Shortest Path Computation: A Comparative Analysis

Ross Sherlock, Peter Mooney and Adam Winstanley
Department of Computer Science, National University of Ireland Maynooth (NUIM),
Co. Kildare, Ireland.
Email: `{se401021,pmooney, adamw}@cs.may.ie`

Jan Husdal
Department of Geography,University of Utah,
Salt Lake City, UT 84112-9155 USA.
Email: `jan.husdal@csbs.utah.edu`

ABSTRACT

Given a network structured data set (N spatially embedded nodes (x_i, y_j) and M edges, each edge E_k mapped to C edge costs) several software options are available allowing one to compute shortest paths between nodes in this data set. Software options include powerful GIS software engines such as ArcView and MapInfo, high level programming language implementations such as C++, Java or LISP and mathematical software such as Matlab and Mathematica. All software options find the same shortest, optimal, paths given the same input criteria. However one may differentiate between each software option based on the relative importance of a small number of criterion. Evaluation criterion can be summarised as software license cost, implementation time (software development hours), implementation cost, underlying data structures, shortest path query execution time (and hence query response time), etc. Current research work into establishing a performance efficiency hierarchy between Java, C++ and ArcView is described. Experimentation will be performed in order to statistically compare shortest path query execution time, response time and implementation issues of ArcView, C++ and Java performing shortest path computations on identical data sets and identical input parameters. This research will provide useful experimental results to GIS researchers uncertain of which implementation best matches their particular shortest path computation needs.

INTRODUCTION

Computation of shortest paths is a famous and classical area of research in Computer Science, Operations Research and GIS. There are many ways to calculate shortest paths depending on the type of network and problem specification. The literature on shortest paths is immense with considerable literature exploring how different software implementations perform against each other given identical data sets and identical input parameters. We have chosen two high level programming languages (Java and C++) and the leading GIS software package ArcView to perform this analysis. The literature is sparse in regard to high level programming implementations performing against GIS engines such as ArcView. We are

JAVA AND C++

The Java programming language is an object-oriented, general-purpose language featuring simple object semantics, cross-platform portability, arbitrarily shaped arrays and security. These features come with a cost: they negatively impact the performance of Java applications. In the literature Java has been compared less than favourably to the performance of C++ and other high-level native code language. Recently a combination of faster virtual machines and optimisation of compilers have resulted in Java performance very close to that of C++ and other platform dependent languages. A Java program only requires a few kilobytes of disk space (and similar amounts of RAM) with no need to download or install unnecessary libraries not used by that program. C++ and Visual Basic applications, on the other hand, require at minimum a run-time library of all the standard functions that those languages can call. The loading time of the executable programs is not of concern to us. All programs are located on a disk drive local to the target platform and as a result executable program size does not become a limiting, or influential, factor in the performance of the programs. (Carmine [3]) The only significant performance difference between a Java program run with a JIT and a native C++ application will be the amount of time required to perform the initial translation of the class file and the types of optimisations that are performed. Carmine [3] concludes that in theory and practice there is rarely any significant performance difference between native C++ and Java applications.

EXPERIMENTAL ANALYSIS DETAILS

To accurately assess the performance of ArcView, C++ and Java robust test suites have been developed. In order to avoid writing special Avenue scripts to process C++ or Java dependent input files it has been decided to exclusively use ShapeFiles (.shp) and DBF (.dbf) files as input. C++ Input Output (IO) functions and Java IO methods are easily developed to process ArcView input files. DBF files are easily parsed to construct the graph data structure model G = (V,E) by extracting Node and Edge objects from the DBF file data. The network structured data used is data freely available transportation data such as that available directly from the ESRI web-site (http://www.esri.com/data/index.html) and TIGER 2000 Census data. This provides robust real world data while avoiding the overhead of generating large randomly structured graph input sets.

The correct choice of data structure for any algorithm is critical [5]. Our tests analyse the efficiency of different types of data structures written in C++ and Java when implementing Dijkstra's algorithm. Dijkstra’s algorithm’s original strategy (outlined in Dijkstra [2]) uses a queue data structure. In the basic Dijkstra's algorithm (Dijkstra) the complexity bottleneck is node selection. Enhanced versions of Dijkstra's algorithm, rather than scanning all temporarily labelled nodes at every iteration to find the minimum distance label, computation time is reduced by maintaining distances in sorted order. Heap (or priority queue) data structures (Ahuja [1]) are implemented in modified Dijkstra algorithms. ArcView uses a modified Dijkstra algorithm implemented using d-Heap's (Graham [8] Weiss [9]) with d = 2. ArcView's Network Analyst implements a modified Dijkstra with a d-heap in combination with a custom memory management strategy to deal with very large networks. The network representation of the spatial data is a proprietary data structure that was built for ArcView Network Analyst to facilitate quick access to the topology of the spatial data.

However Fibonacci heaps offer a better implementation for Dijkstra's algorithm. The Fibonacci time bound is consistently better than that of all other heap implementations for all network densities. It is also currently the best strongly polynomial-time algorithm for solving the shortest path problem (Ahuja [1]). The table below outlines how ArcView is tested against various implementations of Java and C++ versions of Dijkstra's algorithm. The version of Dijkstra's algorithm is outlined in each case. Tests will be carried out on sparse and dense networks.

COMPLEXITY	JAVA	C++	ArcView
O(n)	Linear Search	Linear Search	2-Heap
O(m log n)	Binary Heap	Binary Heap	2-Heap
O(m + n log n)	Fibonacci Heap	Fibonacci Heap	2-Heap

CONCLUSIONS

Test suites for the experiments outlined above have been devised. This research aims to provide an insight into four research issues. Firstly we will establish some type of performance hierarchy between Java, C++ and ArcView in terms of query execution times, data structure initialisation times etc. Secondly this research provides an interesting, and useful, comparison between Java and C++ performing a real world task (shortest path route finding) on various platforms. Thirdly this research highlights how Fibonacci Heaps (implemented in C++ and Java) perform against the d-Heap implementations of ArcView suggesting how Fibonacci Heaps may be implemented as a viable and practical ArcView data structure alternative to d-Heaps. Finally we have addressed the performance and implementation issues of naïve implementation strategies (Java and C++) perform against a powerful GIS engine like ArcView for large input data sets.

BIOGRAPHIES

Ross Sherlock is completing the final year of a Computer Science and Software Engineering honours degree at NUIM. This research work (when completed) will be submitted as his final year dissertation. Peter Mooney BSc (Ph.D Candidate) and Dr. Adam Winstanley are his dissertation supervisors. This work is part of ongoing research work on network algorithms in the Intelligent Graphical Data Research Group at NUIM. Jan Husdal holds an MSc in GIS from the University of Leicester, UK and is a second-year PhD student and a Research Assistant at the University of Utah, Salt Lake City, USA.

REFERENCES

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993

[2] E.W. Dijkstra. *A note on two problems in connexion with graphs*. Numerische Mathematik, 1:269--271, 1959

[3] Mangione Carmine, Performance tests show Java as fast as C++ JavaWorld February 1998 http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf.html

[4] Michael L. Fredman and Robert Endre Tarjan. *Fibonacci heaps and their uses in improved network optimization algorithms*. Journal of the ACM, 34(3):596--615, 1987.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[6] Y. Ding and M. A. Weiss, "The k-d Heap: An Efficient Multi-Dimensional Priority Queue (extended abstract)," Workshop on Algorithms and Data Structures, Montreal, CA, August 1993, Springer-Verlag Lecture Notes #709, 303-314.

[7] Graham I.G. Lecture 11 - Heaps (Priority Queues) http://www.onthenet.com.au/~grahamis/int2008/week11/lect11.html

[8] M A Weiss "Data Structures and Algorithm Analysis in C", Chapter 6 and Section 7.5 Addison-Wesley (Second Edition) 1997