Introducing Diversion Graph for Real-Time Spatial Data Analysis with Location Based Social Networks

Sameera Kannangara
School of Computing and Information Systems, The University of Melbourne, Australia
kannangarad@student.unimelb.edu.au

Hairuo Xie
School of Computing and Information Systems, The University of Melbourne, Australia
xieh@unimelb.edu.au

Egemen Tanin
School of Computing and Information Systems, The University of Melbourne, Australia
etanin@unimelb.edu.au

Aaron Harwood
School of Computing and Information Systems, The University of Melbourne, Australia
aharwood@unimelb.edu.au

Shanika Karunasekera
School of Computing and Information Systems, The University of Melbourne, Australia
karus@unimelb.edu.au

Abstract

Neighbourhood graphs are useful for inferring the travel network between locations posted in the Location Based Social Networks (LBSNs). Existing neighbourhood graphs, such as the Stepping Stone Graph lack the ability to process a high volume of LBSN data in real time. We propose a neighbourhood graph named Diversion Graph, which uses an efficient edge filtering method from the Delaunay triangulation mechanism for fast processing of LBSN data. This mechanism enables Diversion Graph to achieve a similar accuracy level as Stepping Stone Graph for inferring travel networks, but with a reduction of the execution time of over 90%. Using LBSN data collected from Twitter and Flickr, we show that Diversion Graph is suitable for travel network processing in real time.

2012 ACM Subject Classification Information systems → Geographic information systems

Keywords and phrases moving objects, shortest path, graphs

Digital Object Identifier 10.4230/LIPIcs.GIScience.2021.I.7

Funding This research is funded in part by the Defence Science and Technology Group, Edinburgh, South Australia, under contract MyIP:6104.

1 Introduction

Location Based Social Networks (LBSNs) contain a large volume of location information posted by the users. The location data collected from LBSN can be further processed to understand various aspects of users' lives [19, 20]. LBSN data can be processed to infer the travel network between the posted locations [8]. Inferring a travel network is to find a set of edges between the posted locations or a subset of the locations so that a path can be found between any pair of the locations in the network. Processing LBSN data for such purposes is difficult due to the scale of the data to be processed. We are interested in efficient methods for inferring travel networks with LBSN data.

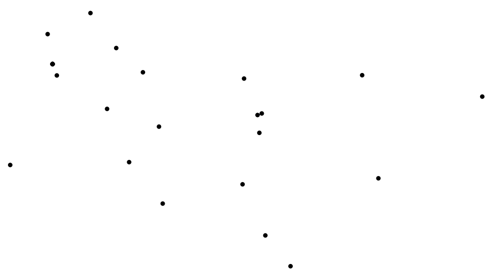
Location data collected from LBSNs is usually in the form of GPS points. Distance-based connected neighbourhood graphs have been used for inferring the relationship between a set of distinct GPS points [5]. Neighbourhood graphs are also called proximity graphs, where edges between the points are built based on certain spatial relationship between the points. Delaunay Triangulation (DT) is a well-known distance-based connected neighbourhood graph. Gabriel Graph (GG) [7], Relative Neighbourhood Graph (RNG) [17] and Urquhart Graph (UG) [18] are extended from DT for movement network analysis. For example, Figure 1 (a) represents locations collected from Twitter relating to a state election. Figures 1 (c,h,i) represent GG, RNG and UG skeletons, which are the geometric realization of neighbourhood graphs and show the geometric shape of the point set.

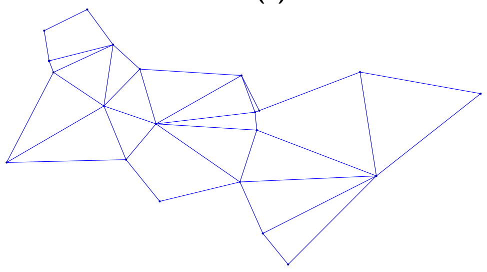
Unlike the aforementioned graphs, there is a type of graphs called variable graphs, which can generate a spectrum of possible skeletons based on different values of given parameters. Therefore, the variable graphs are making them more versatile than other types of graphs. In the rest of the paper, we specify the parameters used by variable graphs in the name of the graphs. The Shortest Path Graph ($SPG(t)$) [6] and the Stepping Stone Graph ($SSG(d)$) [8] are two commonly used variable graphs, built on the idea that the shortest path through the inferred edges can be aligned with the shortest path through the imprecise region represented by the point set. While $SPG(t)$ considers the shortest path over all points when inferring edges, $SSG(d)$ only considers the shortest path that goes through points within the relative neighbourhood between two points. Due to this difference, the travel networks created with $SSG(d)$ are more similar to real world travel networks. Both $SPG(t)$ and $SSG(d)$ can generate various graph skeletons based on a single parameter. Figure 1 (b,d,f,h) represent different $SSG(d)$ skeletons of the same point set based on different parameter settings.

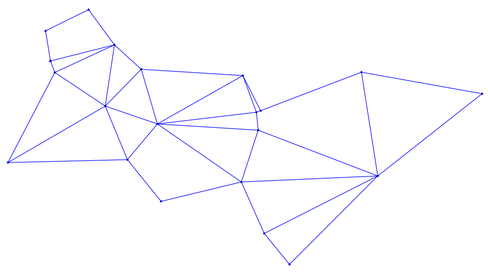
Although existing neighbourhood graphs can process LBSN data with a few hundred locations, they are not suitable for large datasets due to the long running times. With the widespread use of GPS-enabled mobile phones, the size of LBSN datasets tends to be significantly large. Therefore we need to investigate efficient methods to infer travel networks based on the location data collected from LBSNs.

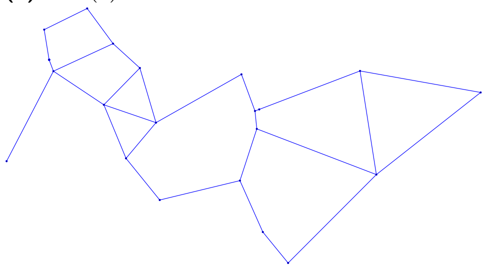
In this paper, we propose a new type of variable graph, which we refer to as the Diversion Graph ($DG(d)$). The skeletons inferred by $DG(d)$ are likely to be close to human perception of the corresponding point set. We show in our experiments that $DG(d)$ is easier and faster to build than $SSG(d)$, and gives similar results in processing certain spatial queries as $SSG(d)$. Similar to $SSG(d)$, $DG(d)$ is defined on top of DT and uses Diversion Neighbourhood ($DN(d)$) to cull edges from DT. Instead of checking all the points that lie in $DN(d)$ between two points, $DG(d)$ only considers points in the neighbouring Delaunay triangles of the edge that is considered for culling. This approach is suitable for inferring travel networks with LBSN data as we assume that the social network data gives us partial data per individual user in terms of its path but with a good picture of where people could be in an event in a city. As explained with the definitions of the $DG(d)$ (Section 3.1), for all endpoint pairs the value of d indicates the preference of inferring a longer alternative path with less distance between all point pairs on the path compared to the direct distance between the endpoint pair. This is useful when we have a very dense point set to cull some connections. Similar to both $SPG(t)$ and $SSG(d)$, as d increases, the number of edges in $DG(d)$ monotonically decreases and therefore the path length between any two non-adjacent points in the skeleton monotonically increases. It is also important to note that GG is a special case of $DG(d)$ when $d = 2$.

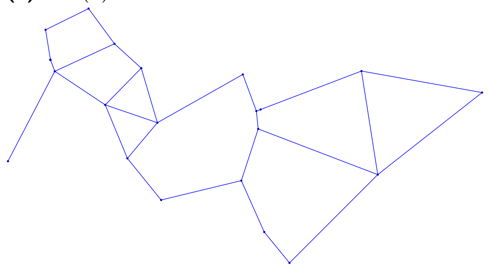
We use publicly available LBSN data to evaluate the performance of $DG(d)$ and $SSG(d)$ for inferring travel networks. We show that $DG(d)$ performs as well as $SSG(d)$ in terms of the quality of the inferred network but $DG(d)$ achieves significantly faster execution times

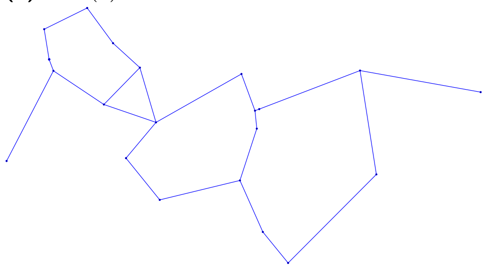

(a) location set.

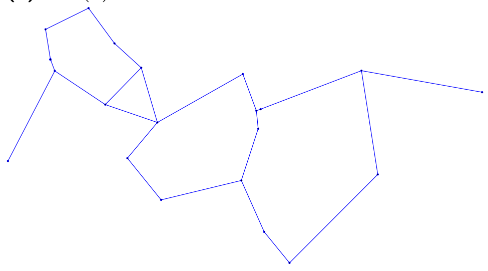

(b) SSG(2) = GG.

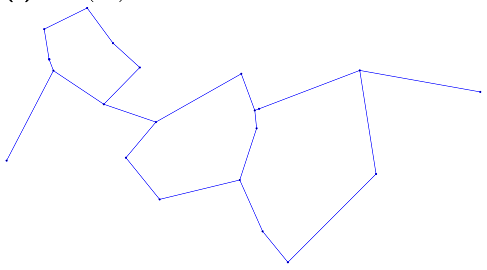

(c) DG(2) = GG.


(d) SSG(4).


(e) DG(4).


(f) SSG(10).


(g) DG(10).


(h) SSG(∞) = RNG.

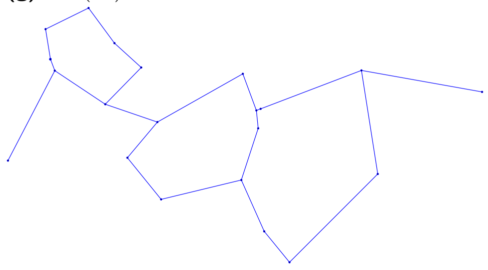

(i) DG(∞) = UG.

Figure 1 SSG(d) and DG(d) skeletons created on a subset of locations from Twitter data set. Note that two skeletons in a row are created using two algorithms, but exhibit the same graph structure.

than $SSG(d)$. In fact, $DG(d)$ takes less than 10% of the time required to infer a movement network than $SSG(d)$. $DG(d)$ contains a few more edges (less than 2% of the total number of edges [1]) compared to $SSG(d)$. However, compared to the time advantage of $DG(d)$, the negative impact of the additional edges in negligible. We observe that the resulting $DG(d)$ and $SSG(d)$ are very similar in terms of their shape and topology.

2 Related Work

In this section, we present the related work in two categories. The first category, LBSN data processing, presents systems and techniques used to process LBSN data. The second category, neighbourhood graphs, provides an overview of neighbourhood graphs related to the proposed Diversion Graph.

2.1 LBSN Data Processing

MacEachren et al. develop an LBSN analysis system SensePlace2 which is used to query and visualize social media data over an interactive map interface [11]. Chae et al. develop another systems for analysing public behaviour using LBSN data [4]. Geospatial heatmaps are used in both systems to provide a summarized view of the spatial distribution of LBSN posts. Many LBSN-based analytics systems support real time processing of LBSN data. For example, RAPID is a real-time analytics platform for interactive data mining [10]. It streams social media data and processes it to generate real time results. There are many types of analytics that can be performed by the systems like RAPID. For example, the detection of the most popular path followed by the users and the extraction of movement corridors [8]. When performing such analytics at real time it is important to have a neighbourhood graph like the proposed Diversion Graph that generates high quality results while minimizing execution time.

2.2 Neighbourhood Graphs

Neighbourhood graphs infer edges between points based on a neighbourhood defined on the points [3]. On two dimensional space neighbourhood represents an area, which can be defined per point, per point pair or per all points in the sample. Neighbourhood graphs that infer edges based on the emptiness (absence of other points within a region) of the neighbourhood surrounding the endpoint pair of the edge are referred as Empty Region Graphs (ERG) [3].

Gabriel Graph (GG) [7] is a static ERG, first proposed as a tool for geographic variation analysis. GG uses the closed circle (a circle where inferring edge becomes a diameter) as the empty region for inferring edges. Relative Neighbourhood Graph (RNG) [17] is another ERG, which uses open lune as the empty regions of inferring its edges. RNG can infer a structure close to human perception of a point set [17]. Both GG and RNG are useful for analysing the shape of a point set.

Urquhart Graph (UG) [18] was first proposed for fast construction of RNG. It was later proved that the UG is not always similar to RNG [16], but UG only differs from RNG by 2% maximally. Therefore UG can be seen as a faster method to approximate RNG [1]. We are combining the thought process behind UG creation and the Diversion neighbourhood of the $SSG(D)$ to create $DG(d)$.

Delaunay Triangulation (DT) is a Triangulated Irregular Network (TIN) with many benefits. It serves as a planar graph which has similar properties as the complete graph. For this reason, it can be used as the starting graph for inferring many other planar graphs. Due to having a low spanning ratio and faster inferring it is used in many travel network analysis problems. Note that $MST \subseteq RNG \subseteq UG \subseteq GG \subseteq DT$. As later shown in properties section, $DG(2) = GG$ and $DG(\infty) = UG$.

Mark de Berg et al. proposed Shortest Path Graph ($SPG(t)$) as a base skeleton for delineating mechanism to identify boundary and cavities within an imprecise region [6]. They show that $SPG(2)$ is better for delineating imprecise regions compared to both Kernel Density Estimation (KDE) and GG [6]. $SPG(t)$'s global evaluation criteria is highlighted as the main reason for its success. However, the quality of results generated using $SPG(t)$ heavily depends on certain parameter settings.

$SSG(d)$ follows the general intuition used in proposing $SPG(t)$, which is to roughly align paths in the graph with paths in the imprecise region. Rather than using global criteria as used in $SPG(t)$, $SSG(d)$ uses local criteria making it faster than $SPG(t)$ and more effective for movement analysis [8].

When the value of the parameter in SPG and SSG approaches infinity, $SPG(t)$ converges to MST and $SSG(d)$ converges to RNG, and our proposed $DG(d)$ converges to UG (Theorem 6) which is a close approximation of RNG. Since UG is a close approximation of RNG, structures generated using $DG(d)$ are closely related to the human perception of the point set. It should be noted that $DG(d)$ may contain some additional edges compared to the $SSG(d)$ with the same d value. However, due to the relaxed nature of evaluation criteria for $DG(d)$, inferring the graph takes much less time compared to inferring $SSG(d)$ or $SPG(t)$.

3 Diversion Graph

Given a set of points, a Diversion Graph ($DG(d)$) connects the points in a traversable manner.

3.1 Definitions

We construct $DG(d)$ in the form of an undirected graph $G(V, E)$ where $V \subseteq \mathbb{R}^2$ represents a given point set and E represents the inferred edges between the points. An edge between two endpoints $p, q \in V$ is represented as $pq \in E$. Length l_{pq} represents the Euclidean distance between the two points.

As $DG(d)$ is defined using Delaunay Triangulation (DT), let us briefly iterate a useful property of DT. DT is a triangulation of a point set, in which each triangle's circumcircle does not contain any other points other than triangle's vertices. Also, DT is the dual of Voronoi diagram. Our proposed graph $DG(d)$ is evaluated by removing some edges from the DT.

▶ **Definition 1** (Diversion Graph). *For $V \subseteq \mathbb{R}^2$, the Diversion Graph of V at $d \in \mathbb{R} : d \geq 2$, denoted $DG(V, d)$ or simply $DG(d)$, is an undirected graph containing a subset of $DT(V)$ such that for each edge $pq \in DT(V)$:*

$$pq \text{ is not an edge of } DG(V,d) \text{ iff } l_{pz}^d + l_{zq}^d \leq l_{pq}^d,$$

where z is the other point in a Delaunay triangle where pq is an edge.

By this definition in common terms, $DG(d)$ is the graph created by removing edges pq from DT if and only if $l_{pz}^d + l_{zq}^d \leq l_{pq}^d$ where z is a vertex of a Delaunay triangle where pq is an edge. Therefore inherently $DG(d)$ is a subgraph of DT. We explore the properties of $DG(d)$ in the next section.

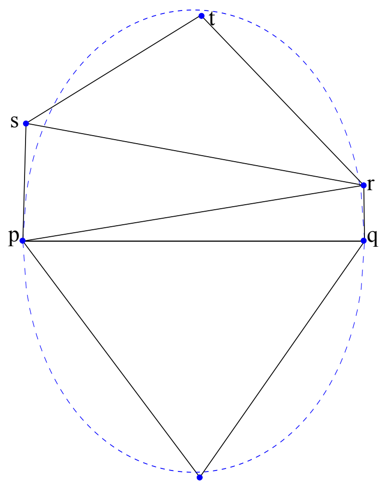


Figure 2 Counter example to show $DG(d)$ does not always equal to $SSG(d)$. Dashed line shows diversion neighbourhood at $d = 4$ ($DN(4)$). Points p and q have equal y coordinates. Both points r and s reside out side of $DN(4)$. Point t lies inside shown $DN(4)$.

3.2 Diversion Graph Properties

Since $DG(d)$ is created by removing edges from DT, we can state the following theorem.

▶ **Theorem 2.** *For* $2 \leq d, DG(d) \subseteq DT$.

Consider the case $DG(2)$ against GG. By comparing definitions of the two graphs we can state the following theorem.

▶ **Theorem 3.** $DG(2) \equiv GG$.

Proof. Consider the definition of GG from [12]. The vertices $p, q \in V$ are least squares adjacent forming the edge $pq \in GG$ iff

$$l_{pq}^2 < l_{pz}^2 + l_{zq}^2 \forall z \in V \setminus \{p, q\}.$$

Furthermore, in the same paper [12] it is proven that the GG can be extracted from DT by evaluating the above inequality on each triangle, for each edge. Now let us look at $DG(2)$ definition. It is extracted from DT by removing edges pq iff $l_{pz}^2 + l_{zq}^2 \leq l_{pq}^2$, for each z which are other points of the Delaunay triangles where pq is an edge. Since both GG and $DG(2)$ are evaluated from DT using the same inequality, they are equivalent. ◀

Since equation used in $DG(d)$ definition is same as the diversion neighbourhood definition [8], one may think $DG(d)$ and $SSG(d)$ are the same thing. As shown in the following theorem, $SSG(d) \subseteq DG(d)$.

▶ **Theorem 4.** *For* $2 \leq d, SSG(d) \subseteq DG(d)$.

Proof. Consider the five points p, q, r, s, t in Figure 2. Their Delaunay triangulation is shown in solid straight lines. Points p and q have equal y coordinates. The dashed line indicates diversion neighbourhood at $d = 4$ ($DN(4)$) between p and q. Both points r and s reside outside of $DN(4)$. Point t lies inside shown $DN(4)$. Let us consider inclusion of edge pq in $DG(4)$ and $SSG(4)$. Since t is within the $DN(4)$ of pq, pq will not be an edge of $SSG(4)$. However, since $DG(4)$ considers only points in neighbouring triangles of pq, it only considers point r when considering the inclusion of pq. Since r is outside the $DN(4)$ of pq, pq becomes an edge of $DG(4)$. Therefore, for $2 < d, SSG(d) \subseteq DG(d)$. ◀

In fact, $DG(d)$ can contain more edges than $SSG(d)$. Therefore we can state the following corollary.

▶ **Corollary 5.** *For $2 < d, DG(d)$ is not always equal to $SSG(d)$.*

Let us consider behaviour of $DG(d)$ when $d \to \infty$.

▶ **Theorem 6.** *As $d \to \infty, DG(d) \to UG$.*

Proof. When $d \to \infty$, by $DG(d)$ definition for an edge pq to be removed from DT, both other edges of the neighbouring Delaunay triangles only needs to be shorter than pq. In other words, if pq is the longest edge in a Delaunay triangle it will be removed from $DG(d)$ when $d \to \infty$. UG is created from DT by removing the longest edge of each Delaunay triangle. Therefore, as $d \to \infty, DG(d) \to UG$. ◀

▶ **Theorem 7.** *For $2 \leq d, DG(d)$ is planar and connected.*

Proof. Since for $2 \leq d, DG(d) \subseteq DT$, $DG(d)$ is planar. Inequality used in $DG(d)$ is the same as the diversion neighbourhood of $SSG(d)$. In [8] it is shown that diversion neighbourhood does not get bigger than open lune neighbourhood. Open lune neighbourhood is proven as the tight neighbourhood that ensures a connected edge embedding in empty region graphs in [3]. Therefore $DG(d)$ is connected. Combining these arguments we can say for $2 \leq d, DG(d)$ is planar and connected. ◀

Next we consider the relationship between two $DG(d)$s as d increases.

▶ **Theorem 8.** *For $2 \leq d \leq d', DG(d') \subseteq DG(d)$*

Proof. Define the *edge weight* of pq with respect to d' as $l_{pq}^{d'}$, for some $d' \geq 2$. Assume that for all $z \in \Lambda(pq) \setminus \{p, q\}$, $l_{pz}^{d'} + l_{zq}^{d'} > l_{pq}^{d'}$, where $\Lambda(pq)$ is the set of points in neighbouring Delaunay triangles of pq. In this case, pq is an edge in $DG(d')$. Now we show that for $d \leq d'$, pq is also an edge in $DG(d)$. Let us write $d = d'\,\epsilon$ where $\frac{2}{d'} \leq \epsilon \leq 1$. Then we need to show that:

$$\begin{aligned} l_{pz}^{d'\,\epsilon} + l_{zq}^{d'\,\epsilon} &> l_{pq}^{d'\,\epsilon} \qquad (1)\\ \frac{l_{pz}^{d'\,\epsilon} + l_{zq}^{d'\,\epsilon}}{l_{pq}^{d'\,\epsilon}} &> 1 \\ \left(\frac{l_{pz}}{l_{pq}}\right)^{d'\,\epsilon} + \left(\frac{l_{zq}}{l_{pq}}\right)^{d'\,\epsilon} &> 1 \\ \left(\left(\frac{l_{pz}}{l_{pq}}\right)^{d'\,\epsilon} + \left(\frac{l_{zq}}{l_{pq}}\right)^{d'\,\epsilon}\right)^{\frac{1}{\epsilon}} &> 1 \end{aligned}$$

Since the function $x \mapsto x^{\beta}$ is subadditive for $\beta \geq 1$ then:

$$\left(\left(\frac{l_{pz}}{l_{pq}}\right)^{d'\,\epsilon} + \left(\frac{l_{zq}}{l_{pq}}\right)^{d'\,\epsilon}\right)^{\frac{1}{\epsilon}} \geq \left(\frac{l_{pz}}{l_{pq}}\right)^{d'} + \left(\frac{l_{zq}}{l_{pq}}\right)^{d'}.$$

We know the right hand side is greater than 1 due to our initial assumption and therefore Eq. 1 is true. Therefore pq is also an edge in $DG(d)$ and this completes the proof. ◀

Algorithm 1 Create $DG(d)$.

```
Input: V - Filtered locality set
d - Configuration parameter
Output: $DG(d)$
1  $DT$ ← create Delaunay Triangulation of $V$
2  initialize $DG(d)$ to empty set
3  foreach ($Edge\ pq : pq \in DT$) do
4      foreach ($Point\ z : z \in \Lambda(pq) \setminus \{p, q\}$) do
5          if ($l^d_{pq} < l^d_{pz} + l^d_{zq}$) then
6              Add $pq$ to $DG(d)$
7          end
8      end
9  end
10 return $DG(d)$
```

3.3 Algorithms

In this section, we present an efficient algorithm to compute $DG(d)$. Since $DG(d)$ is defined based on DT we can use DT as the starting graph to compute $DG(d)$. There are two approaches we can use to compute $DG(d)$ using DT. One approach is to process DT as triangles and check each edge of the triangle for removal from DT. The second approach is to process DT as a set of edges and evaluate each edge against the points in the neighbouring Delaunay triangles to check whether they belong in $DG(d)$. The approach we are presenting in this paper is the second approach which evaluates edges to check their membership of $DG(d)$, as it can be easily compared with the d-spectrum algorithm of the $SSG(d)$.

We propose a simple and effective algorithm to calculate $DG(d)$ (Algorithm 1). In the algorithm, $\Lambda(pq)$ is the set of points in neighbouring Delaunay triangles of pq. Each other point in the $\Lambda(pq) \setminus \{p, q\}$ are evaluated against pq to see whether pq is an edge of $DG(d)$. For simplicity, the condition in line 5 in Algorithm 1 is directly derived from the definition of $DG(d)$. However, it can be further improved by checking whether other edges connected with $\Lambda(pq)$ are longer than pq. The algorithm is readily parallelizable as there is no race condition between separate edge evaluations.

Let us consider the time complexity of the proposed algorithm for calculating $DG(d)$. In line 3, as DT has $\mathcal{O}(n)$ edges, the code between line 4 and line 8 runs $\mathcal{O}(n)$ times. As each edge pq has at most two neighbouring triangles, the code between line 5 and line 7 runs at most two times per edge. Line 5 is assumed to run in $\mathcal{O}(1)$ time. Therefore, the whole algorithm runs in $\mathcal{O}(n)$ time.

3.3.1 Improving Running Time of $SSG(d)$

Introduction of $DG(d)$ allows us to efficiently calculate $SSG(d)$ for a specific $2 \leq d$ value without calculating d-Spectrum [8]. Since $DG(d)$ is a super graph of $SSG(d)$ for $2 \leq d$, once $DG(d)$ is calculated we can use it to evaluate those edge using the triangle sweeping method presented in Algorithm 1 of [8]. As later shown in the experiments $DG(d)$ only contains a very small number of additional edges compared to $SSG(d)$. Therefore this is a very efficient method of computing $SSG(d)$.

However, it should be noted that computing $SSG(d)$ from $DG(d)$ may be slower for varying skeleton generation compared to using d-Spectrum. Since d-Spectrum pre-compute the minimum d-value necessary for an DT edge to be in the $SSG(d)$, varying skeleton generation takes less time. But for generating $SSG(d)$ for a specific $2 \leq d$ using $DG(d)$ is faster than creating d-Spectrum.

3.4 Applications

The proposed $DG(d)$ can be used in many applications detailed as follows.

3.4.1 Nearest Neighbour Queries

The $DG(d)$ graph structure can be used to search for the path to the nearest interesting locations from a given location. For example, this kind of query can be used to find the nearest exit gate in a park. We can use breadth first search starting from the query location and traverse the graph until a required interesting point is found.

Similarly, we can perform breadth first search on $DG(d)$ for finding k-nearest neighbours. Instead of stopping breadth first search when the first interesting point is found, it can be continued until k interesting points are found. As for the edge weights, we can use weights calculated in the section 3.4.4 according to the usage of edges. This will make sure that the most popular path to the nearest neighbour will be found. This approach can be extended to solve reverse nearest neighbour queries and group nearest neighbour queries as suggested in [9].

3.4.2 Refinement of Movement Corridors

Once $DG(d)$ is created using posted localities in LBSN data, user trajectories can be used to refine the created travel network. The approach for refining the travel network is as follows. For each consecutive location pair in user trajectories, the shortest path is determined using $DG(d)$. For each $DG(d)$ edge, the number of trajectories passed through that edge is recorded as a usage count (Definition 9). We can then represent movement corridors in the travel network based on the edges where the usage counts are higher than a given threshold.

▶ **Definition 9** (Usage count). *Assuming a path is a sequence of edges traversed by a trajectory trace, for all $pq \in E$, Usage Count of pq (denoted $UC(pq)$), is defined as the trajectory count,*

$$UC(pq) = |\{path : pq \in path\}|$$

One of the problems of using $DG(d)$, is that it does not consider the existence of obstacles. As trajectories do not appear on obstacles such as rivers, incorporating trajectory information into $DG(d)$ allows filtering edges not used by the trajectories. By filtering edges not used by trajectories, we are able to eliminate edges that do not represent user movement information. In summary, refined movement corridors calculated using $DG(d)$ is an edge subset of $DG(d)$ which are used by the trajectories for movement.

3.4.3 Inferring Road Network

The aforementioned approach for refining movement corridors can be used to infer the road network in an area where we do not have prior knowledge about the road networks. Ideally for this purpose, we need GPS locations published on the road network. The easiest way to obtain such information is to collect LBSN post published while travelling in vehicles. Using the GPS data in LBSN posts and associating the GPS points with trajectories, we can infer the road network in an area.

3.4.4 Most Popular Paths

After calculating usage counts of $DG(d)$ edges, the counts can be used to find the most popular path between locations. To find the most popular path, the edge weight in $DG(d)$ should reflect the popularity of the edges. We can use a shortest path algorithm to calculate the paths. The edge weights are defined in Definition 10. To ensure edges with more usage have lower weights, we divide the length of the edge by the usage count of that edge. For an edge with no usage, the edge length multiplied by a fixed value is used as the edge weight. After calculating edge weights in this manner, the Dijkstra's shortest path algorithm can be used to find the most popular path between two locations.

▶ **Definition 10** (Edge Weight). *For all $pq \in E$, weight of pq is defined as,*

$$weight(pq) = \begin{cases} l_{pq}/UC(pq), & \text{if } 0 < UC(pq). \\ l_{pq} \times C : 1 < C, & \text{if } UC(pq) = 0. \end{cases}$$

3.4.5 Other Applications

$DG(d)$ can be used in other applications such as tour recommendation, trajectory clustering and group movement detections [9]. For all these applications we need to process user trajectories after creating the initial graph structure to incorporate additional movement related information to the created graph skeleton.

4 Experiments

4.1 Data Sets

We conducted experiments on two real world LBSN datasets and one synthetic data set. The first real data set consists of geo-located posts collected from Twitter. It is collected from 06th March to 23rd April 2012, within a bounding box over Australia and New Zealand. It contains 724651 LBSN posts authored by 36639 users. For our analysis, an LBSN post is defined as a tuple containing four elements - userID, voluntarily generated textual content, timestamp and the location where the post was authored.

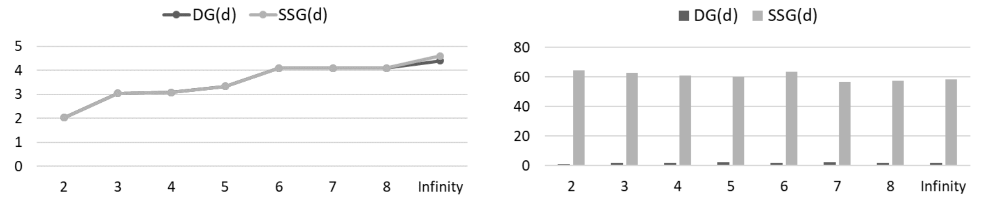
We used Yahoo! Flickr Creative Commons 100M (YFCC100M) dataset [15] as the second real dataset. It contains metadata such as user information, timestamp and location of 100 million photos and videos shared on Flickr. Only the entries with point geo-locations were used for our experiments. For our experiment, we use the localities around the Thames river in London from the YFCC100M data set.

The synthetic data set for our experiments was generated using SMARTS simulator [13]. We simulated vehicle movement in the Melbourne central business district (CBD) and collected GPS locations of the vehicles every 0.5 seconds. The data set used for our experiment contains 100000 GPS points.

4.2 Implementation

To visualize the inferred neighbourhood skeletons, a visualization tool was implemented utilizing GeoTools[1] Java libraries. All the skeleton visualizations presented in this paper are generated using this tool. Both $DG(d)$ and $SSG(d)$ algorithms are implemented using Java 8. The datasets are stored in a MongoDB database.

[1] `http://www.geotools.org/`



(a) Spanning ratio with varying configuration parameter d.

(b) The time to generate the graphs in mili seconds with varying d.

Figure 3 Graphs depicting different properties between $DG(d)$ and $SSG(d)$.

To infer $DG(d)$, firstly, DT is created using SweepHull [14] algorithm. Then, $DG(d)$ is calculated using Algorithm 1. $SSG(d)$ is extracted from the planar d-Spectrum created using DT. We used numerical analysis to calculate d-value of an edge. More specifically, a Java method was implemented to perform Secant method[2] to approximate the d-Value. The same DT was reused for construction of $DG(d)$ and $SSG(d)$ with different d values. We selected 3 as the configuration value for d to compare resulting graphs generated using $DG(d)$ and $SSG(d)$ based on preliminary tests.

4.3 Results

4.3.1 Event Analysis

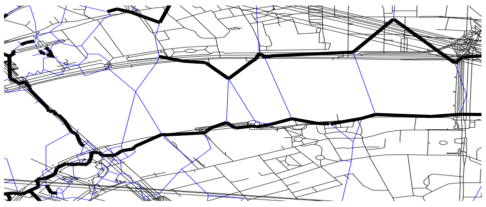
In this experiment, we use a Twitter dataset relating to Queensland state election 2012 [3]. All users participating in the event are there for a common reason and exhibit a similar movement pattern. We implement an LBSN post filtering technique used in [8], to filter posts relating to the election. For temporal bound of the dataset, we took the time period between 23rd and 26th of March 2012. As for the spatial bound, we considered a bounding box over the Queensland state. We consider all users who have posted with "#qldvote" hashtag within spatial and temporal bounds of the event. The data set contains, 1270 unique points after filtering the original Twitter data.

We generate skeletons using $DG(d)$ and $SSG(d)$ with different settings of d. The spanning ratio [2] of graphs are calculated with varying configuration parameters. Spanning ratio of a graph indicates the maximum ratio between the shortest path distance over the graph and direct distance between any point pair. Therefore, graphs with low spanning ratios are preferred to represent movement networks [2].

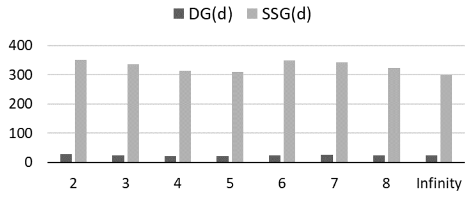
Figure 3 (a) shows the variation of spanning ratio as configuration parameter varies to demonstrate how the shortest path distances between locality pairs change. Both $DG(d)$ and $SSG(d)$ have a low and stable spanning ratio, making them suitable for movement analysis. Furthermore, both $DG(d)$ and $SSG(d)$ have the same spanning ratio when d is less or equal to 8.

The time taken to calculate skeletons of $DG(d)$ and $SSG(d)$ are shown in Figure 3 (b). Execution time for $DG(d)$ calculation is around 95% less compared to $SSG(D)$ for all configuration values. The relaxed criteria for culling edges in $DG(d)$ algorithm gives it a significant advantage in computation time.

[2] `https://en.wikipedia.org/wiki/Secant_method`
[3] `https://en.wikipedia.org/wiki/Queensland_state_election,_2012`

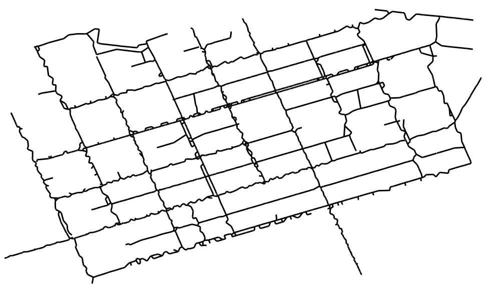


(a) Thick lines indicate refined movement corridors extracted using $DG(3)$ and $SSG(3)$.

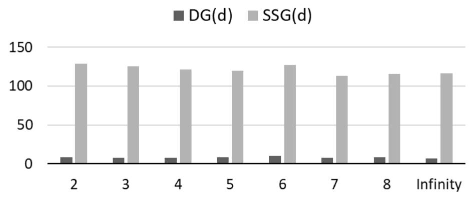


(b) Execution time in mili seconds to generate each graph with varying d.

Figure 4 Results of movement corridor refinement.



(a) Road network extracted using $DG(3)$ and $SSG(3)$.



(b) Execution time in mili seconds to generate each graph with varying d.

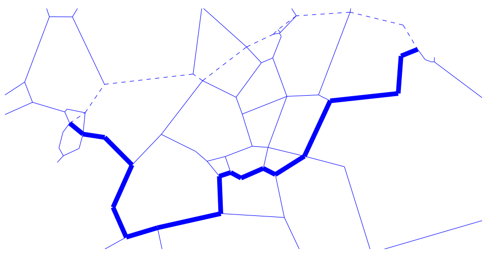
Figure 5 Results of road network extraction.

4.3.2 Movement Corridors Refinement

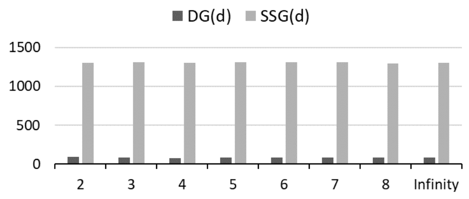
The refined movement corridors refer to the edges of the graph that are used by the users on the move. These edges are selected by aligning user trajectories along the graph edges using shortest path calculation. We analyse the refined movement corridors relating to the trajectories filtered from the YFCC100M dataset, which is collected from around the Thames river in London. We take locations posted over one month. The total number of locations is 6318. To represent the travel networks, $DG(3)$ and $SSG(3)$ are used. This data set is selected because it has higher randomness in tourist movement compared to the Twitter data set. After that trajectories are aligned to both graph skeletons, and all the edges with usage counts of more than 5 are filtered as refined movement corridors. That is, if an edge is used by 5 or more trajectories, the edge is selected as a refined movement corridor. Both graphs resulted in the same refined movement corridor structure (Figure 4 (a)). Edges created across the river are filtered out as there cannot be any movement on those edges. Figure 4 (b) shows the execution times taken to calculate the graph structures. We can see that the time for creating $DG(d)$ is only 8% of the time taken to create $SSG(d)$.

4.3.3 Road Network Inference

Refined movement corridor extraction can be used to infer the road network of an area. Using our synthetic dataset we infer the road network of the Melbourne CBD. In order to simulate the sparseness of data points in LBSN data, we filtered out some of the points in the original synthetic dataset. The filtering process first sorts all GPS points based on the timestamp and then takes one point for every n points from the sorted set. There are 2763 locations collected for this experiment.



(a) Most popular path extracted using $DG(3)$ and $SSG(3)$ is shown in thick lines.



(b) Execution time in mili seconds to generate each graph with varying d.

Figure 6 Results of most popular path extraction.

In Figure 5 (a), we show the road networks inferred using $DG(3)$ and $SSG(3)$. The two road networks almost totally overlap with each other. It should be noted that as the filtering parameter n grows, the data set used to infer road network become more sparse, degrading the result road network. Results heavily degrade when n reaches around 120. Figure 5 (b) shows the execution times taken to calculate the graph structures. We can see that $DG(d)$ creation takes 7% of the time that is used for creating $SSG(d)$.

4.3.4 Most Popular Path Finding

By calculating edge weights to reflect the popularity of edges we can use the resulting graph to calculate the most popular paths. We used the Twitter data set to run experiments on extracting the most popular paths. This data set was used because it contains users with regular movement patterns. We executed the experiment in Melbourne city area where we found 28431 locations. Figure 6 (a) shows a most popular path found between two locations. $DG(3)$ and $SSG(3)$ produce the same path. The dashed lines indicate the shortest path between the two locations while the thick lines indicate the most popular path. When comparing this result with a map there are roads along the most popular path detected while there are building on top of the shortest path. Overall 78% of the edges selected for the most popular path were sitting on the road network of the Melbourne city. Figure 6 (b) shows the executions times taken by $DG(d)$ and $SSG(d)$ to create graphs. Due to the size of the dataset $SSG(d)$ has resulted in running times longer than one second. However, $DG(d)$ has generated the graph in 7% of the time taken by the $SSG(d)$, making it suitable for processing large data sets in real time.

5 Discussion and Future Works

From our experiments, it is clear that given a point set $DG(d)$ creation takes less time compared to $SSG(d)$ creation. Also, $DG(d)$ shows the similar effectiveness as the $SSG(d)$ in solving various queries. The low execution time of DG(d) is due to several reasons. Firstly, for any edge, $DG(d)$ creation algorithm (Algorithm 1) only processes two triangles. However, for $SSG(d)$, it can be empirically shown that per edge at least three triangles are processed. This effect should give a $2:3$ advantage to $DG(d)$ creation compared to $SSG(d)$. However, our result shows that the ratio of the execution times are $1:10$ between $DG(d)$ and $SSG(d)$. Reason for this massive difference is due to the numerical analysis method used to evaluate the d-value of an edge for $SSG(d)$. For $DG(d)$, only a simple inequality is evaluated based on Definition 3.1, per processing triangle. For d-spectrum calculation method of $SSG(d)$, the numerical analysis method runs to determine the least empty diversion neighbourhood around an edge. In our implementation, the analysis method used by $SSG(d)$ is the Secant method , which may need to run hundreds of iterations when processing one edge. Therefore we experience this massive time difference between $DG(d)$ and $SSG(d)$. Due to this reason, it is advantageous to use $DG(d)$ in applications where very little processing time is available to generate results. This also highlights the need for looking into faster ways to locate the d-value for $SSG(d)$, as future work.

In our experiments, we have used data sets with few thousands of locations. As the dataset size increases, one may think we can apply existing data processing techniques applicable on dense GPS data. Even though the LBSN datasets are large, the locations in the datasets are spread across large areas , resulting in a low density of data points. Therefore, existing techniques for processing high-density GPS data may not be suitable for processing LBSN datasets.

The future work on $DG(d)$ can include the autonomous detection of configuration value d and the analysis of the relationship between $DG(d)$ and β-Skeletons. Determining the bounds of the spanning ratio for $DG(d)$ is another promising future direction. Investigating how $DG(d)$ can be used in more application scenarios is also an interesting research topic. Incorporating geographical feature when solving queries with $DG(d)$ is also an interesting research topic. With the introduction of $DG(d)$ as a super graph of $SSG(d)$ it is necessary to investigate how much faster $SSG(d)$ can be generated using $DG(d)$. Since $DG(d)$ definition is distance based, the concept of $DG(d)$ can be extended to higher dimensions and with different distance measurements.

6 Conclusion

We presented the Diversion Graph ($DG(d)$), a connected graph that varies depending on a single parameter d. We analysed how $DG(d)$ relates to some well-known graph structures, and we presented how $DG(d)$ can be used to improve running time of the state-of-the-art graph, the Stepping Stone Graph ($SSG(d)$). We have empirically shown that $DG(d)$ is both efficient and effective to analyse LBSN data due to its distance based local evaluation criteria.

References

1 D. V. Andrade and L. H. de Figueiredo. Good approximations for the relative neighbourhood graph. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 25–28, 2001. URL: http://www.cccg.ca/proceedings/2001/lhf-96805.ps.gz.

2 P. Bose, L. Devroye, W. S. Evans, and D. G. Kirkpatrick. On the spanning ratio of gabriel graphs and beta-skeletons. *SIAM J. Discrete Math.*, 20(2):412–427, 2006. doi:10.1137/S0895480197318088.

3 J. Cardinal, S. Collette, and S. Langerman. Empty region graphs. *Computational geometry*, 42(3):183–195, 2009. doi:10.1016/j.comgeo.2008.09.003.

4 J. Chae, D. Thom, Y. Jang, S. Kim, T. Ertl, and D. S. Ebert. Public behavior response analysis in disaster events utilizing visual analytics of microblog data. *Computers & Graphics*, 38:51–60, 2014. doi:10.1016/j.cag.2013.10.008.

5 J. Cortés, S. Martínez, and F. Bullo. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Transactions on Automatic Control*, 51(8):1289–1298, 2006. doi:10.1109/TAC.2006.878713.

6 M. de Berg, W. Meulemans, and B. Speckmann. Delineating imprecise regions via shortest-path graphs. In I. F. Cruz, D. Agrawal, C. S. Jensen, E. Ofek, and E. Tanin, editors, *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems,*

ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings, pages 271–280. ACM, 2011. doi:10.1145/2093973.2094010.

7 K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Biology*, 18(3):259–278, 1969.

8 S. Kannangara, E. Tanin, A. Harwood, and S. Karunasekera. Stepping stone graph for public movement analysis. In F. Banaei-Kashani, E. G. Hoel, R. H. Güting, R. Tamassia, and L. Xiong, editors, *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2018, Seattle, WA, USA, November 06-09, 2018*, pages 149–158. ACM, 2018. doi:10.1145/3274895.3274913.

9 S. Kannangara, E. Tanin, A. Harwood, and S. Karunasekera. Stepping stone graph: A graph for finding movement corridors using sparse trajectories. *ACM Trans. Spatial Algorithms and Systems*, 5(4):23:1–23:24, 2019. doi:10.1145/3324883.

10 K. H. Lim, S. Jayasekara, S. Karunasekera, A. Harwood, L. Falzon, J. Dunn, and G. Burgess. RAPID: real-time analytics platform for interactive data mining. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, and N. Hurley, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part III*, volume 11053 of *Lecture Notes in Computer Science*, pages 649–653. Springer, 2018. doi:10.1007/978-3-030-10997-4_44.

11 A. M. MacEachren, A. R. Jaiswal, A. C. Robinson, S. Pezanowski, A. Savelyev, P. Mitra, X. Zhang, and J. I. Blanford. Senseplace2: Geotwitter analytics support for situational awareness. In *2011 IEEE Conference on Visual Analytics Science and Technology, VAST 2011, Providence, Rhode Island, USA, October 23-28, 2011*, pages 181–190. IEEE Computer Society, 2011. doi:10.1109/VAST.2011.6102456.

12 D. W. Matula and R. R. Sokal. Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis*, 12(3):205–222, 1980.

13 K. Ramamohanarao, H.Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn. Smarts: Scalable microscopic adaptive road traffic simulator. *ACM TIST*, 8(2):26:1–26:22, 2017. doi:10.1145/2898363.

14 D. Sinclair. S-hull: a fast radial sweep-hull routine for delaunay triangulation. *CoRR*, abs/1604.01428, 2016. arXiv:1604.01428.

15 B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L. Li. YFCC100M: the new data in multimedia research. *Commun. ACM*, 59(2):64–73, 2016. doi:10.1145/2812802.

16 G. T. Toussaint. Comment: Algorithms for computing relative neighbourhood graph. *Electronics Letters*, 16(22):860–860, October 1980.

17 G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

18 R. B. Urquhart. Algorithms for computation of relative neighbourhood graph. *Electronics Letters*, 16(14):556–557, July 1980.

19 X. Wei and X. Angela Yao. A conceptual framework for representation of location-based social media activities (short paper). In S. Winter, A. Griffin, and M. Sester, editors, *10th International Conference on Geographic Information Science, GIScience 2018, August 28-31, 2018, Melbourne, Australia*, volume 114 of *LIPIcs*, pages 62:1–62:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.GISCIENCE.2018.62.

20 J. Xie, T. Yang, and G. Li. Extracting geospatial information from social media data for hazard mitigation, typhoon hato as case study (short paper). In S. Winter, A. Griffin, and M. Sester, editors, *10th International Conference on Geographic Information Science, GIScience 2018, August 28-31, 2018, Melbourne, Australia*, volume 114 of *LIPIcs*, pages 65:1–65:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.GISCIENCE.2018.65.