Mining City-Wide Encounters in Real-Time

Anthony Quattrone, Lars Kulik and Egemen Tanin
Department of Computing and Information Systems
The University of Melbourne
quattronea, lkulik, etanin@unimelb.edu.au

ABSTRACT

Recent advancements in data mining coupled with the ubiquity of mobile devices has led to the possibility of mining for events in real-time. We introduce the problem of mining for an individual's encounters. As people travel, they may have encounters with one another. We are interested in detecting the encounters of traveling individuals at the exact moment in which each of them occur. A simple solution is to use a nearest neighbor search to return potential encounters, this results in slow query response times. To mine for encounters in real-time, we introduce a new algorithm that is efficient in capturing encounters by exploiting the observation that just the neighbors in a defined proximity needs to be maintained. Our evaluation demonstrates that our proposed method mines for encounters for millions of individuals in a city area within milliseconds.

CCS Concepts

•Information systems → Spatial-temporal systems; Data mining;

Keywords

Spatial Databases; Data Mining

1. BACKGROUND

The occurrence of encounters between two or more people occurs when they are within proximity to one another. Many individuals keep their smartphones present with them at all times or at least within the same room [3]. This provides an approximate location of where the individual is located. With spatial data captured from mobile devices, it is possible to mine for potential encounters by checking for individuals that are in proximity to each other via performing queries on the spatial data. Many devices now send continuous positioning estimates, allowing for the detection of encounters in real-time.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA
© 2016 ACM. ISBN 978-1-4503-4589-7/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2996913.2996995

Proximity is a constraint required for individuals to be having an encounter. We exploit this property by maintaining an index that stores points that are close to each other in a given area. Our novel approach is based on using a dynamic grid that maintains and stores the cells where individuals have not moved from for a specified time period. An efficient scan can be executed to map people's locations to the grid cells. People that are in the same grid cell are identified as potential candidates for encounters. To evaluate our approach, we developed a simulator that creates an environment of people traveling in an inner city area, where they may have an encounter with each other.

To the best of our knowledge, this is the first work that focuses specifically on encounters. We demonstrate our novel algorithm to mine for encounters in real-time. Our algorithm can find potential encounters within milliseconds for two million people in Melbourne's inner city. It is significantly more suitable for real-time mining than conventional nearest-neighbor search techniques, which run orders of magnitude slower.

Traditionally, location data is indexed with tree-based data structures. The TPR tree is an extension of an R-Tree designed to handle queries in the time dimension. It takes into account current and future positions of moving object data [14]. For a large set of neighbor investigations, even a minor change in the size of the area under investigation will lead to large execution times, making this approach inefficient for our problem. To overcome the high cost in node splitting exhibited in TPR trees, the use of dynamic B trees [7, 8] has been explored. The B^x tree maintains a directory in each node containing the id, velocity, mapping and the last update time of an object [10]. The B^x tree requires an object move in fixed velocity and cannot be directly applied to our problem.

Certain Quadtrees [4, 5] can be used to index moving objects as demonstrated in [11]. By using loose quadcells, an object can move and not need to be reinserted provided the new location is still within the quadcell. There are no defined bounds where a person may move in a city area. These approaches also do not consider any tolerance constraints nor the fact that we are interested in ANN query types rather than k-NN.

Our problem can be represented as the All Nearest Neighbor (ANN) problem that states for all N points, find the k nearest neighbors. Efficient solutions for the ANN problem have been proposed with the basic premise being to exploit redundancy [2, 12, 13]. Initial ANN approaches assume that the points are static, which means that the data structures in most cases need to be recalculated when the positions of the points change. There has not been much attention focused on continuous ANN. One approach recently proposed in [1] presents an algorithm that uses cell tower radius as a proxy to reduce the search space. This is still not an acceptable solution to our problem due to the overlapping of the coverage areas of cell towers and the fact that these areas are relatively large in comparison to a central area in the city.

Our algorithm makes use of constraints in both the time and space domains to reduce computation. As a result, there are far less points that need to be stored in the index and the number of combinations that need to be processed.

2. MINING FOR ENCOUNTERS

A potential encounter can occur when people are in close proximity. It is assumed people are within proximity when the distance between them is within ϵ distance. We define the constraint nearest neighbor (c-NN) query to return objects in proximity at a given location.

In order to search for people that are within proximity to one another quickly, a grid structure can be constructed and used as a spatial index. In a grid, the distance between two objects can be approximated as the distance between the two grid cells, where the objects are located. Constructing a grid removes the need to calculate the exact distances between all possible pairs of points. In other words, people in proximity can be found by considering all the people that are in the same cells or surrounding cells if the grid is constructed and interpreted carefully.

We propose TimeGrid, a spatial index that takes into account time as well as space. Since in many cases it may be only of interest to capture encounters that last a certain amount of time, the minimum time an encounter has to last before it is reported is defined as τ. The underlying data structure is required to maintain a list of people that fall within the grid cell for τ seconds. Given the cells are set to have a side of size $\epsilon/\sqrt{2}$, a person that is in the same grid cell for τ seconds with another person indicates a potential encounter.

Let P be a finite set of people and L be the set of locations where a person can be located as well as an encounter may occur, a person $p_a \in P$ at a location $l_m \in L$ is returned by the function $\text{Loc}_t(p_a)$ defined as follows:

$$\text{Loc}_t(p_a) = l_m \text{ where } t \in T,\ p_a \in P,\ l_m \in L$$

Each person $p_a \in P$ is positioned at a location $l_i \in L$, all locations are within $\mathbb{R}^2$. The space itself can be partitioned into a grid which can then in turn be used to index each person in the set P. We define a grid overlaid with each cell to be of size $\delta = \epsilon/\sqrt{2}$ in each dimension. Thus, locations within a grid cell would be within a maximum of ϵ distance from one another. The grid over the entire space is defined as G with each cell having a side of δ. Each person in P is indexed to determine which grid cell they reside in and represented in a set.

The function $\text{CellID}_t(p_a)$ where $p_a \in P$ returns the index of a cell a person is in, defined as

$$\text{CellID}_t(p_a) = \left(\left\lfloor \frac{x}{\delta} \right\rfloor, \left\lfloor \frac{y}{\delta} \right\rfloor \right),\ (x, y) = \text{Loc}_t(p_a)$$

The set PG represents all the cell locations people are located in at a given time. It is assumed that all people in set P can then be mapped to a grid cell in the set PG at time $t \in T$ as follows:

$$PG_t = \{\text{CellID}_t(p) \mid p \in P\}$$

To efficiently check inside cells, we can create an index inside each cell for some really dense city streets. For the sake of simplicity, we divide each cell into four quadrants in this paper as an example. Since our main approach only requires to check active cells. Subdividing a cell into quadrants allows for efficient checking of surrounding active cells that are part of the space.

Each quadrant within the cell has a side of $\epsilon/2\sqrt{2}$. In a manner the same as a QuadTree, quadrant assignment is labeled based on the cardinal directions (NE,NW,SE,SW). For example, in the case where a person is in the top left corner of the cell, this would be labeled as NE.

We use virtual boundaries made up of quadrant cells to ensure all people within ϵ distance are considered. Consider a person $p_a \in P$ in the NE corner of one cell, a person p_b in the SW corner of the adjacent top right cell would satisfy the property $\text{Dist}(\text{Loc}(p_a), \text{Loc}(p_b)) \leq \epsilon$.

The function $\text{CloseQuadsP}_t(c)$ where $c \in PG_t$ returns the set of people in surrounding cells by checking quadrants that are within ϵ distance of one another. Hence, the people in proximity to the person $p_a \in P$ can be determined by finding all the people within the same cell as $p_a \in P$ and within surrounding quadrants within ϵ distance.

Initially each person is assigned to a grid cell. At each interval of τ a scan is performed to determine if a person is still in the same grid cell or has changed cell locations. With τ being set, only the cells where one or more individuals have stayed longer than a certain amount of time needs to be checked. Such cells are marked as active cells. These cells can be determined by taking the intersection the cells identified at different times, this results in the TimeGrid TG as follows:

$$TG = PG_{tnow} \cap PG_{tnow-\tau}$$

Proximity results are in the TimeGrid output TG. A cell in the proximity result must satisfy one of the two following conditions. First, the cell contains at least one person. Second, a surrounding cell contains at least one or more people. The spatial index is refreshed at increments of τ seconds.

To mine for potential encounters, active cells are required to be checked. People within the active cells are within proximity to one another. It is also possible that a person in an active cell is in proximity to people in another active adjacent cell. We can determine who is inside the cell. The function $\text{CellP}_t(c)$ where $c \in TG$ returns the list of people inside a cell is as follows:

$$\text{CellP}_t(c) = \{p \mid \text{CellID}_t(p) \equiv c,\ p \in P\}$$

Suppose a person $p_a \in P$ is indexed in a cell $c \in TG$ and no other person can be found in the adjacent cells, then only the people inside c would be in proximity to p_a. Thus, the following relationship can be satisfied as follows:

$$\text{c-NN}_t(p_a) = \{p \in \text{CellP}_t(c) \mid c = \text{CellID}_t(p_a)\}$$

Consider $\text{CellP}_t(c)$ to be the set of people inside an active cell and $\text{CloseQuadsP}_t(c)$ where $c = \text{CellID}_t(p_a)$ and $p_b \in \text{CloseQuadsP}_t(c)$ to be the set of people in surrounding quadrants that are within ϵ distance. With these defined, $c\text{-NN}_t(p_a)$ where $p_a \in P$ in a cell $c \in TG$ can be defined using TimeGrid as follows:

$$\text{c-NN}_t(p_a) = \left\{ \begin{aligned} &\text{CellP}_t(c) \ \cup \ p_b \mid \\ &\qquad\qquad c = \text{CellID}(p_a), \\ &\qquad\qquad p_b \in \text{CloseQuadsP}_t(c), \\ &\qquad\qquad \text{Dist}(\text{Loc}(p_a), \text{Loc}(p_b)) \leq \epsilon \end{aligned} \right\}$$

The distance between people found in quadrants needs to be verified to ensure they are within ϵ distance as points can be on the outer edge of a quadrant that slightly exceed ϵ. The set of encounters E_t can be derived using TimeGrid is as follows:

$$E_t = \{\mathcal{P}(\text{c-NN}_t(p)) \mid p \in P\}$$

The powerset ($\mathcal{P}$) of all the people in proximity is derived resulting in every possible combination of encounters.

3. EXPERIMENTAL EVALUATION

This section presents our evaluation comparing TimeGrid with different adapted approaches that make use of spatial indexes for the purpose of detecting encounters. Positioning and movement data of people was generated by a simulator and passed to the mining algorithms in real-time. The simulator uses real geospatial data provided by Open Street Map (OSM). People are initially distributed in a uniform manner within the tested area when the simulation begins. Parameters tested are summarized in Table 1. The only inputs available to the mining algorithms were the locations of people at a given time. This is typical of what would be available by a service collecting location data in real-time. The city we selected to simulate encounters was Melbourne, Australia. The population density is high in the inner city areas and is low in surrounding areas which is similar to many big cities around the world.

TimeGrid demonstrated that it significantly outperforms competing approaches in every use case we tested. It can scan for encounters in under a second even in situations where millions of people were considered and every possible encounter location was tested. We first compare TimeGrid against certain nearest neighbor algorithms that use static indexes [6, 9] as can be seen in Figure 1.

Mining for encounters can be done using indexing of moving objects. We found that velocity based approaches did not work well for our problem at all. For example, when indexing moving objects with TPR tree, we found that scan duration grows substantially as the time period of the simulation increases. Figure 3 plots the time elapsed since the TPR tree was built with the scan duration. It can be seen that the scan duration grows significantly as time increases. The trend resulted from the fact that when a person moves, the amount of places the person could possibly visit increases as time elapses. TPR trees tend to perform better for queries where the the common velocity vector is fixed in a limited direction. For example, which airplanes are going to land on the runway in the next ten minutes. We also attempted a B^x tree, however we found that this approach cannot be applied to our problem due to the requirement of fixed velocity vectors.

A continuous approach to answering k-NN queries for moving objects proposed in [15] was also tested. In attempting to apply this method, the evaluation time took much longer than any other method we used mainly due to cases where the closest neighbor is far away. When looking for

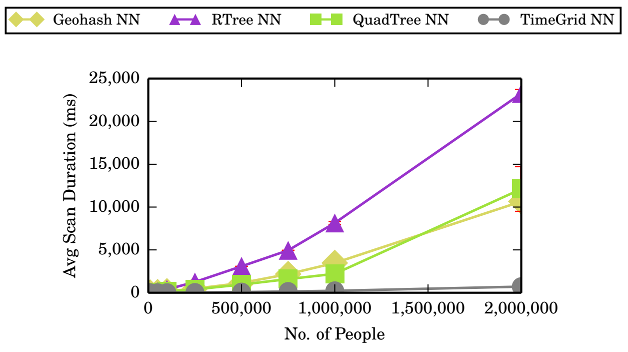


Figure 1: Avg Scan Duration by Number of People

A: $30km^2$, τ: 1s, ϵ: 5m, ν: $5km/h$

Parameter	Values Tested
# of People	1k, 10k, 50k, 100k, 250k, 500k, 750k, 1M, 2M
ϵ	$5m$, $10m$, $15m$
τ	$1s$, $5s$, $10s$, $30s$
Speed ν	$0km/h$, $5km/h$, $10km/h$, $30km/h$, $60km/h$
Area A	$30km^2$, $100km^2$, $200km^2$

Table 1: Parameters Tested

k neighbors, grid cells would incrementally expand until it finds at least one neighbor. It would take significant time to find even just one neighbor in cases where the closest neighbor is far away. In our experiments, this situation occurred many times. In one preliminary test, when considering just a thousand people with 50 fixed encounter locations, this approach took an average of 14 seconds to find encounters. As more people and encounter locations were considered, the approach could not scale.

Increasing the size of the search space has the effect of lowering how densely packed people are within the tested area. It can be clearly seen in Figure 2a that increasing the search space improves the scan performance as people will be more sparsely distributed.

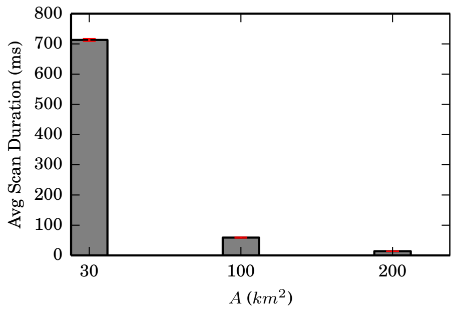
Varying the speed did not appear to make a major difference in performance as seen in Figure 2b. While traveling from one grid cell to another quicker may reduce how many active cells are indexed, there is a large number of people in the city that do not move and stay in the same position for long periods of time.

In theory, increasing the size of ϵ causes the TimeGrid to take longer to scan for encounters, this is due to having to check for more combinations in the powerset. This is evident in Figure 2c. Larger values of ϵ would result in people being too far away to have an encounter.

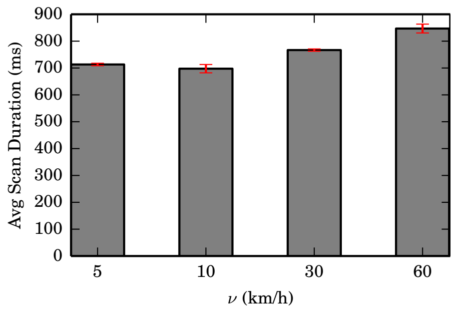
Increasing the interval τ reduces the amount of times the index needs to be updated, however possible encounters could be missed. Changing how often the index was updated had no real noticeable trend for affecting the scan time. This is shown in Figure 2d.

Initially, people are placed at a random location and move at the assigned speed of $\nu km/h$. Our aim was to realistically simulate the movements in a city environment. We also tested an extreme case of 2 million people not moving with the parameters set as $\nu = 0km/h$, $A = 30km^2$, $\tau = 1s$ and $\epsilon = 5m$. TimeGrid ran in just over a second on average even though movements over time were not being exploited. Thus, TimeGrid still performs well whether people are moving or not.

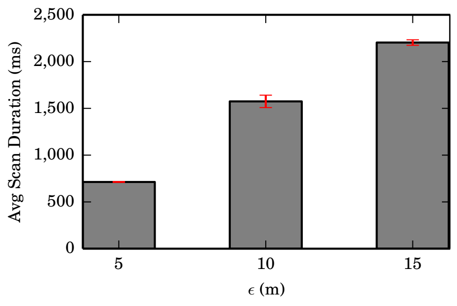
Our evaluation demonstrated how TimeGrid is better suited for mining encounters. While other techniques may miss quick encounters, our approach would capture them. We also gave insights into why current velocity based techniques designed to handle moving objects are not suitable for our application domain. Overall, the TimeGrid algorithm performed orders of magnitude better than conventional techniques across all the use cases we tested. Samples were captured fast enough to detect encounters in real-time, even if they last for just a second. We demonstrated that our approach is capable of mining encounters in real-time for a city of two million people.



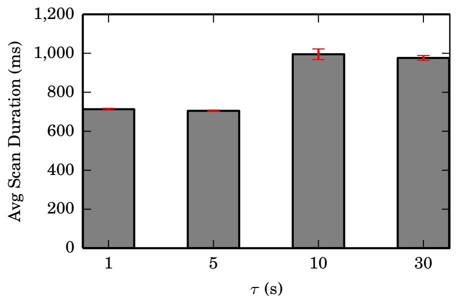
(a) A - People: 2M, τ: 1s, ϵ: 5m, ν: $5km/h$



(b) ν - People: 2M, τ: 1s, ϵ: 5m, A: $30km^2$



(c) ϵ - People: 2M, τ: 1s, A: $30km^2$, ν: $5km/h$



(d) τ - People: 2M, ϵ: 5m, A: $30km^2$, ν: $5km/h$

Figure 2: Varying Parameters of the TimeGrid Algorithm

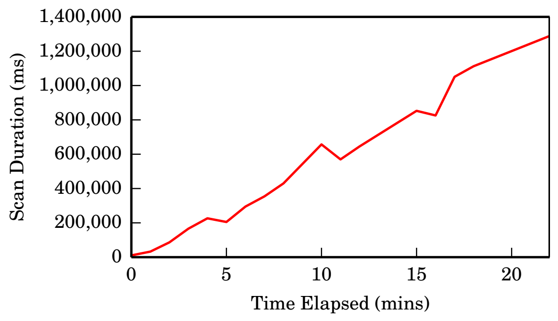


Figure 3: TPR Tree Performance as Time Increases

4. REFERENCES

[1] G. Chatzimilioudis, D. Zeinalipour-Yazti, W.-C. Lee, and M. D. Dikaiakos. Continuous all k-nearest-neighbor querying in smartphone networks. In *MDM 2012*.

[2] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *FOCS 1983*.

[3] A. K. Dey, K. Wac, D. Ferreira, K. Tassini, J.-H. Hong, and J. Ramos. Getting closer: An empirical investigation of the proximity of user to their smart phones. In *Ubicomp 2011*.

[4] A. Frank. Problems of realizing LIS: storage methods for space related data: the fieldtree. In *Institute for Geodesy and Photogrammetry, Technical Report 71, 1983*.

[5] A. U. Frank and R. Barrera. The fieldtree: a data structure for geographic information systems. In *SSD 1990*.

[6] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *SSD 1995*.

[7] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient B^+-tree based indexing of moving objects. In *VLDB 2004*.

[8] C. S. Jensen, D. Tielsytye, and N. Tradilauskas. Robust B^+-tree-based indexing of moving objects. In *MDM 2006*.

[9] J. Kuan and P. Lewis. Fast k nearest neighbour search for R-tree family. In *SIGMOD 1997*.

[10] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *SIGMOD 1986*.

[11] H. Samet, J. Sankaranarayanan, and M. Auerbach. Indexing methods for moving object databases: Games and other applications. In *SIGMOD 2013*.

[12] J. Sankaranarayanan, H. Samet, and A. Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. In *Computers and Graphics 31(2), 2007*.

[13] P. M. Vaidya. An O(n logn) algorithm for the all-nearest-neighbors problem. In *Discrete & Computational Geometry 4(2), 1989*.

[14] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD 2000*.

[15] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE 2005*.