different road network servers and therefore can not match constraints. The number of messages for the centralized server experiment (1 server case) includes only location updates from moving objects. There are no inter-server messages because there is no gateway server forwarding messages.

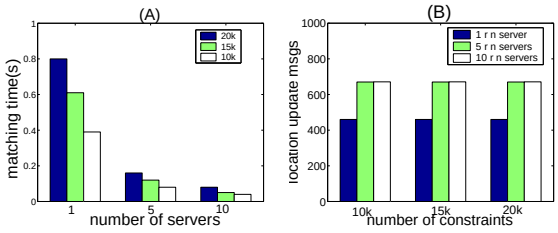

Figure 13: Scalability Results

Performance With All Optimizations Enabled: The previous controlled experiments were conducted to study the various aspects of our approach in isolation. We now conduct an end-to-end evaluation on real data sets and road networks that combines all techniques to achieve maximal efficiency. The results for different constraint types are shown in Fig. 14. We observe that the processing time is reduced as the number of partitions increases. For clarity of presentation, the performance of the baseline approach, where no optimizations are used, is not shown since it is twenty times larger than the partition-based approach (with 20 partitions). For instance, the constraint load with 20K min-sum constraints that requires 20 minutes to process with the baseline approach, is processed in less than 10 seconds using our algorithm with all optimizations combined.

Also, we find that the movement speed does not significantly effect the experimental results. For very small and very large alerting distances, processing is sped up, due to improved selectivity. The processing time is linear in the number of objects, n, involved in a constraint. We omit these figures due to space limitations.

Memory Use: Memory consumption is an important performance concern. There are two factors that contribute to memory use in our approach: (1) The road network representation and its partitioning, and (2) the constraints and moving objects. The road network graph is organized as adjacency lists. Its partitions are organized in an array. Both components are typically fixed, given a specific road network. In all our experiments, the memory used for these components never exceeded 10 MB. The memory use for constraints and objects is proportional to their quantity. It ranges from 2 MB to 25 MB when constraints vary from a few thousand to one-hundred thousand in our experiments. The *network expansion* algorithm does not require additional memory, because the expansion is restricted by the alerting distance which is a constant. In our implementation when road network partitioning was applied, the total memory consumption never exceeded 70 MB. Memory consumption is thus not a limiting factor for our approach.

8. RELATED WORK

Spatio-temporal queries in Euclidean space include k closest pairs [4], k nearest neighbor (kNN) [9], and continuous range query [14] *et cetera*. In our own prior work we studied the

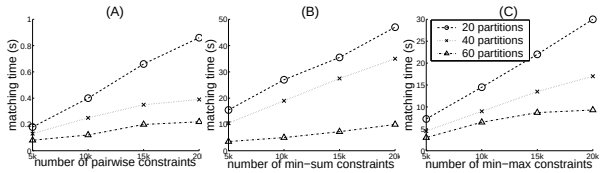

Figure 14: All Optimizations Enabled

processing of proximity relations among moving objects in Euclidean space under precisely and imprecisely given location position data, respectively [17, 16]. All these approaches apply exclusively to query processing support in Euclidean space. The query processing model studied in this paper is based on evaluating, correlating, and tracking queries over a road network space. None of the above listed approaches extends directly to such a model.

For road network query processing, Cho *et al.* and Kolahdouzan *et al.* proposed the *UNICONS* and VN^3 algorithms for evaluating kNN and continuous kNN queries (CNN) in road networks [3, 8]. The approach incorporates the use of pre-computed nearest neighbor lists based on Dijkstra's algorithm. Recent work reduces the storage cost and processing time for kNN query and distance computation by making use of an encoding of precomputed distances between nodes [12, 6]. Another approach uses network distance computations based on distance signatures [7]. The signatures discretize distances between objects and network nodes into categories and then compress the categories. The kNN and CNN queries in the above approaches are fundamentally different from the location constraints we advocate in this paper. kNN and CNN are static queries that retrieve the nearest neighbor(s) to a static query point (a fixed point or a fixed path on the road network.) However, our location constraints track various different proximity relations among a set of moving objects, without a fixed query point. Location constraints resemble standing or continuous queries that once registered with the query processor, continuously produce output triggered by location updates. Our approach evaluates many such location constraints concurrently. kNN monitoring for moving objects is studied in [10]. The work assumes that both the object of interest and the query are moving on the road network. Despite involving a moving query, a kNN is different from the constraints we study, because the query in kNN is given, but the MSC and MMC in our work are the results to be computed. Also, the kNN work does not consider query pruning, which applies to optimize the continuous processing of large amounts of concurrent queries, a central concern in our work.

Papadias *et al.* [11] studied nearest neighbor, range search, closest pairs query and e-Distance joins in road networks. They integrate network and Euclidean information to prune the search space. They develop two frameworks, the *Euclidean restriction* and the *network expansion* to process queries. Their queries differ fundamentally from the queries and constraints presented in this paper. The nearest neighbor and range search are static queries that do not support the matching of location constraints central to our application context. The closest pairs query and e-Distance joins study a proximity relation between two sets of objects, where the proximity relation involves two objects, one in each set, while our location constraints can involve an arbitrary number of moving objects and is not based on an a priori division of objects into two sets.

The approach introduced in this paper resembles publish/subscribe [5]. The difference is the expressive constraint language and processing in road network space.

9. CONCLUSIONS

In this paper we introduced query support for evaluating proximity relations that can induce location constraints. Proximity relations and location constraints specify whether a given set of moving objects are in a specific constellation in road network space. The algorithms we developed in this paper are designed to concurrently monitor and evaluate thousands of location constraints under changing location position information. Applications ranging from location-based services to multi-player online gaming can benefit from this query support.

We developed query processing algorithms for computing the proximity relations and for tracking the value of the relations to determine, if the associated location constraints deviate from a specified alerting distance. We developed several pruning techniques that are based on road network partitioning and on lower and upper bound calculations for the constraint evaluation. Based on given movement trajectories of objects, our approach can be used to predicatively determine future constraint matches and assess historic constraint matches. Experimental evaluations based on real data sets showed that, with all optimizations combined, the total processing overhead is reduced by more than 95%. Our approach is therefore directly applicable to detecting proximity relations in real-time for moving objects representing mobile entities in major cosmopolitan areas or in multi-player online gaming.

10. REFERENCES

[1] Cabspotting. *http://cabspotting.org*.
[2] Google Ride Finder. *http://labs.google.com/ridefinder*.
[3] H.-J. Cho and C.-W. Chung. An Efficient and Scalable Approach to CNN Queries in a Road Network. In *VLDB*, 2005.
[4] A. Corral et al. Closest pair queries in spatial databases. In *SIGMOD*, 2005.
[5] F. Fabret et al. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, 2001.
[6] S. Gupta, S. Kopparty, and C. Ravishankar. Roads, Codes, and Spatiotemporal Queries. In *PODS*, 2004.
[7] H. Hu, D. L. Lee, and V. C. S. Lee. Distance Indexing on Road Networks. In *VLDB*, 2006.
[8] M. R. Kolahdouzan and C. Shahahi. Continuous K Nearest Neighbor Queries in Spatial Network Database. In *STDBM*, 2004.
[9] K. Mouratidis et al. Conceptuual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *SIGMOD*, 2005.
[10] K. Mouratidis et al. Continuous Nearest Neighbor Monitoring in Road Networks. In *VLDB*, 2006.
[11] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB*, 2003.
[12] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
[13] D. B. West. Introduction to Graph Theory. Prentice Hall, 1996.
[14] K.-L. Wu et al. Efficient Processing of Continual Range Queries for Location-Aware Mobile Services. *ISF'05*.
[15] Z. Xu. Efficient Location Constraint Processing For Location-aware Computing. Ph.D. Thesis, Univ. of Toronto, 2009.
[16] Z. Xu and H.-A. Jacobsen. Adaptive Location Constraint Processing. In *SIGMOD*, 2007.
[17] Z. Xu and H.-A. Jacobsen. Evaluating Proximity Relations Under Uncertainty. In *ICDE*, 2007.