Vol. 7(26), Oct. 2017, PP. 3665-3673

Contents list available at IJMEC

International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)

Journal Homepage: www.aeuso.org

PISSN: 2411-6173 - EISSN: 2305-0543

A Review of Acceleration and Parallelization Methods and Techniques for Finding the Shortest Path on Graphs

Havva Alizadeh Noughabi[1], Farzaneh Ghayour Baghbani[2]

[1]*Department of Computer Engineering, University of Gonabad, Iran*

[2] *Department of Computer Engineering, University of Tehran, Iran*

Phone Number: *+98-9158798606*

****Corresponding Author's E-mail:*** *h.alizadeh@gonabad.ac.ir*

Abstract

Due to the wide use of the problem of finding the shortest path in different fields of study, many attempts have been done to provide solutions to increase the responsiveness of shortest path queries. These methods are known as acceleration methods and most of them use parallel processing. In this paper, at first, the problem of the shortest path in graph theory and the related works are reviewed. The shortest path problem in computational geometry will be examined in two-dimensional and three-dimensional space. Then, methods and techniques used to accelerate and parallelize the algorithm of the shortest path on the graph will be classified and described in 10 sections. The 10 methods include partitioning, separators, hierarchical, marking the edge, routing based on access, geometric maintenance, bilateral search, search to the target, important vertices and delta step. Each method along with its time and space complexities will be explained and analyzed.

Keywords: *The shortest path on graph, The shortest path algorithm acceleration techniques, The shortest path algorithm, parallel methods.*

1. Introduction

The problem of the shortest path on graphs, due to its applications, has been studied a lot. On the other hand, in most real applications, the size of the graph is relatively large and the running time of algorithms such as Dijkstra is not acceptable. Hence, many efforts have been done to speed up response to queries of the shortest path. These methods that are known as acceleration methods, often use parallel processing. In these methods, by spending time to preprocess and store the results in memory, query time is reduced. The most ideal way to reduce query time is to calculate the shortest path between all pairs of vertices and save it in the memory. Accordingly, the query time will be O(1), but this method requires a memory of $O(n^2)$ and for graphs of real applications is not practical. In section 2, the problem of the shortest path, i.e. graph theory and computational geometry will be introduced in two parts. Then, in Section 3, methods and techniques employed for acceleration and parallelization of algorithms of the shortest path on graphs will be classified and described in 10 sections. Finally, in Section 4, the conclusion is presented.

Article History:

Received Date: Jun. 28, 2017
Accepted Date: Aug. 16, 2017
Available Online: Oct. 01, 2017

2. The Introduction of the Shortest Path Problem

2.1. The Shortest Path in Graphs Theory

Dijkstra's algorithm calculates the shortest path from a single source [1]. Depending on its implementation, Dijkstra's algorithm has the time order $O(v^2)$, $O(e \log v)$ and $O(e + \log v)$ where v is the number of vertices and e is the number of edges in the graph. For the special case of flat graphs with non-negative edges in 1997, Henzinger et al. put forth a linear algorithm with Dijkstra's algorithm as its basis [2]. This algorithm achieved the linear time order by repeating zoning, using separators. Also in 1997, Throup presented an algorithm from the order $O(e)$ for general graphs with integer edge weights. The basis of this algorithm is also similar to Dijkstra with the exception that the order of meeting vertices is different [3].

The problem of finding the shortest path between all pairs of vertices has also attracted a lot of attention. The most famous algorithm for this algorithm is Floyd-Warshall's with the running time of $O(v^3)$ [4]. A more efficient algorithm is Johnson's algorithm with the time order of $O(v^2 \log v + ve)$ [5]. Chan (2006) introduced an algorithm from the order of $O(ve)$ for graphs without direction and weight [6].

All the shortest path algorithms presented can be classified in two main categories, namely label setting and label correcting. In label setting methods such as Dijkstra, each vertex has a label, which is initially infinite and represents a high rate of the shortest path from the source to that vertex. Vertices are divided into three categories: not met, waiting and calculated. In these methods, updating the labels can be done only from an edge that one of its ends is calculated, and during the process of the algorithm, the vertices one after another receive their final label gradually. However, in label correcting methods such as Bellmanford, updating the label of each vertex can be continued to the end of the algorithm.

2.2. The Shortest Path in Computational Geometry

The shortest path in two-dimensional space: The simplest form of the problem of finding the shortest path in computational geometry is to find the shortest path in two-dimensional spaces. In the two-dimensional weightless state, there are several obstacles on the page that are often polygon and the goal is to find the shortest path between two points on the surface so that the path does not meet the obstacles. In weighted state, the page is divided into polygons that the cost of passing from unit of length for each area is expressed as the weight of that area. For obstacles, extreme weight is considered. For all kinds of the two-dimensional problem where areas (or barriers) are in the form of polygons, problem is easily reduced to the shortest path problem in graphs. Therefore, the optimal path may change direction only in polygon parts and with the forming of the graph of vision corresponding points, the vertices of polygons are considered as the vertices of the graph, and then between vertices with a straight line between them which do not meet an obstacle, an edge is drawn. Different algorithms of constructing a vision graph exist. The simplest of them has a time order of $O(v^2 \log v)$. An algorithm with the time order of $O(v^2)$ was presented by Welz [7]. There are output-sensitive algorithms such as the algorithms of Pocchiola and Vetger with the time order of $O(v + v \log v)$ and memory of $O(v)$ [8].

In the method of vision graph, the length of the shortest path may be much smaller than the size of the constructed vision graph and we might face the additional cost of the formation of vision graph. Note that the size of the vision graph is from the order $O(v^2)$ and the size of the shortest path is from the order $O(v)$. Suri and Hershberger provided the continuous Dijkstra's algorithm with the running time of $O(v \log v)$ in 1995 [9]. In the continuous Dijkstra, the map of the shortest path can be built directly.

The shortest path in three-dimensional space: Finding the shortest path in three dimensions and more is much more difficult than the two-dimensional problem. Even the non-weight mode of the

problem and with polygon obstacles is NP-Hard. For an intuitive understanding of the cause of difficulty of the problem, one can state that unlike the two-dimensional state, maps of the shortest path cannot be placed on any discrete graph and in spite of the fact that the fracture points of optimal paths are placed on the edges of the obstacles, determining the fracture place on the edge accurately is not easy. Therefore, more focus is on finding approximation algorithms and solving the special states of problem.

One of the widely used special form of this problem is to find the shortest path between two points on multi-faceted surfaces (polyhedron). Schorr and Sharir first provided an algorithm for convex forms with the time order $O(v^3 \log v)$ pre-processing and query time $O(k + \log v)$ where k is the number of edges in the shortest path [10]. An algorithm similar to that of continuous Dijkstra was put forth by Han and Chen in 1996 with the time order $O(v^2)$ which can also be used for the non-convex polyhedron [11]. In this algorithm, as we start to move from the source vertex, whenever we reach an edge, we rotate the next page so that it is in line with the current page. By repeating this process and drawing the Euclidean shortest path between the source and the destination, the response is obtained. After opening the pages and imaging them in a two-dimensional surface, as the areas are weightless, we can obtain the shortest path by drawing a line between the two points of source and destination. As a matter of fact, this algorithm provides a structure for each source vertex that can be used to answer any query with a fixed origin and different destination in a very short time. What is important to note here is the order of selecting edges for rotation.

Mitchell and Papadimitriou investigated the problem of finding the shortest path on weighted multifaceted surfaces in 1991 and introduced an algorithms with an approximation factor of $1 + \varepsilon$ and time order $O(n \hat{} L)$ where $L = \log (vNW / w\varepsilon)$ is the accuracy factor of the problem. N is the largest correct integer coordinates of the surface points and W and w are the highest and lowest correct integer weight of the areas and ε is the desired degree of accuracy [12].

3. Acceleration and Parallelization Methods and Techniques for Finding the Shortest Path on the Graph

In this section, the acceleration methods for finding the shortest path on graphs are described in 10 sections. In every section, the approach is also analyzed.

3.1. Partitioning

A common method for the acceleration and parallelization of the shortest path algorithm on straight graphs is to divide the graph into different areas or partitions. Depending on the method of partitioning and communication of areas, there are different versions of this method. This technique has been used due to its capacity to be implemented in parallel processing systems with distributed memory.

Lanthier provided the parallel algorithm for finding the shortest path on weighted Triangulated Irregular Network (TIN)[13]. This algorithm uses Multi-dimentional Fixed Partionaning (MFP) method for the better sharing of data between processors. Consider a shortest path algorithm based on Dijkstra. If data distribution is continuous between the processors (each processor is assigned a connected area of the graph), when running the algorithm, processors will be idle. Figure 1 shows the implementation of Dijkstra's algorithm on a TIN using 9 processors that are connected in a 3 x 3 network. As can be seen, as processing takes place in growing edges in Dijkstra's algorithm, some processors are idle, whereas if we allocate the discontinuous parts of the graph to each processor, this problem is resolved. Figure 2 shows the use of this method.

Of course, this method requires more exchange of information among processors. One way to reduce the need to exchange data in this method is that instead of using the same pattern for the allocation of data to processors in network, the pattern in each section is repeated with a shift towards the neighboring part. Figure 3 shows this method.

To fix the problem of heterogeneity of TIN and clustered TIN, use of the hierarchical structure is recommended. That is, first we consider a network division of the TIN Graph. In areas where the number of vertices is acceptable, we stop the division, but for areas where the number of vertices is high, we run the division of the next layer. Figure 4 shows this method.

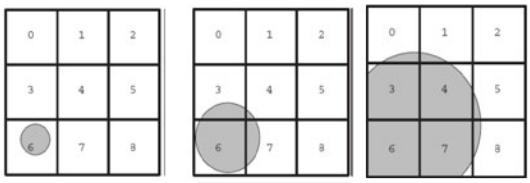

Figure 1 : Unemployment of some processors in the parallel implementation of Dijkstra's algorithm, with the allocation of continuous parts from TIN to each processor [14]

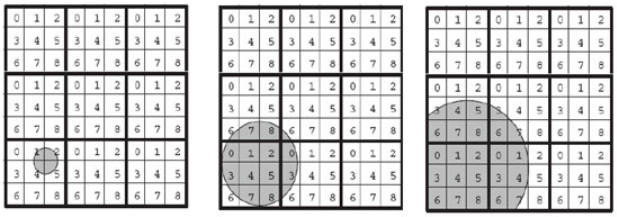

Figure 2 : Allocating the discontinuous areas to each processor and preventing the unemployment of processors [14]

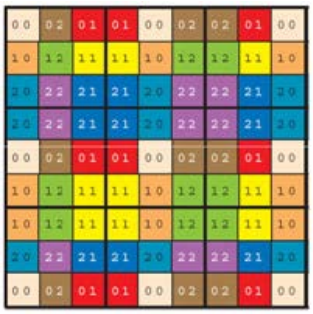

Figure 3 : Reducing the need for communication among processors by changing the allocation pattern [14]

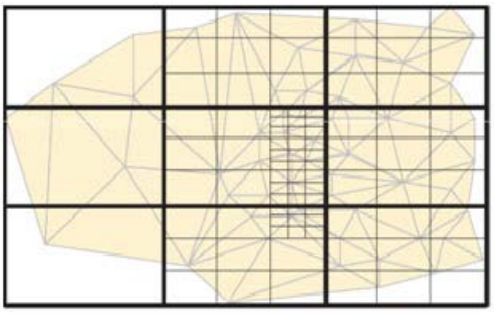

Figure 4 : Hierarchical division of the TIN for heterogeneous graphs [14]

Maue, instead of storing the shortest path between all the border points, the points of origin and destination and the shortest path between each pair of the area is stored and thus the memory is reduced to $O(K^2)$ which, compared to earlier methods, is a significant improvement [15]. Then, using this information, at the time of the search, upper and lower limits for the path are calculated and the search space is reduced accordingly. This method is briefly called Precomputed Cluster Distances (PCD). The approach that has been selected for partitioning is as follows: Suppose the purpose is to divide the graph into k areas. We select the number of $k' > k$ and k' vertex randomly from the graph. Consider a hypothetical vertex which is connected to all the k' vertex with an edge with zero weight and run the shortest path algorithm from this hypothetical vertex. Therefore, the k' vertex of the area

is obtained where each vertex is assigned to the closest vertex of the *k'* of the center vertex. Now we remove some areas. Our criterion for removing areas is the smallest area (the minimum number of vertices) or area with the smallest radius. To remove any area, we disneighbors. The reason we initially chose more than *k* vertices and did the partitioning was to remove the very small areas and achieve a better partitioning. After the areas are formed, for each area consider an additional vertex which is connected to all the other vertices with an edge of zero weight and then run the Dijkstra's algorithm for all such additional vertices. In this way, in the time of *O(k.D (n))* the shortest path between any two areas can be obtained. *D(n)* is the running time of Dijkstra's algorithm.

3.2. Separators

The separator method is in fact a special case of partitioning method. If we do the partitioning in the right way and repeat it enough, linear time order can be achieved.

In 1979, Lipton and Tarjan showed that straight graphs have separators from the order of *O(√n)* and put forth an algorithm with linear time order for it [16]. By separator from the order *O(√n)* we mean that *O(√n)* vertex can be selected and removed from the graph so that the graph is divided into two areas, no edge between the two parts remains, and the size of each area is more than 2/3 of the size of the original graph.

Before the introduction of the algorithm of finding such separator, the concept of the dual of a straight graph needs to be explained. For any straight graph which is embedded on the page, if for every facet of the graph, we place a vertex and put an edge between the corresponding vertices which are neighbors, a dual graph is obtained. Figure 5 shows a straight graph with its dual.

The point is that if we remove corresponding edges of a round in the first graph from the dual graph (or vice versa), the dual graph is divided into the two areas of inside and outside of the circle. Figure 6 shows this. Round C in the original graph has divided the vertices of the dual graph into two categories and the C' in the dual graph has divided vertices of initial graph into two groups. Given these facts it can be proved that when a straight graph has a spanning tree with the depth of *d*, it has a separator with the maximum size of *2d + 1*.

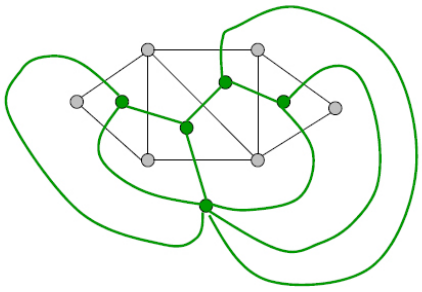

Figure 5: A planar graph with its dual

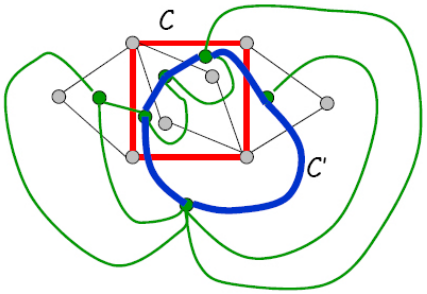

Figure 6: Round in a straight graph and dividing the vertexes in its dual into two groups

The algorithm is given here. If the graph is not connected, it is sufficient to divide only the largest connected component. So we can assume that the graph is connected. Form a spanning tree of the first level on the graph and number its levels (Figure 7). Number the vertices of the graph by arranging

the first meeting and name the level where *n/2* of vertex is located L_i (Figure 8). If the number of vertices is *O(√n)*, the vertexes of this level can be announced as a separator and then it is done. Otherwise, find the levels L_i and L_k in such a way that L_i is higher than L_j and L_k is lower than L_j, their size is *O(√n),* and the size of all the levels between them is equal to or larger than *O(√n)*. (Note that L_k can be empty.) Consider the number of layers between L_i and L_k imagining a hypothetical root for these levels as a tree with a depth of less than *O(√n)* and find the separator for it. The separator of the tree with L_i and L_k is separator of the original graph from the order O (√n) (Figure 9).

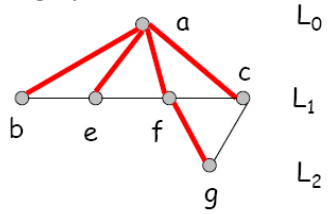

Figure 7 : Forming the first tree of level and numbering the levels

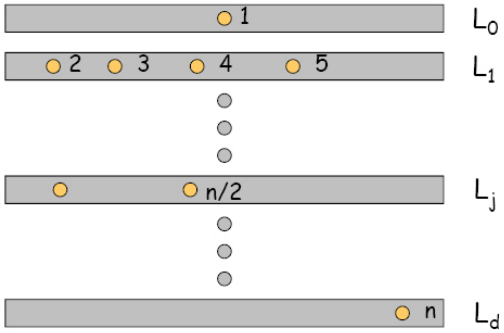

Figure 8 : Finding the middle vertex

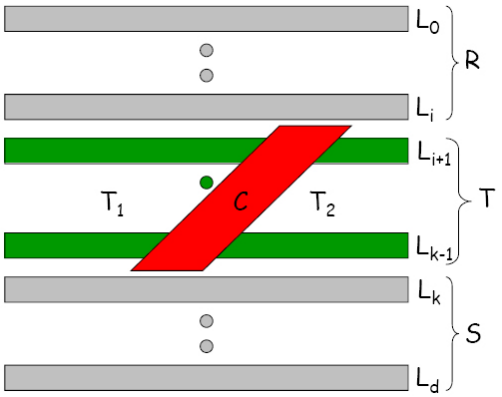

Figure 9 : Graph separator includes separator of T part as well as vertexes of the layers L_i and L_k

3.3. Hierarchical

One of the well-known methods is to use the hierarchical highway [17,18]. In this approach, some of the edges are selected and stored as highways, according to their positions in the graph, and at the time of the query, finding the shortest path on the highway is done based on them, and the search space along with the required time is reduced consequently. Calculating the edges in the highway is done during the pre-processing time.

3.4. Marking the edge

In marking the edge method, in the pre-processing stage, it becomes clear which edges are involved in the shortest path of which areas. Thus, at the time of the query, the number of edges which needs to be checked is reduced to a large extent. This method also needs partitioning and the way it works is in a way that if the origin and destination are in one area, it is not different from the normal Dijkstra's algorithm. However, if the origin and destination are in different areas, it can use the saved information [19]. Marking the edge needs the memory $\Theta(nk)$ (*k* is the number of areas). Pre-processing time is also

$\Theta(B.D(n))$ where B is the number of vertices of areas and $D(n)$ is the running time of Dijkstra's algorithm.

3.5. Routing based on access

The routing method also requires a relatively large pre-processing and memory. This method is similar to marking the edge approach. Access to each edge is defined as follows: consider all the shortest paths that this edge is inside. Calculate the shortest distance of the edge to either end of the path, and among all the shortest paths, place the shortest distance to both ends as access to edge [20]. Calculating access for each edge at the time of query reduces the time necessary for finding the shortest path.

3.6. Geometric maintenance

Geometric maintenance, among other methods, is similar to marking the edge. This method is applicable to graphs that have been formed on the basis of geometric structure. In this method, in the pre-processing stage, for each vertex, it is calculated that the shortest path from this vertex to any another vertex which passes from an adjacent edge, according to geometric information. This approach needs a relatively large time and memory in the pre-processing stage [21].

3.7. Bilateral search

Bilateral search is a well-known and simple technique. Here, the shortest path algorithm both from the source and the destination (in case of a directed graph, a reverse direction is taken) is run concurrently, and whenever the boundaries of the two search area reach each other for the first time, the shortest path is obtained. Thus, on average, fewer vertexes to find the shortest path are met and the running time of the algorithm is reduced. In most cases, the number of vertices visited is reduced to 50%.

3.8. Search towards the target

In search towards the target methods, through considering more information from the problem, the search is done in a way that is more close to the destination and the search space is reduced. A* search is the most famous of these methods. The method presented by Maue et al., which was introduced in the partitioning section, can be considered part of this group [15].

Figure 10 shows the impact of using methods of bilateral search and search towards the target. The gray area has been traversed in the simple Dijkstra's algorithm. Using bilateral search only, the search space is reduced to the light blue area. Using search to the target techniques, the performance is improved (the dark blue area) and using better functions to calculate the low limit of the shortest path to the destination, more acceleration can be achieved.

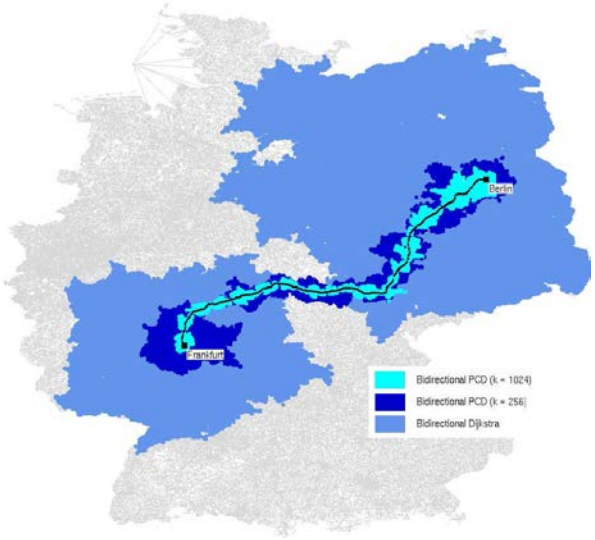

Figure 10 : The impact of bilateral search and search towards the target in reducing the search space [15]

3.9. Important vertices

In this method, *k* vertices are selected as important vertices and the shortest path between each important vertex with all the other vertices is maintained [11]. This method is very fast at the time of query. But a relatively large memory *O(nk)* is necessary in it. Pre-processing time is $\Theta(k\,D\,(n))$. The way this method works is through maintaining the lower limit for the passing path from each vertex with regard to the triangle inequality. Afterwards, this lower limit is used in the bilateral search algorithm A*. This method is not complex and shows remarkable results with little pre-processing [22].

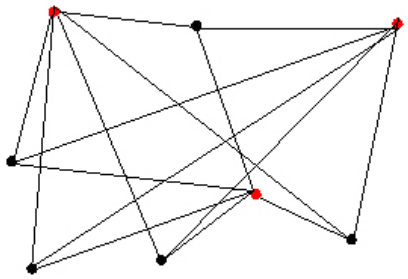

Figure 11 : The method of important vertices: three important vertices and four regular black vertices are marked in red and black, respectively.

3.10. Delta step

Among the acceleration and parallelism methods of the shortest path algorithm, delta-step is the only one which is only suitable for parallelization. This algorithm is associated with an increase in work and as a result spends much more time than the conventional Dijkstra's algorithm [23].

The algorithm works with the idea of combining the two methods of label setting and label correction. Here, from the vertices whose shortest path has been calculated, the possibility of continuing the path to Δ, similar to the methods of label correction, exists. In fact, the algorithm proceeds from layer to layer on the graph. The layers go ahead in the form of label allocation and within each layer in the form of label correction. Δ is the maximum diameter required for the layers. Therefore, with the minimum diameter of the layer, we reach the Dijkstra's algorithm and with the maximum diameter, the Bellman Ford algorithm.

Conclusion

In this paper, the acceleration and parallelization methods for finding the shortest path on graphs were discussed. These methods were presented in 10 sections, including partitioning, separators, hierarchical, marking the edge, routing based on access, geometric maintenance, bilateral search, search to the target, important vertices, and delta step. Each approach was discussed; advantages, disadvantages, and time and space complexities of each one were also investigated.

References

[1] E. W. Dijkstra, *A Note on Two Problems in Connection with Graphs*, Numerische Mathematik 1, pp. pp. 269-271, 1959.

[2] M. R. Henzinger, P. Klein, S. Rao, S. Subramanian, *Faster Shortest-Path Algorithms for Planar Graphs*, Journal of Computer and System Sciences, Vol. 55, No. 1, 1997.

[3] M. Thorup, *Undirected single-source shortest paths with positive integer weights in linear time*, Journal of the ACM (JACM), Vol. 46, No. 3, 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms* ,3rd ed., MIT Press, 2009.

[5] D. B. Johnson, *Efficient Algorithms for Shortest Paths in Sparse Networks*, Journal of the ACM (JACM), Vol. 24, No. 1, 1977.

[6] T. M. Chan, *All-pairs shortest paths for unweighted undirected graphs in o(mn) time*, in proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, New York, 2006.

[7] E. Welzl, *Constructing the visibility Graph for n Line Segments in O(n^2) Time*, Information Processing Letters, Vol. 20, No. 4, pp. 167-171, 1985.

[8] M. Pocchiola, G. Vegter, *Computing the Visibility Graph via Pseudotriangulations*, In Proceesing of 11th Annual ACM Symposim of Computational Geometry, pp.248-257, 1995.

[9] J. Hershberger, S. Suri, *An optimal Algorithm for Euclidean Shortest Paths in the Plane*, SIAM Journal on Computing, Vol. 28, No. 6, pp. 2215-2256, 1999.

[10] M. Sharir, A. Schorr, *On Shortest Path in Polyhedral Spaces*, SIAM Journal of Computing, Vol. 15, No. 1, pp. 193-215, 1986.

[11] J. Chen, Y. Han, *Shortest Paths on a Polyhedron*, In Proceedings of the sixth annual symposium on Computational geometry, pp. 360-369, 1990.

[12] J. S. B. Mitchell, C. H. Papadimitriou, *The Weighted Region Problem: Finding Shortest Path Trough a Weighted Planar Subdivision*, Journal of the ACM, Vol. 38, 1991.

[13] M. Lanthier, D. Nussbaum, J. R. Sack, *Parallel implementation of geometric shortest path algorithms*, ACM Parallel Computing, Vol. 29, No. 10, 2003.

[14] A. K. Phipps, *Parallel algorithms for geometric shortest path problems*, Master's thesis, School of Informatics, University of Edinburgh, 2004.

[15] J. Maue, P. Sanders, D. Matijevic, *Goal-directed shortest-path queries using precomputed cluster distances* , Journal of Experimental Algorithmics (JEA), Vol. 14, 2009.

[16] R. J. Lipton, R. E. Tarjan, *A separator theorem for planar graphg,* SIAM Journal on Applied Mathematics, Vol. 36, No. 2, 1979.

[17] P. Sanders and D. Schultes, *Highway Hierarchies Hasten Exact Shortest Path Queries*. In Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05), Vol. 3669 of Lecture Notes in Computer Science, pp. 568–579, 2005.

[18] P. Sanders and D. Schultes. *Engineering Highway Hierarchies.* In Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06), Vol. 4168 of Lecture Notes in Computer Science, pp. 804–816, 2006.

[19] U, Lauther. *An Experimental Evaluation of Point-To-Point Shortest Path Calculation on Roadnetworks with Precalculated Edge-Flags*, The Shortest Path Problem: Ninth DIMACS Implementation Challenge, 2006.

[20] HR. Gutman, *Reach-based Routing: A New Approach to Shortest Path Algorithms optimized for Road Networks,* in proceedings of 6th International Workshop on Algorithms Engineering and Experiments, pp. 100-111, 2004.

[21] D. Wagner , T. Willhalm, *Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs,* in proceedings of 11th European Symposium on algorithms, Vol. 2832 of Lecture Notes in Computer Science, pp. 776-787, 2003.

[22] Goldberg, A.V., Harrelson, C., *Computing the shortest path: A* meets graph theory.* In proceedings of 16th ACM-SIAM Symposium on Discrete Algorithms, pp.156-165, 2005.

[23] K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, *An Experimental Study of A Parallel Shortest Path Algorithm for Solving Large-Scale Graph Instances*, In Workshop on Algorithm Engineering and Experiments, New Orleans, 2007.