Traversal Algorithm for Large Scale Graphs based on New Greedy Algorithm

Coen233-Computer Networks

Chinki Kumari
Kiran kumar makkineni
Richard
# Table of Contents

1. Introduction ........................................................................................................ 4  
   1.1. Objective ........................................................................................................ 4  
   1.2. Problem .......................................................................................................... 4  
   1.3. Why this project related to this class ............................................................ 4  
   1.4. Why other approach is no good .................................................................... 4  
   1.5. Why do you think your approach is better ................................................... 4  
   1.6 Statement of the problem ................................................................................ 5  
   1.7 Area or scope of investigation ....................................................................... 5  

2. Theoretical bases and Literature review ............................................................... 5  
   2.1. Definition of the problem ................................................................................ 5  
   2.2. Theoretical background of the problem ......................................................... 5  
   2.3. Related research work .................................................................................... 5  
   2.4 Advantage/disadvantage of previous research ............................................... 5  
   2.5 Solution to solve this problem ........................................................................ 6  
   2.6 Where your solution is different from others ................................................. 6  
   2.7 Why your solution is better ............................................................................ 6  

3. Hypothesis ........................................................................................................... 6  
   3.1 Positive/negative hypothesis ........................................................................... 6  

4. Methodology ........................................................................................................ 7  
   4.1. How to generate/collect input data ................................................................. 7  
   4.2. How to solve the problem ............................................................................. 7  
   4.3. How to generate output ................................................................................ 7  

5. Implementation .................................................................................................... 8  
   5.1 Code ................................................................................................................ 8  
   5.2 Design Document and Flowchart .................................................................... 8  

6. Data Analysis and discussion .............................................................................. 9  
   6.1 Output Generation ......................................................................................... 9  
   6.2 Output Analysis ............................................................................................... 9  
   6.3 Abnormal Case explanation ............................................................................ 9  
   6.4 Discussion ........................................................................................................ 9  

7. Conclusion and Recommendation ...................................................................... 9
7.1 Summary and conclusion...........................................................................................................10
7.2 Recommendations for future Studies....................................................................................10

8. Bibliography..............................................................................................................................10
9. Appendices...............................................................................................................................10
1. Introduction

1.1 Objective

In computer science and mathematics, graphs are abstract data structures that model structural relationships among objects. They are now widely used for data modeling in application domains for which identifying relationship patterns, rules, and anomalies is useful. These domains include the web graph, social networks, the Semantic Web, knowledge bases, protein-protein interaction networks, and bibliographical networks, among many others. The ever-increasing size of graph-structured data for these applications creates a critical need for scalable systems that can process large amounts of it efficiently.

1.2 What is the Problem

As the size of complex data sets from social networks, simulations, bioinformatics, and other applications increases as we move into the peta-scale and beyond, we see a need for a more efficient method for large-scale graph analysis.

1.3 Why this project is related to this class

This project is related to class as it deals with the following-
   a) Nodes which are randomly distributed in the network, with source and destination.
   b) Nodes need to communicate with each other.
   c) Wireless sensor networks as in P2.

1.4 Why other approach is no good

Although there have been several large-scale graph processing algorithms that both leverage innovative algorithms and advancements in hardware technology (such as massive parallel processing units like GPU’s) there is often a better solution to a problem within the context of the problem itself. In the case of the problem illustrated in P2 (the wireless sensor array), there are certain greedy choices we may apply to the problem at hand to possibly expedite the computation of a path to an access point.

1.5 Why do you think your approach is better

Our Algorithm will build upon established principles of graph traversal algorithms, while using our domain knowledge of the problem at hand to possibly improve the throughput of the problem. Please keep in mind we are only dealing with the graph traversal aspect that was touched on in P2, not the energy conservation algorithm.

1.6 Statement of the Problem
In our project, we plan to find an algorithm to get an efficient path in the traversal of a graph to pre-specified “end nodes” (in the case of P2, the access points). The edges of the graph are determined by the Euclidean distance between each individual node and are un-weighted (minimum node jumps). We also assume that it is a wireless sensor network that has been randomly distributed (i.e. randomly scattered over an area and sparse).

1.7 Area or Scope of Investigation

Analyze large and dynamic graphs
Find a shortest path from source to destination

2. Theoretical bases and Literature review

2.1. Definition of the Problem

Growing research interest in using large-scale graphs to analyze complex data sets from social networks, simulations, bioinformatics, and other applications. As the size of these data sets increases as we move into the peta-scale and beyond, we see a need for a more efficient method for large-scale graph analysis.

2.2 theoretical background of the problem

Sources of real-world large graphs include:
- Social graphs (Facebook, Twitter, Google+, LinkedIn, etc.)
- Endorsement graphs (web link graph, paper citation graph, etc.)
- Location graphs (map, power grid, telephone network, etc.)
- Co-occurrence graphs (term-document bipartite, click-through bipartite, etc.)

These graphs have common characteristics. The first is their large scale. For example, by January 2011, Facebook had about 600 million nodes; major search engines have indexed tens of billions of webpages over a trillion nodes. Second, these graphs are sparse, meaning the number of edges at one vertex is far less than the total number of vertices. The third characteristic of the graphs is rich information on nodes and edges, and we expect the graphs of interest in this project will have substantial information associated with vertices and/or edges. For example, each node in Facebook can have attributes such as age, gender, interests, etc. and each edge in Facebook can have attributes such as creation time, type of relation, communication frequency, etc. As the size of these data sets increases to $10^{15}$ floating operation/sec and beyond. Drawing upon the inspiration of the wealth of information contained in each node, our algorithm will attempt to bestow information in the nodes upon formation of the graph and then leverage that in the final computation.

2.3 Related research to solve the problem
There is a wealth of research in this area as this is a very important problem in computer science. Much contemporary research has currently been focused on processing graph traversal using massive parallel processing units (e.g. GPU or massive distributed systems), but we feel that an algorithm based approach can also yield very satisfactory results.

2.4 Advantage/disadvantage of those research

The advantage of our solution is that we are using very specific domain knowledge of the nuances of the problem to create a better algorithm. There are two big problems, one we do not know if all assumptions can realistically be met, and secondly even if every assumption is met, our solution to the problem only functions in the context of the problem at hand, it be hard to apply it to other similar but different problems.

2.5 Your solution to solve this problem

We will sort the nodes by their Euclidean distance from the access points before constructing the graph itself (O(nlogn) operation). Then during graph construction, we will add the nodes in order based on their Euclidean distance from access points taking note on the number of jumps for the initial nodes (first greedy choice here: not all closest nodes after the first will have a node connected to the access point) and then simply adding on to each node connected to it (second greedy choice here to avoid a second O(nlogn) sort of the edges of each node, we will check edges based on the sorted order, i.e. instead of an O(n^3 * logn) graph construction process, we do a O(n^2) step which can contain some nodes that fail). Then given the jump data in each node, we have a very clear guideline for how to traverse the graph.

2.6 Where your solution different from others

Other solutions are either more broad than our solution (i.e. provides the general structure for a solution of a graph traversal problem), or just as specific but not applicable to our problem.

2.7 Why your solution is better

Our solution is better because we are leveraging specific domain knowledge of our problem.

3. Hypothesis (or goals)

3.1 Positive/negative hypothesis

We believe that our method of traversing the graph will be more efficient than the breadth first search. Not only that, but besides the minor overhead that is needed anyway, it will make subsequent searches even faster.

4. Methodology
4.1 How to generate/collect input data

We will simulate the graph traversal on the computer using both the breadth first search and the modified greedy algorithm described and compares the speed of each approach.

4.2 How to solve the problem

We are using a new greedy algorithm to improve traversal of an un-weighted undirected graph that is randomly distributed to specific access points.

4.3 Algorithm design

Sort the nodes by Euclidean distance then construct a graph that keeps number of jumps to the access point stored in each node. When traversing the graph, use the jump data stored in each node to find a faster solution.

4.4 Language used

C++

4.5 Tools used

Visual Studio 2010

4.6 How to generate output

Our output will be generated by running the simulations on our program. We will either use a console display if we are constrained on time or, if we have the time, we will give a graphical display of the results.

4.7 How to test against hypothesis

We will compare the outputs of the two methods (i.e. breadth first search and our modified greedy algorithm) and see which one is faster.

4.8 How to prove correctness

Test with program.

5. Implementation

5.1 Code: attached
5.2 Design document and flowchart

Flow chart of graph traversal based on hop count
Breadth first Search.
The $O(n!)$ where $n = \# \text{ of nodes } O(|E| + |V|) = O(b^d)$, space complexity is the same.

Our algorithm based on hop count:
Worst case: same as BFS
Average case: $O(n^2)$
When constructing graph, add nodes in order of Euclidean distance to access points $O(n\log n)$ time.
Then, compute each hop distance as they are added to the graph. Find the shortest hop node each step.

6. Data analysis and discussion

6.1 Output generation

We run the program against various sets of test data to generate the output with which to test our hypothesis against.

6.2 Output analysis

Based on even a rudimentary look at the output, it is quite clear that our algorithm is significantly better than a breadth first search in extremely dense graphs, that is, even in relatively small dense graphs, breadth first search will never find a solution and is very memory intensive, whereas our algorithm finds solutions instantaneously despite a slightly larger overhead in the initial graph construction.

6.3 Abnormal case explanation

The worst case scenario for our algorithm would leave it at the same speed as breadth first search, it cannot be slower as in the worst case it is just a breadth first search.

6.4 Discussion

Although it is significantly better than breadth first search, that isn’t to say that a BFS is just completely unviable as a graph traversal algorithm. Firstly, BFS is embarrassingly parallel, which makes it very important in terms of research in the field of graph traversal utilizing massive multiprocessing. Secondly, the BFS we used was the most brute force possible, there are ways we can better optimize it. Finally, there are situations where our algorithm is not practically feasible, i.e. the initial overhead is never that bad, but there are real world situations where it simply never has a chance to compute it.

7. Conclusion and Recommendation
7.1 Summary and Conclusion
In computer science, graph traversal is the problem of visiting all the nodes in the graph, in a particular manner, updating and or checking their values along the way. Unlike tree traversal, graph traversal may require that some nodes be visited more than once, since it is not necessarily known before transitioning to a node that it has already been explored. As graphs become more dense, this redundancy becomes more prevalent, causing computation time to increase; as graphs become more sparse, the opposite holds true. If each node in a graph is to be traversed by a tree-based algorithm (such as DFS or BFS), then the algorithm must be called at least once for each entirely distinct sub graph of the graph. This is easily accomplished by iterating through all the nodes of the graph, performing the algorithm on each node that is still unvisited when examined. In our case we use a hop count to traverse the graph. And in extremely dense situation our algorithm produces very good results, where the breadth first search fails to give any results.

7.2 Recommendations for future studies

There is still much work to be done in the field of graph traversal and there are many ways we can approach the problem. As we have already mathematically proven certain problems to be too complex (i.e. NP) for conventional computing, we must tackle them with either a heuristic approach or from the perspective of leveraging emerging technologies such as massive distributed computation and quantum machines.

8. Bibliography
http://en.wikipedia.org/wiki/Graph_traversal
http://www.systap.com/pubs/xdata/Large-Scale-Graph-Processing-Algorithms-on-the-GPU.pdf
https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/doern/GT.pdf
http://www.cs.uah.edu/~rcoleman/Common/C_Reference/C++For%20DUMMIES.pdf

9. Appendices

Source Code
Readme file