# Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 11 (12): 2255-2259, 2011
ISSN 1812-5654 / DOI: 10.3923/jas.2011.2255.2259


# Dijkstra Algorithm Heuristic Approach for Large Graph

[1]Sahar Idwan and [2]Wael Etaiwi
[1]Department Computer Science and Application, Hashemite University,
P.O. Box 150459, Zarqa 13115, Jordan
[2]Department of Computer Science and Application,
Al-Balqa' Applied University, Al-Salt, Jordan

**Abstract:** In this study, we developed an algorithm that improves a Dijkstra implementation in a large and dense graph based on hMetis partitioning. hMetis is a hyper graph partitioning algorithm that divides the massive graph into sub-graphs. The algorithm is called h-Dijkstra consists of pre-processing the flat graph by partitioning it onto a sub graph then building a two-level hierarchal graph. The shortest path algorithm computes on the top-level graph. Our experimental results show the dominance of our algorithm over the traditional Dijkstra algorithm and other alternative solutions based on the time and the number of Input/Output (I/O) operations.

**Key words:** h-Dijkstra, Dijkstra, hMetis, graph, hierarchal graph, RAM

## INTRODUCTION

Dijkstra algorithm computes a Single Source Shortest Path (SSSP) in weighted graphs with n nodes and m edges from a node to all others (Cormen *et al*., 2001). Goldberg (2001a, b) presented the algorithms for the single-source shortest path problem. Their labeling algorithm based on multi-level bucket shortest path algorithm that used to determine the length of the edges. We also can use the Breadth First search algorithm to compute the shortest paths for unweighted graphs. It is easy to apply the Dijkstra on the small graph since it fits in the memory, as for large graph, Dijkstra poses many challenges related with its storage in the memory and the I/O operations. Different algorithms are introduced to overcome the difficulties related with computing the shortest path using Dijkstra on the massive graph based on partitioning the graph into a set of fragments that fit in the memory.

This study introduces an acceleration algorithm for enhancing Dijkstra computation (h-Dijkstra) on a large dense graph based on the RAM model. The main idea is dividing the large graph into the set of sub-graphs that fit in the memory by using hMetis. The partition techniques reduce the number of crossing edges between the sub-graphs, which reduce the number of I/O operations. Then we build the two-level hierarchical graph that used to compute the shortest path.

Chan and Zhang (2001), Jing *et al*. (1998), Jing *et al*. (1996), Huang *et al*. (1996) and Shekhar *et al*. (1997) presented a disk based shortest path algorithms by partitioning the graph into a set of square-like blocks depends on spatial proximity. Jing *et al*. (1998) proposed building the hierarchal graph from the flat graph based on fragmentation, they pushed all border nodes, that nodes belong to more than one fragment to generate the next level. Another approach proposed by Huang *et al*. (1996) is to sort all edges by the x coordinates of their source nodes and then applies a plane sweep technique to sweep all x-sorted edges from left to right. The sweeping process stops periodically to sort the edges swept since the last storage by the y-coordinates of their origin nodes. This technique is applicable only if coordinates are available for the vertices of the graph and usually yield partitions that are worse than those partitioned by other partitioning techniques.

Huang *et al*. (1995), Houtsma *et al*. (1991), Jung and Pramanik (1996) and Shekhar *et al*. (1993) proposed hierarchal graph models but they are not guaranteeing the optimality of the retrieved path.

Gutman (2004), Meyer (2001) and Schulz *et al*. (2002) are presented the hierarchical decomposition based on the geometric information. Gutman (2004) computed the shortest path by pruning the nodes that are far a way from the possible shortest path based on the "reach" information. This information computed from the original

**Corresponding Author:** Sahar Idwan, Department of Computer Science and Application, Hashemite University, P.O. Box 150459, Zarqa 13115, Jordan Tel: +96253903333 Fax: +96253826613

graph and it represents the importance of the node. The reach value and the Euclidean information associated with each node. His work is time consuming in the preprocessing phase and need more assumptions about the input.

Meyer (2001) presented a complicated algorithm to compute the shortest path algorithm for edges drawn from a uniform distribution with a linear complexity.

Schulz *et al*. (2002) built a multi-level graphs by adding one level of edges to the original graph based on different criteria for including vertices on the subsets such as the selection of the importance of the station, highest degrees, or random choice. Their work need a lot of investigation about the best choice of subsets and the size to determine the best values for the number of levels which could lead sometimes to the bad choice.

Our proposed algorithm guarantees the optimality in computing the shortest path using the Dijkstra algorithm on a large graph. It consists of two phases: the pre-processing phase and the computation phase. The main idea is to divide the massive graph into smaller sub-graphs by using hMetis that guarantee the number of nodes in each partitioning is equal and minimizes the number of crossing edges. We build the two-level hierarchal graph that contains the border nodes and the source and the target nodes. Then apply the shortest path algorithm on the top-level graph.

We run our algorithm using a grid data with different number of nodes and compared it with the Dijkstra's algorithm. We also compared our results with the HEPV (Hierarchal Encoded Path View) and FEPV (Flat Encoded Path View) presented by Jing *et al*. (1996) that use different approaches of partitioning.

## IMPROVED DIJKSTRA ALGORITHM IMPLEMENTATION IN LARGE GRAPH (h-Dijkstra)

Here, we addresses the acceleration algorithm (h-Dijkstra) for finding the shortest path of a weighted massive graph. Our approach consists of pre-processing the original (large) graph by partitioning it into sub-graphs using hMetis. Then we build a two-level hierarchal graph.

**Graph partitioning:** We used the hMetis software package that was developed for partitioning the VLSI circuits by Karypis and Kumar (1998) and Karypis *et al.* (1997) to partition the original graph. Graph partitioning is NP-complete and consequently the VLSI design automation community has explored several approaches for practical graph partitioning. In a traditional graph, each edge connects two vertices. hMetis permits to specify a parameter K, which represents the desired number of fragments. It partitions the graph into K fragments such that each fragment contains approximately the same number of vertices, while minimizing the number of edges that cross the fragments. Figure 1 illustrates the graph that has been partitioned. There are other methods to partition the graph that depends on spatial proximity.

**Compute the shortest path by using h-Dijkstra:** Our algorithm contains two steps: The first step is the pre-processing step that uses hMetis. Then we build a two-level hierarchal graph that contains the source and the target nodes. The hMetis algorithm partitions the vertices of a graph into fragments. The edges of the graph are of two types: local and border. A local edge is one whose vertices belong to the same fragment, whereas a border edge is one both vertices of which belong to different fragments. Two fragments are said to be adjacent if they have edges with endpoints belonging to different fragments. The endpoints of the border edges are border nodes other nodes are local nodes. The pre-processing step considers specifying the type of each node in the fragment.

We build the two-level hierarchical graph from the flat graph by inserting a border node at the half point of each border edge. Figure 2 illustrates these concepts. Two fragments are said to be adjacent if they have at least one

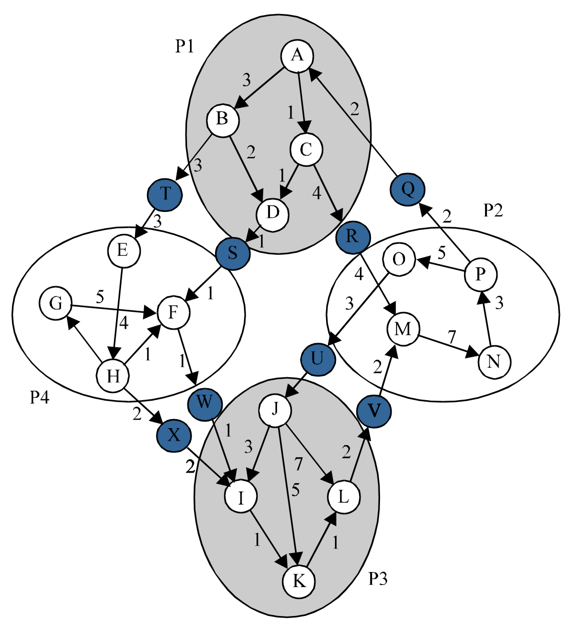


Fig. 1: Partitioning the graph that contains 16 vertices into four fragments consisting of four vertices each

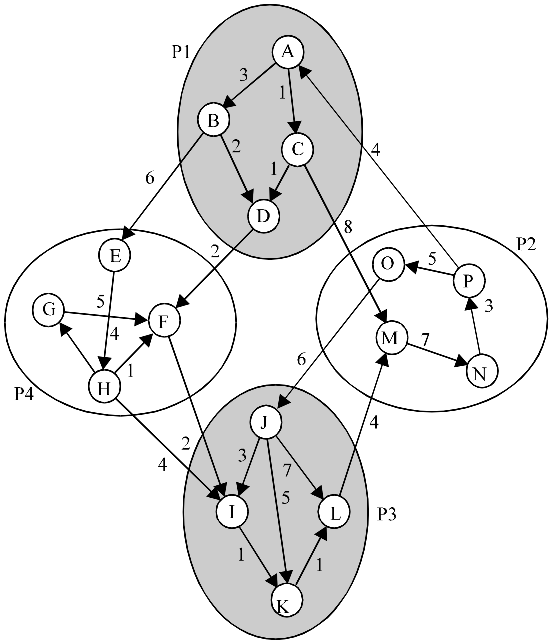


Fig. 2: Local, border and extra nodes

common border node. For example, P1 and P2 are adjacent, but P1 and P3 are not.

We moved up the border nodes to the top-level, where they form the vertices of the top-level graph Fig. 3. Edges added to the top-level graph so if the shortest path distance between a pair of border nodes in the original flat graph is some quantity Z, then there is a shortest path of length Z at the top-level graph. The algorithm of building the hierarchal graph is presented on Algorithm 1.

Algorithm 1: Building the hierarchal graph

**Generate Hierarchical Graph(G)**

- Fragment list= hMetis(G)
- For each fragment F in fragment list do
  - For each border node ∊ F
  - sPath=compute the shortest path by using the edges in the flat graph

Having created a hierarchal graph, we are ready to solve the point-to-point shortest path computation. We compute the shortest path between the source and the target by using the top-level graph. An edge is added from the source to the border nodes adjacent to the fragment that contains the source node. The length of the edge is the length of the shortest path between the source and the border nodes in the flat graph. Similarly, an edge is added between the border nodes adjacent to the fragment that contains the target and the target node. The shortest path between the source and target is presented in Fig. 4. The algorithm is presented in Algorithm 2.

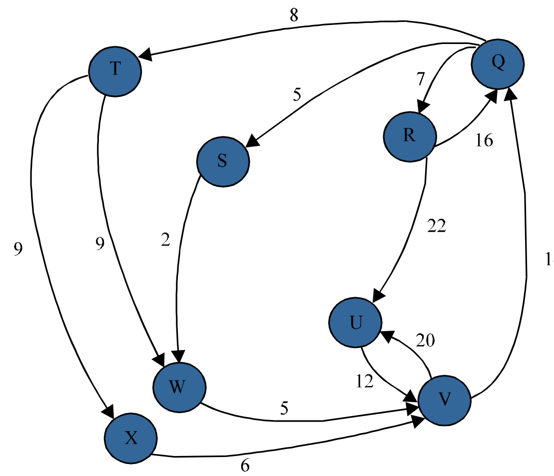


Fig. 3: Hierarchal graph

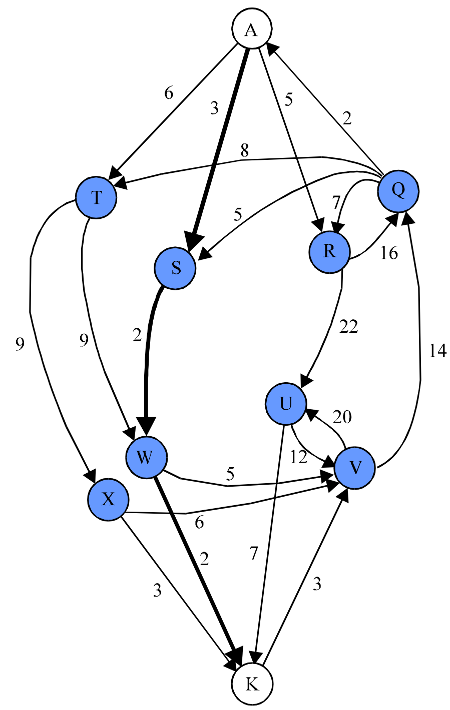


Fig. 4: Illustration of computing the shortest path in the top-level graph. Vertices A and K denote the source and the target nodes, respectively. Edges (A,T) (A,S) (A,R) (Q,A) (X,K) (W,K) (U,K) and (K,V) are added to the top-level graph. The lengths of these edges are the lengths of the shortest path between the vertices in the original graph. The shortest path from A to K computed by using the Dijkstra's algorithm The Shortest path is A-S-W-K

Table 1: Results on Grid with different partitioning

| | | | h-Dijkstra | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Dijkstra | | 4 Partitions | | 8 Partitions | | 16 Partitions | |
| Rows×Columns | Time (sec) | # (I/O) | Time (sec) | # I/O | Time (sec) | # I/O | Time (sec) | # I/O |
| 40×40 | 0.23 | 1450 | 0.031 | 236 | 0.032 | 397 | 0.033 | 550 |
| 100×100 | 3.80 | 11349 | 0.998 | 744 | 0.904 | 1474 | 1.032 | 2234 |
| 150×150 | 13.20 | 24764 | 4.945 | 1552 | 4.908 | 3189 | 4.618 | 4135 |
| 200×200 | 56.10 | 42543 | 15.319 | 2812 | 14.228 | 4008 | 14.57 | 5210 |

# I/O = No. of input/output operations

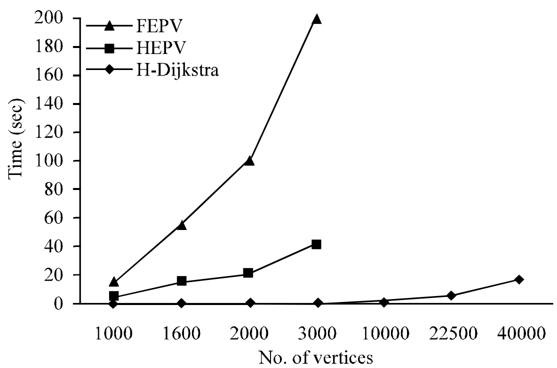


Fig. 5: Computing the shortest path by using h-Dijkstra vs HEPV and FPEV

Algorithm 2: Computing the shortest path by using h-Dijkstra

**H-Dijkstra(source, destination)**

- Add edge from the source to the border node belong to source fragment: Length of the edge= Shortest path (source, border nodes)
- Add edge from the border node belong to the target fragment to the target node: Length of the edge= Shortest path (border nodes, target)
- Spath=Shortest path(HG, source, destination)// HG: hierarchal Graph

## IMPLEMENTATION AND EXPERIMENTAL RESULTS

Our algorithm was developed in Visual C++® 6.0 Enterprise Edition in (Malik, 2006) and (Deitel and Deitel, 2005) were implemented on an Intel® Pentium® M with speed 1.73 GHz processor and 504 MB RAM, running on the Microsoft Windows XP Professional Service Pack 2. In all experiments, the results contain the following:

- Running time
- No. of I/O operations
- No. of partitioning

We ran our experiments on 40×40, 100×100, 150×150 and 200×200 directed weighted grids graph (this models the road networks structure in urban areas). Table 1 shows these results. We make the following observations about the results of our experiments:

- Table 1 show that our h-Dijkstra are significantly superior to the Dijkstra under either measure the run time and number of I/O operations. The Dijkstra takes longer time due to the large number of I/O operations for the massive graph that doesn't fit in the memory
- The number of I/O operations increase as we increase the number of partitions but still less than the number of I/O of Dijkstra

Figure 5 compared h-Dijkstra with the HEPV (Hierarchal Encoded Path View) and FEPV (Flat Encoded Path View) presented (Jing *et al*., 1996) that use different approaches of partitioning the massive graph. We found that superiority of h-Dijkstra approach over the other alternative solutions.

## CONCLUSION

The contribution of this study is to show that when we partitioned the large graph into fragments by using hMetis where the number of nodes in each fragment is equal and reduces the number of the crossing edges and computed point-to-point shortest path computation at the top level, h-Dijkstra has several advantages over Dijkstra, HEPV and FPEV. The experimental results show that we enhance the running time and reduce the number of I/O operations. Our future work includes the deployment of the h-Dijkstra on a sparse graph and introduces a new parallel version of h-Dijkstra.

## REFERENCES

Chan, E.P.F. and N. Zhang, 2001. Finding shortest paths in large network systems. Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems, Nov. 05-10, ACM, Atlanta, GA, USA., pp: 160-166.

Cormen, T.H., C.E. Leiserson, R.L. Rivest and C. Stein, 2001. Introduction to Algorithms, Section 24.3: Dijkstra = s Algorithm. 2nd Edn., MIT Press, McGraw-Hill, New York.

Deitel, H. and P. Deitel, 2005. C++ How to Program. 5th Edn., Prentice Hall, New Jersey, USA., ISBN-10: 0131857576.

Goldberg, A., 2001a. Shortest path algorithms: Engineering aspects. Algorithms Comput., 2223: 502-513.

Goldberg, A.V., 2001b. A simple shortest path algorithm with linear average time. Algorithms ESA, 2161: 230-241.

Gutman, R., 2004. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. Proceedings 6th International Workshop on Algorithm Engineering and Experiments (IWAEE=04), New York, pp: 100-111.

Houtsma, M.A.W., F. Cacace and S. Ceri, 1991. Parallel hierarchical evaluation of transitive closure queries. Proceedings of the 1st International Conference on Parallel and Distributed Information Systems (PDIS), Dec. 4-6, IEEE Computer Society Press, Miami Beach, Florida, USA., pp: 130-137.

Huang, Y.W., N. Jing and E.A. Rundensteiner, 1995. Hierarchal path views: A model based on fragmentation and transportation road types. Proceedings of the 3rd ACM Workshop on Geographic Information Systems, Dec. 1-2, ACM Press, pp: 93-100.

Huang, Y.W., N. Jing and E.A. Rundensteiner, 1996. Effective graph clustering for path queries in digital map databases. Proceedings of the 5th International Conference on Information and Knowledge Management, Nov. 12-16, ACM, Rockville, Maryland, USA., pp: 215-222.

Jing, N., Y.W. Huang and E.A. Rundensteiner, 1996. Hierarchical optimization of optimal path finding for transportation applications. Proceedings of the 5th International Conference on Information and Knowledge Management, Nov. 12-16, Rockville, MD, USA., pp: 261-268.

Jing, N., Y.W. Huang and E.A. Rundensteiner, 1998. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. IEEE Trans. Knowledge Data Eng., 10: 409-432.

Jung, S. and S. Pramanik, 1996. HiTl graph model of topological road maps in navigation systems. Proceedings of the 12th International Conference on Data Engineering, pp: 76-84. http://portal.acm.org/citation.cfm?id=655601 and CFID=1527816 and CFTOKEN=64673181.

Karypis, G. and V. Kumar, 1998. hMETIS: A Hypergraph Partitioning Package, Version 1.5.3. University of Minnesota, St. Paul, USA..

Karypis, G. and V. Kumar, 1999. MultiLevel k-way hypergraph partitioning. Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, June 21-25, New Orleans, LA, USA., pp: 343-348.

Karypis, G., R. Aggarwal, V. Kumar and S. Shekhar, 1997. MultiLevel hypergraph partitioning: Application in Vlsi domain. Proceedings of the 34th Annual Conference on Design Automation (DAC=97), ACM Press, New York, USA., pp: 526-529.

Malik, D.S., 2006. C++ Programming: From Problem Analysis to Program Design. 3rd Edn., Thomson Course Technology, Boston, MA, ISBN-13: 9781418836399.

Meyer, U., 2001. Single-source shortest paths on arbitrary directed graphs in linear average time. 12th, ACM-SIAM Symposium on Discrete Algorithms, pages 797-806, http://portal.acm.org/citation.cfm?id=365784.

Schulz, F., D. Wagner and C. Zaroliagis, 2002. Using multi-level graphs for timetable information railway systems. Algorithm Eng. Exp., 2409: 43-59.

Shekhar, S., A. Fetterer and B. Goyal, 1997. Materialization trade-offs in hierarchical shortest path algorithms. Adv. Spatial Databases, 1262: 94-111.

Shekhar, S., A. Kohli and M. Coyle, 1993. Path computation algorithms for advanced traveler information system. Proceedings of the 9th International Conference on Data Engineering, April 19-23, Vienna, pp: 31-39.