Towards Identifying the Best New Connection in a Spatial Network:

Optimising the Performance of Hole Discovery

Femke Reitsma
The Department of Geography,
The University of Canterbury,
Christchurch,
New Zealand
femke.reitsma@canterbury.ac.nz

Tony Dale, William Pearse
BlueFern Supercomputing and Services Facility,
The Univesrity of Canterbury,
Christchurch,
New Zealand
tony.dale@canterbury.ac.nz

Abstract—Networks are used to represent phenomena such that we can measure their structure. Spatial networks are a special class of networks that reflect the embedding space within which the network is contained, incorporating the property of spatial autocorrelation and its impact on network measures. These measures of spatial network structure have thus far largely focused on the structure of the network, as opposed to the absence of that structure. This paper builds on past work in identifying an aspect of the absence of structure, that of identifying holes or chordless cycles in a network. It is the first step in identifying the best new connection to make in the network, identifying where the absence of structure is having the most significant impact. This paper presents the implementation of optimisations in discovering network holes.

Keywords-network; spatial network; chordless cycle; hole.

I. INTRODUCTION

Networks (or graphs) are used widely to model and analyse features in the world that either appear network like, such as infrastructure and river networks [1][2], or that we can model as a network, such as the relationships among groups of people in a social network [3]. Identifying the best new connection to create in a spatial network for the purposes of increasing the connectivity of that network has many relevant geographic applications. For example, identifying the best place to put a new cycle path that maximally increases the overall connectivity of the network, or identifying the best new connection in a power supply network that reduces the potential for power outages. Part of the process of identifying the best new connection is to first determine where the gaps in connectivity are and to recognise which would be the best gap to fill.

This paper presents the optimised implementation of a part of the process of identifying the best new connection in a spatial network. It improves past research that has developed the methodology but which was limited to very small networks due to an inefficient algorithm implementation [4]. This paper does not explore the application context where the discovered holes could be prioritized for developing a new connection across them, as this is the scope of future work.

The paper begins by reviewing some of the background literature on spatial networks, which is followed by a description of how holes in a network are discovered, presenting search optimisation methods in Section 2. Efficient methods for storing the input and output of the search method are described in Sections 3 and 4 respectively. And in Section 5, performance optimisations are discussed, followed by concluding comments in Section 6.

II. BACKGROUND

Work on spatial networks has developed in many different fields, including transportation engineering, hydrology and social geography. Research has repeatedly found that spatial networks have properties that are distinct from other kinds of networks due to spatial autocorrelation. Consequently they require unique measures [6][7]. For example, Ravasz and Barabasi [7] have found that scale-free patterns that are found in a large range of networks, such as the world wide web, do not exist in geographic networks, such as internet routers and power grid structures.

Barthélemy [6] thoroughly reviews measures for spatial networks, all of which focus on local and global measures that characterise a network, such as the work by Cardillo *et al.* [8], who describe the structural properties of urban street networks. Little thought has thus far been given to the absence of that structure, which is the focus of this paper, other than comparing the measures of network structure when new links or nodes are added or removed. For example, Buhl *et al.* [9] consider the robustness of street networks, measuring the robustness of a network by studying how it becomes fragmented as an increasing number of nodes are removed, where the impact on street network reliability varies whether nodes of high degree or low degree are removed.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-178-6

An algorithm for identifying one kind of lacking structure, namely holes, or chordless cycles, has been developed by Chandrasekharan, Lakshmanan and Medidi [5]. We build upon this early work and present an alternative implementation and the details of simple steps to optimize the performance of the algorithm presented below.

III. DETECTING HOLES

A hole in a network is mathematically defined as a chordless cycle. A chordless cycle is a cycle in a graph, a path such that the first node is connected to the last node, which includes at least four nodes with no chord connecting those nodes. A minimum size can also be defined, such that it is composed of more than four nodes.

For the future purposes of the application of this method to spatial data, the relationship between each node and its spatial identifier can be stored in an external spatial data file. The discovered holes can then be highlighted in the spatial data by joining the appropriate tables.

A. Search method

To detect a chordless cycle a vertex **v** from the graph is selected. This vertex and all outbound edges from this vertex are considered a tree structure, with the root of the tree being vertex **v**. A depth-first search (traversal) of this tree structure is performed. When the search returns to the vertex **v** we have discovered a cycle. This method can be described by the following recursive algorithm:

```
dfs (graph, startVertex, currentVertex,
visitedVertices) {
   if(visitedVertices.contains(currentVertex)){
      if (currentVertex == start){
         // we have found a cycle! do something...
      }
      return;
   }
   visitedVertices.add(currentVertex);
   for each (outboundChildVertex from
                            currentVertex){
       dfs(graph, startVertex,outboundChildVertex,
                             visitedVertices)
   }
   visitedVertices.remove(currentVertex);
}
```

Each cycle found must be checked for the following conditions:

- The cycle contains at least four vertices.
- Any non-adjacent vertices in the cycle are not connected by a single graph edge.

B. Search method optimisation

The disadvantage of the basic search method described in earlier is that each vertex will be visited many times before the search completes. This leads to very high computational complexity. The computational complexity can be significantly reduced by removing vertex **v** from the graph after a depth-first search from vertex **v** has been completed. If the depth-first search begins a vertex $\mathbf{v}_n$ this optimization can be implemented by instructing the search algorithm to ignore any vertices $\mathbf{v}_m$ where $m < n$.

In almost all instances, this significantly reduces the number of edges that must be traversed to complete subsequent searches. Removing vertex **v** from the graph does not reduce the number of cycles found, because after the search (beginning at vertex **v**) has been completed all cycles beginning and ending at vertex **v** have been found. Furthermore, all cycles beginning and ending at vertex **v** are actually all cycles containing vertex **v**. This is due to the nature of a cycle. Removing these vertices from the graph also has the positive effect of removing duplicate cycles from the search results, as all cycles beginning and ending at a vertex actually contain that vertex.

IV. INPUT NETWORK FORMAT

A network can be expressed as an adjacency matrix, which we use as the input structure for the method presented. The matrices are read and written through plain text files, with formatting based on the UCINET full matrix format [10].

The basic format of a UCINET full matrix text file is a two-line header, followed by the data. The data values must be separated by at least one space and commas or other punctuation symbols are not allowed. We have simplified this for our purposes, where the first line in the file is the number of columns, and therefore rows, in the matrix. The remainder of the file must contain a number of 1 or 0 characters. The exact number depends on the size of the matrix. The second and subsequent lines in the file represent each node in the graph; and its connection via edges to other nodes in the graph are indicated by a 1. For example, Figure 1 is a 5 x 5 matrix in a simplified UCINET matrix format:

```
5
0 1 1 1 1
1 0 1 0 0
1 1 0 0 1
1 0 0 0 0
1 0 1 0 0
```

Figure 1. Sample matrix format

The corresponding graph for the matrix above is depicted below in Figure 2.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-178-6

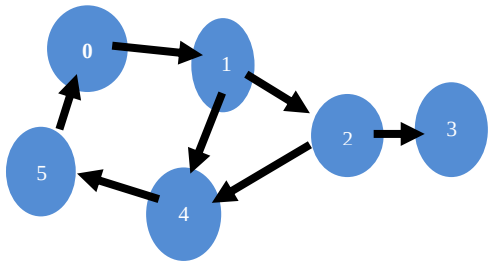

Figure 2. Network corresponding to simple adjacency matrix

The same matrix can also be represented in this compact representation:

```
5
011111010011001100010100
```

V. RECORDING SEARCH RESULTS

Consider the graph depicted in Figure 2 above. This graph has 6 vertices and 7 directed edges. Whether the edges are directed or not makes no difference to the problem, however directed edges significantly simplify the problem explanation.

This graph can be represented by the adjacency matrix shown below in Figure 3. Rows and columns are labeled 0-5 for convenience. A value of 1 in an adjacency matrix element indicates the presence of an edge. Row i in the adjacency matrix depicts the outgoing edges from node i. Similarly, column j contains the incoming edges for node j.

```
  0 1 2 3 4 5
0 0 1 0 0 0 0
1 0 0 1 0 1 0
2 0 0 0 1 1 0
3 0 0 0 0 0 0
4 0 0 0 0 0 1
5 1 0 0 0 0 0
```

Figure 3. Input adjacency matrix

A simple back-stepping depth-first search beginning at each node in this graph will reveal the presence of four chordless cycles within this graph. These four cycles are:

```
0-1-4-5, 1-4-5-0, 4-5-0-1, 5-0-1-4
```

It is immediately obvious that these four cycles are actually one cycle. If, however, vertex 0 is removed from the search space after the first cycle `0-1-4-5` is found (as proposed in Section 2), the three following duplicate cycles will not be discovered.

It is important, especially in the case of very large networks, that the solution is stored in a compact format. This can be done by storing the edges of each chordless cycle in a second adjacency matrix. For every cycle found, each edge in the cycle is stored in the adjacency matrix. Any existing edge in this second adjacency matrix may be overwritten. Following the discovery of the four chordless noted above, they are stored in an adjacency matrix, which contains only the edges from these chordless cycles:

```
0-1, 1-4, 4-5, 5-0
```

Despite the identification of the same chordless cycle four times, there is no duplication of edges in the solution adjacency matrix as they are overwritten (Figure 4).

```
  0 1 2 3 4 5
0 0 1 0 0 0 0
1 0 0 0 0 1 0
2 0 0 0 0 0 0
3 0 0 0 0 0 0
4 0 0 0 0 0 1
5 1 0 0 0 0 0
```

Figure 4. Solution adjaceny matrix with chordless cycles

To reconstruct the chordless cycles the original back-stepping depth-first search is repeated on the graph depicted by the solution adjacency matrix.

This method of storing chordless cycles has a number of advantages. First, it requires a known storage capacity, in contrast to other storage techniques such as flat-file format, or linked-lists, which require non-deterministic storage capacity. The storage capacity required is compact, where duplicate cycles do not cause the required storage capacity to 'balloon'. By comparing elements of the first and second adjacency matrices the nodes that do (or do not) appear in chordless cycles can easily be determined (Figure 5). Very little computational effort is required to store data in the matrix and retrieve it. The adjacency matrix containing chordless cycles is typically sparse compared to the original adjacency matrix, making retrieval of cycles relatively effortless. The main disadvantages of storing chordless cycles in this manner is that it may take a significant amount of time to obtain a list of chordless cycles from the matrix because the reconstruction algorithm is the same as the algorithm used to find the cycles.

```
  0 1 2 3 4 5
0 0 0 0 0 0 0
1 0 0 1 0 0 0
2 0 0 0 1 1 0
3 0 0 0 0 0 0
4 0 0 0 0 0 0
5 0 0 0 0 0 0
```

Figure 5. Adjacency matrix delta contains edges: 1-2, 2-3, 2-4

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-178-6

VI. PERFORMANCE OPTIMIZATION

Initial performance testing was performed on an Apple MacBook Pro with a 2.66 GHz Intel Core 2 Duo and 4 GB of memory. The code was compiled using GCC 4.2. The code was then further tested on an IBM p575 supercomputer (now replaced). The code was run under AIX 5.3 on a node with 16 1.9 GHz dual-core CPUs and 32 Gbytes of memory.

A. *Compiler optimization*

Before optimization, total execution time for a test matrix of 146 nodes, where the maximum cycle length searched is limited to 9 nodes, was 1 minute and 51 seconds. After removing assertions (such as ensuring given matrix coordinates were inside the bounds of the matrix) execution time was reduced to 1 minute 17 seconds. Using the compiler flags -funroll-loops -Os execution time was further reduced to 48 seconds. Without rewriting any code the program execution time was reduced to 43% of the original execution time.

B. *Code tuning*

The 'node_create' method allocates a block of memory for a node structure, and the malloc function is known to be time consuming. To reduce memory allocations, the node_create function, which consumes 29% of execution time, was removed from the code. This was done by replacing the node structure with a single unsigned long data type. After refactoring the total execution time for the test matrix was reduced to 19 seconds.

The code was then transferred to an IBM p575 computer at the BlueFern facility [11] and compiled to optimize the output and allow code profiling. Figure 6 shows that the **dfs** routine, split into two calls by compiler optimization, now account for the majority of CPU time consumed, and should be our next target for tuning. The **dfs** code is currently single-threading, and could benefit from a parallel implementation such as one of those surveyed by Freeman [12]. Future work will focus on tuning a parallel implementation of the network code and especially of the depth-first search routine.

VII. CONCLUSION AND FUTURE WORK

Using the methods described above, the performance of the hole discovery algorithm has been significantly improved, where this optimisation will enable the analysis of much larger spatial networks. Given it remains constrained by processing power, however, future work will also consider parallelising the code. While we can clip spatial networks to a manageable size within a GIS, there will invariably be scenarios where it would be useful to consider much larger networks that represent a larger geographical system, such as the road network of California which includes 1,965,206 nodes and 5,533,214 edges [13].

The next phase of this research is to apply this optimized algorithm to urban cycle networks. Identifying holes in urban cycle networks will aid in the determination of the best new cycle path to make while considering factors such as the origin and destinations of cyclists, existing flows, and demographics.

ACKNOWLEDGMENT

The University of Canterbury's Summer Scholarship Scheme is gratefully acknowledged.

REFERENCES

[1] Lammer, S., B. Gehlsen, and D. Helbing (2006). "Scaling laws in the spatial structure of urban road networks", Physica A 363(1) pp. 89-95

[2] Tarboton, D. G. (1996). "Fractal river networks, Horton's laws and Tounaga cyclicity". Journal of Hydrology 187: 105-117.

[3] Scellato, S., A. Noulas, R. Lambiotte, and C. Mascolo (2011). "Socio-spatial Properties of Online Location-based Social Networks". Proceedings of ICWSM 11, 329-336 .

[4] Reitsma, F. and S. Engel (2004). "Searching for 2D Spatial Network Holes". Computational Science and its Applications -ICCSA 2004 Conference, Assisi, Italy, May 14 Ð 17 2004, Proceedings, Part II: Lecture Notes in Computer Science 3044 Springer: pp. 1069-1078

[5] Chandrasekharan, N., V. S. Lakshmanan and M. Medidi (1993). "Efficient Parallel Algorithms for Finding Chordless Cycles in Graphs". Parallel Processing Letters 3(2): 165-170.

[6] Barthélemy, M (2011). "Spatial Networks". Physics Reports 499:1-101.

[7] Ravasz, E. and A. Barabasi (2003). "Hierarchical Organization in Complex Networks". Physical Review E 67(026112).

[8] Cardillo, A., S. Scelllato, V. Latora, and S. Porta (2006). "Structural properties of planar graphs of urban street patterns". Physical Review E, 73: 066107.

[9] Buhl, J., J. Gautrais, N. Reeves, R. V. Sole, S. Valverde, P. Kuntz, and G. Theraulaz (2006). "Topological patterns in street networks of self-organized urban settlements". The European Physical Journal B, 49: 513-522.

[10] http://www.analytictech.com/networks/dataentry.htm last accessed 1/10/2011

[11] http://www.bluefern.canterbury.ac.nz last accessed 1/10/2011

[12] Freeman, J. (1991). "Parallel Algorithms for Depth-First Search". University of Pennsylvania Technical Report MS-CIS-91-71.

[13] http://snap.stanford.edu/data/ last accessed 1/10/2011

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-178-6