Analysis of algorithms for shortest path problem in parallel

Bogdan Popa[1], Dan Popescu[2]
[1] Department of Automation and Applied Informatics, University of Craiova, Craiova, Romania
[2] Department of Automation and Applied Informatics, University of Craiova, Craiova, Romania
Email: bogpo89@yahoo.com, dpopescu@automation.ucv.ro,

Abstract — **This article brings out the usefulness of improving classical algorithms, analyzing, and optimizing the efficiency of parallel execution time at any price, depending on the situation of use. It can be said that nowadays the searching algorithms are involved in multiple actions in different domains starting from the data bases, the online searching engine, GPS systems or in the emergency situations. Also these algorithms are to be found in the network where we can talk about real priority schemes and data transfer speed that matters a lot today. Even in the top management systems these algorithms are very useful today because every decision has an important component of time and the searching for information is direct related with these type of algorithms. This study offers an innovative and efficient approach of Dijkstra's algorithm, Bellman Ford algorithm, Floyd-Warshall algorithm and Viterbi algorithm through parallel programming and analysis of the results obtained in different tests but also a comparison of those searching strategies on graph systems. This study can generate new approaches over those strategies and can generate new discussions over the necessity of improving the old algorithms.**

Keywords—parallel programming; Dijkstra's algorithm; study of algorithms; shortest path; comparison of algorithms for shortest path;

I. INTRODUCTION

Dijkstra's algorithm, conceived by computer scientist Edsger Dijkstra in 1956 and published in 1959,[1][2] is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing the shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms. For a given source vertex (node) in the graph, the algorithm finds the path with the lowest cost (i.e. the shortest path) between that vertex and every other vertex [3] (although Dijkstra originally only considered the shortest path between a given pair of nodes [4]).

Route planning in transportation networks is one of the most active topics in the research field of algorithm engineering with many real world applications like navigation systems, online route planning systems (e. g. Google Maps) or timetable information systems. Usually, the transportation network is modeled as a weighted graph. Then the problem can be solved by applying Dijkstra's algorithm to find the shortest-path between two nodes s and t [5]. Dijkstra's algorithm is an algorithm that will determine the best route to take, given a number of vertices (nodes) and edges (node paths). So, if we have a graph and if we follow Dijkstra's algorithm we can efficiently figure out the shortest route no matter how large the graph is. The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The Floyd–Warshall algorithm is an example of dynamic programming, and was published in its currently recognized form by Robert Floyd in 1962 [6]. In computer science, the Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights but with no negative cycles [7]. The Viterbi algorithm is named after Andrew Viterbi, who proposed it in 1967 as a decoding algorithm for convolutional codes over noisy digital communication links [8]. Viterbi (path, algorithm) has become a standard term for the application of dynamic programming algorithms to maximization problems involving probabilities [9]. The Viterbi algorithm also solves the shortest path route, but taking into account the probability of every edge. For example, this algorithm is used in GPS systems for traveling where there are priorities or visiting places where it is more important to arrive. The analysis between those 4 algorithms depends on the type of the algorithm and the method that resolve the shortest path issue with different strategies. The weight of an edge may correspond to the length of the associated road segment, the time needed to traverse the segment or the cost of traversing the segment [10].

II. PROPOSED STRUCTURE/ARCHITECTURE OF THE PARALLEL CONCEPTS

In the cases of those algorithms there are defined different strategies for the shortest path. In the first case the Viterbi algorithm is most different because it takes into account the percent for every edge and cannot be compared with the same cases for the testing examples. Another difference is between the algorithms that solve the problem for negative edges and with negative cycles or not. Floyd-Warshall solves the all-pairs shortest path problem while Dijkstra's algorithm is for the single-source shortest path problem. The proposed change of the algorithms makes the parallelization from the beginning of the process. It starts to find the shortest path from the starting node to every other node using in the best cases different paths and in the worsts cases it can use the same path for more nodes. The favorable cases can be less than 1% in the random test packages,

978-1-4673-8606-7/16/$31.00 ©2016 IEEE

but there are cases when this scenario is useful. One drawback is that the proposed scenario for the parallel algorithm uses information not related to roads traveled to compare, because it starts calculating directly from a central node all other nodes in part. Also for every algorithm this case can be different, but some analysis will be presented for the first three types of algorithms.

The pseudo-code is as follows for the Dijkstra's algorithm.

```
for all v in G:
d(v) = infinity;
D = {s};
U = {G-s}; // all other nodes in G
for all u adjacent to s:
d(u) = w(u, s);
while U is not empty {
/* parallelization takes place here */
Let v be the node from U with minimal d(v);
U = U / v;
D = D union v;
for all u adiacent to v {
if d(u) > w(u,v) + d(v) then
d(u) = w(u,v) + d(v);
} }
```

The pseudo-code is as follows for the Bellman Ford algorithm.

```
Bellman-Ford(G, source)
for i in 1 to |U| do
distance[i] = +inf
predecessors[i] = null
distance[source] = 0

for i in 1 to (|U| - 1) do
for each Edge e in Edges(G) do
/* parallelization takes place here */
if distance[e.from] + length(e) < distance[e.to] do
distance[e.to] = distance[e.from] + length(e)
predecessors[e.to] = e.from

for each Edge e in Edges(G) do
/* parallelization takes place here */
if distance[e.from] + length(e) < distance[e.to] do
error("Graph contains cycles of negative length")
return predecessors
```

In the testing part of the algorithms it was detected that the type of the test is the most important, but there are types of routes that can offer interesting results, also that method is redundant, because it starts another execution for every other node without the starting node.
For example, it is important to do test with simple route or graphs for better understand the evolution of the algorithms.
For the first time it should be taking into account the execution time over the sequential algorithms, that is presented in the next table and in the next chart.

Number of nodes	Dijkstra's	Bellman Ford's	Floyd-Warshall's
250	0.01431	0.01749	0.13489
400	0.04	0.04998	0.53218
600	0.07321	0.08214	0.89912
1000	0.09238	0.10326	
1200	0.11787	0.13521	
1500	0.15765	0.18532	
1800	0.18021	0.21892	
2000	0.20021	0.24136	
2400	0.24532	0.2891	
3000	0.29311	0.34122	
3600	0.33427	0.39922	
4000	0.43921	0.51021	
5000	0.59043	0.72925	

Table 1: Results in seconds of the execution times for sequential algorithms

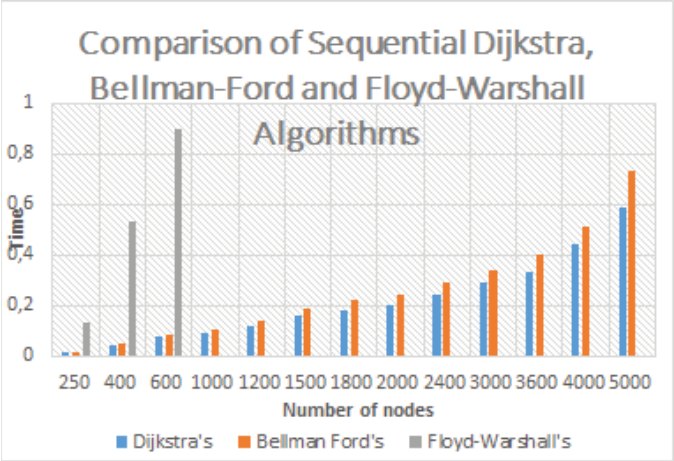

Figure 1: Chart with the results for the first algorithms for the first table, in seconds and number of nodes

The execution time is directed influenced by the type and structure of the graph, it involves parameters like:
-number of vertices
-number of edges
-for this case specially the high level of the graph
In the previous tests there was the simple structure of roads with fewer connections between the vertices.

It is obvious that the Floyd-Warshall algorithm is more slower because it is a minimal roads algorithms and can consume more time for the execution and will not be tested in the next chapters. The Floyd–Warshall algorithm is a good choice for computing paths between all pairs of vertices in dense graphs, in which most of all pairs of vertices are connected by edges.

		Negative edges?	**Negative cycle?**	**Time complexity**
Dijkstra	Single source	No	No	$O(\|V\|^2)$
Bellman Ford	Single source	Yes	Yes	$O(\|V\| \cdot \|E\|)$

Table 2: Comparison between the Dijkstra and the Bellman Ford sequential algorithms

Taking into account the results and the time complexity study, the Dijkstra algorithm is more efficient than Bellman Ford algorithm for the sequential implementations.The Dijkstra algorithm is faster than Bellman Ford algorithm. Bellman Ford strategy is not suggested for larger networks.

The favorable cases are found at very high processing capabilities in case of systems with a central point of the search, for example a telephone or a dispatcher with network connections for a city of several million inhabitants. In these cases there are centralized commands and information systems that must respond quickly, architecture for these types of networks can be included in favorable types of application for the proposed solution.

From the beginning it is obvious that the main idea of these implementations consists in the structure of the graph. With less nodes tests it can be concluded that there are some different cases :

1) Worst case scenario:

Figure 2: Example minimal worst case scenario

2) Best case scenario:

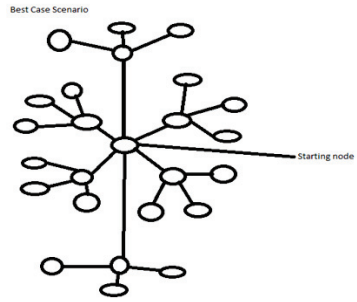

Figure 3: Example best case scenario

3) Middle example- Minimal case:

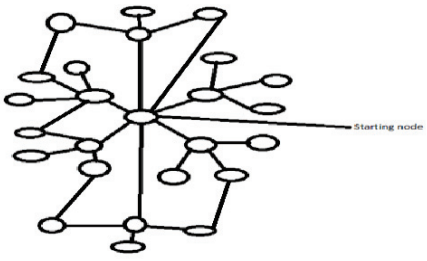

Figure 4: Example of middle case scenario with centralized starting node

Taking into account the efficiency, in terms of big O notion it is O(n^2), which is efficient. Specifically, suppose G has n vertices and m edges. In the cases described above, its evidenced structure is favorable for the development of the strategy chosen.
Thus, it can be seen that the most important factor for the use of parallel algorithm is given by the ratio of the depth of the graph and the number of nodes. The higher and closer this ratio to 0 is, the parallel strategy is more useful.

Number of nodes	Dijkstra's Sequential	Bellman Ford's Sequential	Dijkstra's Parallel	Bellman Ford's Parallel
250	0.01431	0.01749	0.04432	0.04551
400	0.04	0.04998	0.07432	0.07621
600	0.07321	0.08214	0.10873	0.11001
1000	0.09238	0.10326	0.13674	0.13882
1200	0.11787	0.13521	0.16672	0.16992
1500	0.15765	0.18532	0.19652	0.20671
1800	0.18021	0.21892	0.24001	0.25011
2000	0.20021	0.24136	0.26019	0.28732
2400	0.24532	0.2891	0.29115	0.31882
3000	0.29311	0.34122	0.32271	0.35611
3600	0.33427	0.39922	0.36511	0.38321
4000	0.43921	0.51021	0.48821	0.51221
5000	0.59043	0.62925	0.62991	0.65819

Table 3: Table with the results for the first algorithms in seconds

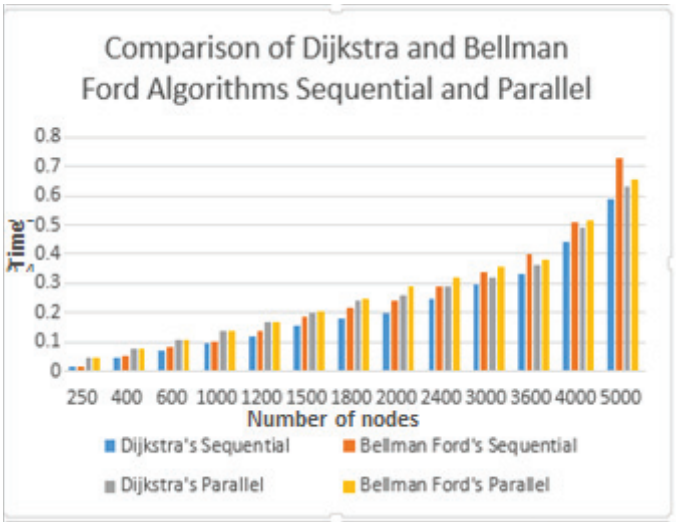

Figure 5: Chart with the results for the first algorithms for the Table 3 results

The previous images are related strictly to the cost time for the two algorithms and offer results that conclude that the parallel implementations are not useful in those cases and are slower than the sequential form. But also today there are some efficient parallel strategies for this type of algorithms.

III. TECHNOLOGY USED

Message Passing Interface (MPI) is a standardized and portable message-passing system, designed by a group of researchers from academia and industry, to function on a wide variety of parallel computers[11]. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77 or the C programming language. Several well-tested and efficient implementations of MPI include some that are free and in the public domain. These fostered the development of a parallel software industry, and they encouraged the development of portable and scalable large-scale parallel applications[12]. It was testing only the sequential application and the parallel application on simple and normal C implementation and on MPI implementation. The results of the tests, randomly test, comparable only low than 200 nodes and 500 edges denotes that the sequential algorithm is better on this configurations. In the project of dr. Russ Miller and Adytia Pore, „Parallel implementation of Dijkstra's algorithm using MPI library on a cluster" , another strategy of parallelization for Dijkstra's algorithms based on MPI technology and clustering systems is presented . An interesting report of the efficiency is presented in the next graphic :

Number of Cores	1	2	4	8	16	32
Input-Size	RUNTIME		IN		SECONDS	
625	0.76589	0.70187	0.58532	0.42618	0.25125	0.30325
2500	1.08971	0.79816	0.57821	0.41344	0.38815	0.44516
10000	3.25618	1.89876	1.10542	0.78516	0.54812	0.80124

Table 4: Tabulation of Results of second experiment [13]

In this experiment where the proposed work is distinguished, according to the author, the following conclusions are to be drawn:

a) Run Time varies Inversely with the number of Cores.

b) Algorithm found to be most-effective performance-wise for16 Core configuration.

c) 32-Cores: Run time increases Slightly as communication overhead defeats the purpose of using more number of cores for computation.

The Figure 5 offers the results of implementations over execution time, over the sequential algorithm and the parallel algorithm presented before for random types of graphs. The results are in milliseconds depending on the number of nodes in graph. An complete analysis can bring some conclusions over the time direct dependent by the structure of the input tests. Also, the Bellman Ford algorithm offers more time for execution in both types of implementations.

From the execution experience, probably the algorithm will give better results than the sequential algorithm in test for about 100.000 nodes-compared with the Best Case Scenario. The favorable structure is for bushy trees with small height with simple connectivity and no more roads between the vertices.

Another MPI implementation based on the cluster systems that identify the closest vertices to the source vertex, uses parallel prefix to select the globally closest vertex and broadcasts the results to all core. This was made by Priyanka Ramanuja Kulkarni, Meenakshi Muthuraman, Aslesha Pansare from the University of Buffalo, and the results graphic confirm that on many cores and on more complex tests can obtain better results.

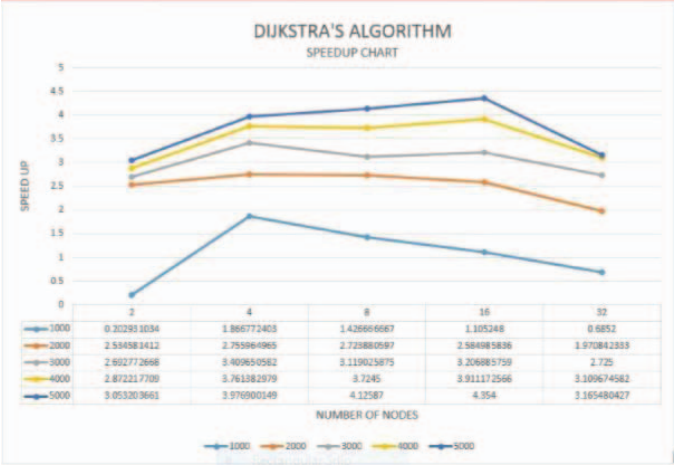

Figure 6: Example of Dijkstra algorithm speed up chart[14]

In these speed-up chart that represent the results of the proposed MPI implementations results[14] that at the maximum number of cores(32), and over the most difficult testing system, at 5000 nodes, the results are approximately the same like at the 4000 nodes, and these represent the benefits of the clustering system of allocation over the node system. Also, there are more differences between the 1000 nodes and 2000 nodes, for the time, because is a major increasing number of operations.

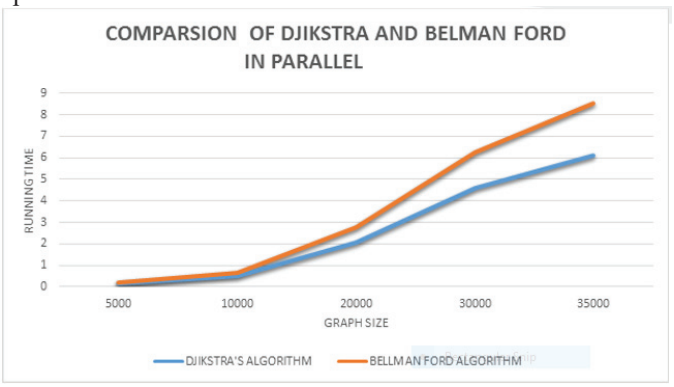

Figure 7: Comparison of Dijkstra and Bellman Ford in Parallel- Spped-Up analyse [14]

On the clustering systems implemented on the Bellman Ford algorithm, they also reflects that the Dijkstra's method is better for execution time in every case with minimal differences like in our results.

Another interesting part of the study may take into account the percentage of CPU utilization or under test packages for this time of the algorithm using it. Also for modern computing systems, we can mention the impact of the number of processors

on the calculation for this algorithm, more percentage of use of each processor involved in the necessary tests.

In the figure 8, the CPU usage over the execution of the random case study of graph is presented, in parallel scenario, taking into account the Intel I7 CPU Q720 at 1.60GHz type, the 64-bit system used from Microsoft. It is obvious presented in the graphic that for this process in operation up to 4 cores were ordered and a specific architecture such as Gauss bell schedule, with a fast starting and finish time. Comparing this example with another examples [14], there is many differences and for this type of supporting structures are implemented parallel solutions.

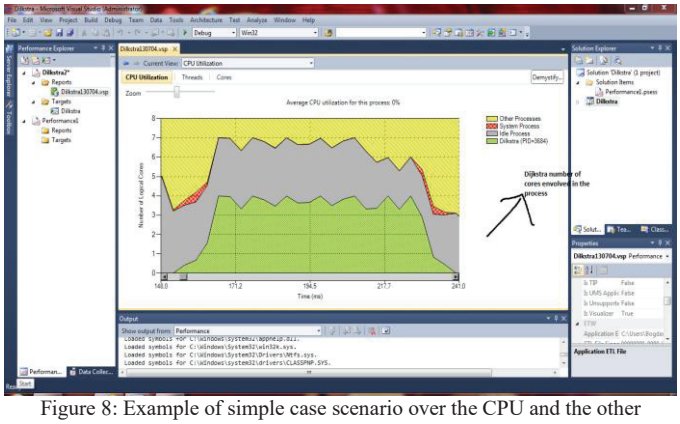

Figure 8: Example of simple case scenario over the CPU and the other processes.

Another interesting study about this algorithm has proposed simple criteria which divide Dijkstra's sequential SSSP (Single Source Shortest Path) algorithm into a number of phases, such as the operations within a phase that can be done in parallel[15]. They give a PRAM (Parallel Random-Access Machine) algorithm based on these criteria and analyze its performance on random digraphs with random edge weights uniformly distributed in [0, 1]. Also Y. Tang, Y. Zhang and H. Chen, tested a performance system over 15-fold speedup on 16 processors in an IBM cluster. They focus on satisfying the actual demands of quickly finding the shortest paths over real-road networks in an intelligent transportation system [16].

IV. Conclusions

This article proposes a parallel implementations form over most utilization of the shortest path algorithms today, useful in some cases of application, such as computer systems with a low height, but big density of nodes. Taking into account the major impact, the Dijkstra algorithm is today utilized in network system, GPS systems, 3D wireless sensors. The Bellman Ford offer today perspectives in distance-vector routing protocols, for optimal routes.The Floyd Warshall is utilized in this applications where it is interested in finding the path with the maximum flow between two vertices, for fast computation of Pathfinder networks. The Viterbi algorithm is used for the application of dynamic programming algorithms to maximization problems involving probabilities, on the speech analysis and for the probability systems roads. George Dvorsky tried to explain the importance that the algorithms have in our world today and which are the most important for our civilization. He made a top of algorithms with the major impact today and the Dijkstra's algorithm is on the 3rd place, following the sorting algorithms and the Fourier transform. Today, even when we have better solutions to the problem of finding the shortest path, Dijkstra's algorithm is still used in systems that require stability [17]. In the testing part of the study, there are explained the strategies for parallel solutions offered today, the type which is used and a short analysis over the results. It can be said that the proposed method is unusable because there are better proposed strategies, taking into account the clustering systems over the tests with multiple number of vertices and edges. The efficiency of the algorithms is expensable today, in time, in consumption, in information transmission time and directed in money for big companies. Nowadays, this subject is in the top of the improvement of the basic systems.

References

[1] Dijkstra, Edsger; Thomas J. Misa, Editor . "An Interview with Edsger W. Dijkstra". "What is the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path which I designed in about 20 minutes. One morning I was shopping with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.", August 2010, Communications of the ACM 53 (8): 41–47. doi:10.1145/1787234.1787249

[2] Dijkstra, E. W. , "A note on two problems in connexion with graphs",1959 Numerische Mathematik 1: 269–271. doi:10.1007/BF01386390.

[3] Mehlhorn, Kurt; Sanders, Peter , Algorithms and Data Structures: The Basic Toolbox. Springer, 2008

[4] Dijkstra, Edward W., Reflections on "A note on two problems in connexion with graphs

[5] Verification of Shortest-Path Algorithms. Automated Software Analysis Group. Kalsruhe Institut of Technology, Thomas Pajor, Mana Taghdiri, Dorothea Wagner

[6] Floyd, Robert W. . "Algorithm 97: Shortest Path". Communications of the ACM, June 1962

[7] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill. ISBN 0-262-03141-8. See in particular Section 26.2, "The Floyd–Warshall algorithm", pp. 558–565 and Section 26.4, "A general framework for solving path problems in directed graphs", 1990, pp. 570–576.

[8] G. David Forney Jr: The Viterbi Algorithm: A Personal History, 29 Apr 2005

[9] Daniel Jurafsky; James H. Martin. Speech and Language Processing. Pearson Education International. p. 246

[10] Wei Zhang ,Hao Chen , Chong Jiang , Lin Zhu,"Improvement And Experimental Evaluation Bellman-Ford Algorithm",International Conference on Advanced Information and Communication Technology for Education (ICAICTE 2013),2013

[11] Dept. of Comput. Sci. & Eng., Ohio State Univ., Columbus, OH ; Koop, M.J. ; Panda, D.K," High-performance and scalable MPI over InfiniBand with reduced memory usage.", SC 2006 Conference, Proceedings of the ACM/IEEE

[12] Table of Contents — September 1994, 8 (3-4). "Mpi Terms and Conventions", International Journal of High Performance Computing Applications September 1994

[13] Dr. Russ Miller, Adytia Pore, Parallel implementation of Dijkstra's algorithm using MPI library on a cluster.

[14] Priyanka Ramanuja Kulkarni, Meenakshi Muthuraman, Aslesha Pansare, Nikhil Kataria, "Parallel Implementation of Dijkstra's and Bellman Ford Algorithm", University of Buffalo, 2014

[15] A. Crauser, K. Mehlhorn, U. Meyer, P. Sanders, "A parallelization of Dijikstra's shortest path algorithm", in Proc. of MFCS'98, pp. 722-731, 1998.

[16] Y. Tang, Y. Zhang, H. Chen, "A Parallel Shortest Path Algorithm Based on Graph - Partitioning and Iterative Correcting", in Proc. of IEEE HPCC'08, pp. 155-161,2008.

[17] George Dvorsky," The 10 Algorithms That Dominate Our World "