SOME INTERESTING AND NOVEL RESULTS FROM ANALYSIS OF SHORTEST PATH SEARCHING ALGORITHMS

[1]SANTOSH KU. MAJHI, [2]SAMPA CHAUPATTANAYAK

Department of Computer Science and Application, Utkal University,Vanivihar, Bhubaneswar,Odisha-751004
Email: santoshism9@gmail.com

Abstract: In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. It is very challenging and tough task for designing or modifying the parallel algorithms due to the time and space complexity. In this paper we studied a brief overview of different shortest path algorithms and its issues and applications in the different areas. We also describe the different techniques used for designing and its methodologies. Lastly we presented the complexity analysis for the different algorithms.

Keywords: BFS, Queue, Dijkstra, Bellman-Ford, PSPS Algorithm

I. INTODUCTION

The **shortest path problem** is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. There are several types of shortest path problems. The **single-source shortest path problem**, in which we have to find shortest paths from a source vertex *v* to all other vertices in the graph. e.g. Dijkstra's algorithm [1],[2].The **single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex *v*. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph. The **all-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices *v*, *v'* in the graph. e.g. Floyd–Warshall algorithm [5]

For solving the shortest path problems different algorithms have been developed. These are **Dijkstra's algorithm** conceived by Dutch computer scientist Edger Dijkstra in 1959 [1], is a shortest path algorithm that solves the single-source shortest path problem for a graph with non negative edge path costs, outputting a shortest path tree. This algorithm is often used in routing. The calculation time for Dijkstra's algorithm is $O(|V|^2)$ where V is the number of vertices of the graph. **Bellman–Ford algorithm** conceived by Richard Bellman and Lester Ford, Jr. is a shortest path algorithm that solves single source shortest path problems if the edge weights may be negative. The calculation time for Bellman–Ford algorithm is O (|V|.|E|) where |V| and |E| are the number of vertices and number of edges. **A* search algorithm** conceived by Peter Hart, Nils Nilsson and Bertram Raphael in 1968 is a shortest path algorithm that solves the single pair shortest path using heuristics to speed up the search. It has better performance than the Dijkastra's algorithm. **Floyd–War shall algorithm** conceived by Robert Floyd and Stephen War shall in 1962 solve all pair's shortest path problems. The calculation time for Floyd-War shall algorithm is $\Omega(|V|^3)$, where |V| is the number of vertices. **Johnson's algorithm** conceived by Donald B. Johnson in 1977 which finds the shortest paths between all pairs of vertices in a sparse directed graph and may be faster than Floyd–War shall algorithm on sparse graphs. **Perturbation theory** finds (at worst) the locally shortest path.

II. EXISTING WORK

E. W. Dijkastra gave a way to find shortest path between two nodes. Let's consider n points (nodes), some or all pairs of which are connected by a edge; the length of each edge is given. Here one restrict to the case where at least one path exists between any two nodes. Now consider two issues:
1: construct the tree of minimal total length between the 'n' nodes (a tree is a graph with one and only one path between the every two nodes).
2: Find the path of minimum total length between two given nodes P and Q.

The above problems can be solved by some processes which will give the shortest path between the nodes.

V Kumar, Computer sc. Department, University of Minnesota proposed a way of determine the Scalability of Parallel Algorithms for All Pair Shortest path Problems. This Paper uses the isoefficiency matrix to analyze the scalability of several parallel Algorithms for finding shortest path between all pairs of nodes in densely connected graph. Scalability is analyzed with respect to the mesh, hypercube, and shared memory architecture.

Kamesh Madduri in 2000 defined a way of finding shortest path in Parallel for Solving Large-Scale Graph Instances It presents an experimental study of the delta- stepping parallel algorithm for solving the single source shortest path problem on large-scale graph instances. In addition to

applications in combinatorial optimization problems; shortest path algorithms are finding increasing relevance in the domain of complex network analysis.

James B. Orlin, Sloan School of Management, MIT, Cambridge, MA in 2001 proposed a faster Algorithm for the Single Source Shortest Path Problem with Few Distinct Positive Lengths. In this paper, we propose an efficient method for implementing Dijkstra's algorithm for the Single Source Shortest Path Problem (SSSPP) in a graph with positively lengthed edges, and where there are few distinct lengths. The SSSPP is one of the most widely studied problems in theoretical computer science and operations research. On a graph with n vertices, medges and K distinct edge lengths, our algorithm runs in O (m) time if nK <=2m and O (mlog nK/m) time, otherwise. We tested our algorithm against some of the fastest algorithms for SSSPP on arbitrarily lengthed graphs. Our experiments on graphs with few edge lengths confirmed our theoretical results as the proposed algorithm consistently dominated the other SSSPP algorithms that did not exploit the special structure of having few distinct edge lengths.

F. Benjamin Zhan, Department of Geography and Planning, Southwest Texas State University, San Marcos, TX 78666, USA in 2003, proposed three fastest shortest path algorithms on real road networks. It is well known that computing shortest paths over a network is an important task in many network and transportation related analyses. Choosing an adequate algorithm from the numerous algorithms reported in the literature is a critical step in many applications involving real road networks. In a recent study, a set of three shortest path algorithms that run fastest on real road networks has been identified. These three algorithms are: 1) the graph growth algorithm implemented with two queues, 2) the Dijkstra algorithm implemented with approximate buckets, and 3) the Dijkstra algorithm implemented with double buckets. As a sequel to that study, this paper reviews and summarizes these three algorithms, and demonstrates the data structures and procedures related to the algorithms. This paper should be particularly useful to researchers and practitioners in transportation, GIS, operations research and management sciences.

U Meyer et.al. Proposed Delta-stepping algorithm, a generalization of Dial's algorithm and the Bellman-Ford algorithm, improves this situation at least in the following "average-case" sense: For random directed graphs with edge probability d/n and uniformly distributed edge weights a PRAM version works in expected time O (log3 n/log log n) using linear work. The algorithm also allows for efficient adaptation to distributed memory machines. Implementations show that the approach works on real machines. As a side effect, they get a simple linear time sequential algorithm for a large class of not necessarily random directed graphs with random edge weights.

Topkis, D.M. et al studied an efficient and flexible algorithm which is presented for finding a k shortest loopless path with distinct initial links from one node to each other node. Low-order polynomial bounds are established for the worst-case time complexity of the algorithm, showing it to offer a substantial improvement over applying known algorithms to the problem. The algorithm can incorporate various extensions, including the ability to handle an algebraic objective, which enhance its applicability to diverse network models. In addition, the k shortest path formulation and algorithm are proposed as a basis for network survivability measures where path length bounds exist.

Gabriel Y. Handler et al developed a Lagrangian relaxation algorithm for the problem of finding a shortest path between two nodes in a network, subject to a knapsack-type constraint. For example, we may wish to find a minimum cost route subject to a total time constraint in a multimode transportation network. Furthermore, the problem, which is shown to be at least as hard as NP-complete problems, is generic to a class of problems that arise in the solution of integer linear programs and discrete state/stage deterministic dynamic programs. One approach to solving the problem is to utilize a kth shortest path algorithm, terminating with the first path that satisfies the constraint. This approach is impractical when the terminal value of k is large. Using Lagrangian relaxation we propose a method that is designed to reduce this value of k. Computational results indicate orders of magnitude savings when the approach is applied to large networks.

Humblet, P.A. et al authors give a distributed algorithm to compute shortest paths in a network with changing topology. The authors analyze its behavior. The proof of correctness is discussed. It does not suffer from the routing table looping behavior associated with the Ford-Bellman distributed shortest path algorithm although it uses truly distributed processing. Its time and message complexities are evaluated.

III. OVERVIEW OF SOME ALGORITHMS

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959 [1], is a shortest path algorithm that solves the single-source shortest path problem for a graph with non negative edge path costs, outputting a shortest path tree. For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a

single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First) [4].

Illustration

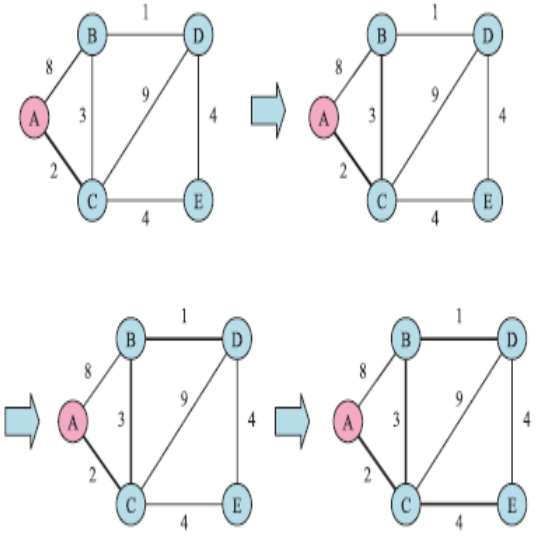


Fig-1.Dijkstra's algorithm which is based on sequential search

Applications

- Applied to automatically find directions between physical locations, such as driving directions on web mapping websites like MapQuest or Google Maps
- Shortest path algorithms can be used to find an optimal sequence of choices to reach a certain goal state
- Operations research, plant and facility layout, robotics, transportation, and VLSI design

Dijkstra's shortest path algorithm, which is the conventional one, finds the shortest paths from the source on a program counter-based processor. The calculation time for Dijkstra's algorithm is O ($nlogn$) when the number of nodes is n. When the network scale is large, calculation time required by Dijkstra's algorithm increases rapidly. It's very difficult to compute Dijkstra's algorithm in parallel because of the need for previous calculation results, so Dijkstra's algorithm is unsuitable for parallel processors.

So for finding the shortest paths in parallel a **new parallel shortest path searching** algorithm [3] developed in 2007 by Hiroyuki ISHIKAWA.

Parallel Shortest path searching algorithm (PSPS)

This algorithm finds the shortest paths using a simultaneous multi-path search method where several nodes can be determined at one time.

So for finding the shortest path in the large networks, we are partitioning the network into different groups (network groups) and find the all-node pair's shortest path in each group using a pipeline operation. Networks can be abstracted, and the shortest paths in very large networks can be found easily. The proposed algorithm can decrease calculation time from O (n^2) to O (n) using a pipeline operation on a structured parallel reconfigurable processor.

Illustrations

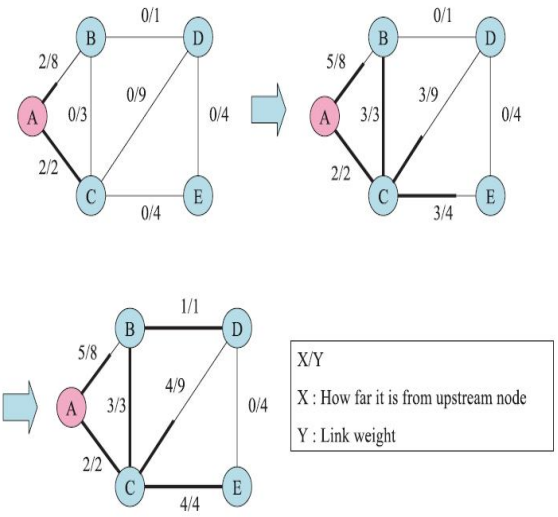


Figure-2 .Parallel shortest path searching algorithm based on multipath search

Applications

Shortest path algorithms are essential to improving efficiency and safety in many areas network navigation [8] such as highway systems and disaster clean-up

Algorithms are used for determining time dependent shortest paths in highway systems given variations in traffic density

Algorithms are used in connection with GPS tracking technology to provide shortest paths for trucks to follow to specific dumping areas when cleaning up after a natural or terrorism related disaster

Vehicles equipped with shortest path algorithms along with AVL technologies are faster, safer, and more efficient than those without

In the not to distant future most vehicles will be equipped with GPS tracking technology, but the real market change will be in the access and permission to use shortest path algorithms and AVL systems

The GPS-based Automatic Vehicle Location system was used for the first time to manage the debris removal in the disaster recovery setting at the World Trade Center site.

Enabled staff to locate each truck as it loaded materials at the site and transported them to disposal sites

Removing bottlenecks and placing debris removal resources where they were needed at the right time improved efficiency drastically

Many state and federal agencies have expressed their intent to incorporate this technology into future recovery efforts following such natural disasters as earthquakes, floods, and hurricanes.

IV. DETAIL STUDY AND ANALYSIS

Dijkstra's Algorithm Dijkstra's algorithm[4], conceived by Dutch computer scientist Edsger Dijkstra in 1959 [1], is a graph search algorithm that solves the single-source shortest path problem for a graph with non negative edge path costs, outputting a shortest path tree. This algorithm is often used in routing.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

Let $G = (V, E)$ be a directed graph where V is the set of nodes, and E is the set of edges. Let s be the source node, and w be a function assigning a non-negative valued weight to each edge of G. The cost of a link can be thought of as the distance between the two nodes. For each $v \in V$, $d(v)$ represents the cost of the shortest path from the source node s to v. The theoretically most efficient sequential shortest path algorithm is Dijkstra's algorithm. It calculates the shortest path between the source node and all other nodes. It is expressed as follows.

Illustration

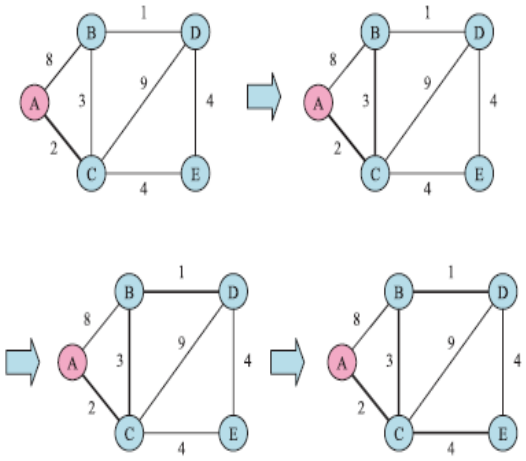


Fig-1: Dijkastra's algorithm which finds shortest path sequentially

Figure 1 shows an example of Dijkstra's algorithm. In this example, the network consists of five nodes. Small integers w (u, v) are assigned to link (u, v), with more preferred links being assigned smaller values than others. Link weights have been assigned symmetrically in this example, although they do not have to be. All nodes have infinite cost except the source. The source node is A and d (A) = 0. Set d (B) = 8 because there is a link between A and B as well as d(C) = 2. Choose the closest node C and add C to S. Update d (B) = 5, d (D) = 11, d (E) = 6 because they are connected with C. Now node B is the closest, add it to S. Update d (D) = 6, d (E) = 6. Then node D is the next closest node, choose it and add it to S. Update d (E) = 6. Finally, add node E to S and the task is completed.

Running time

An upper bound of the running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of $|E|$ and $|V|$ using the Big-O notation. The simplest implementation of the Dijkstra's algorithm stores vertices of set Q in an ordinary linked list or array, and operation Extract-Min (Q) is simply a linear search through all vertices in Q. In this case, the running time is $O(|V|^2+|E|) = O(|V|^2)$. For sparse graphs, that is, graphs with fewer than $|V|^2$ edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a binary heap, pairing heap, or Fibonacci heap as a priority queue to implement the Extract-Min function efficiently. With a binary heap, the algorithm requires $O((|E|+|V|) \log |V|)$ time (which is dominated by $O(|E| \log |V|)$ assuming every vertex is connected, that is, $|E| \geq |V| - 1$), and the Fibonacci heap improves this to $O(|E| + |V| \log |V|)$.

PARALLEL SHORTEST PATH SEARCHING ALGORITHM

The basic idea of this parallel shortest path searching algorithm [4] is simultaneous multipath search from the source node depending on the cost which determines the shortest path to all other nodes. Same pace will proceed in every edges connected to a particular node.

Let r(u,v)=how far it is from the current position to v in a link(u,v). Let three sets S,L,V where S=set of all visited nodes, L=set of links in which the current position proceeds .V=set of all the nodes.

Pseudo code

1. Function _parallel_search(graph,source);
2. For each vertex 'v' in graph
3. Set S=∅
4. s∈S //s=source node
5. d(s)=0,d(v)=∞
6. for all links l∈L
7. Take r(p,q) steps for all l∈L,where r(p,q) is the minimum value for (p,q)∈L
8. Add q to S and d(q)=d(p)+w(p,q)

9. Set r(u,v)=r(u,v)-r(p,q) for all (u,v)∈L
10. If S=V,complete the task
11. Delete (v,q) from L
12. if ther is a link from q to V-S,then go back to step-3.

Illustration

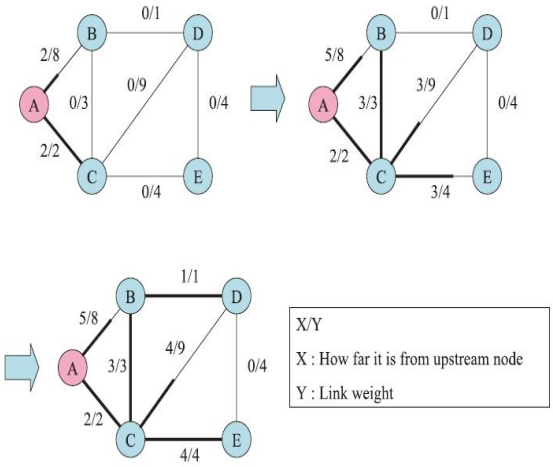


Fig-2: Parallel shortest path algorithm based on multipath search.

Figure 2 shows an example of our proposed scheme. In this example, the topology is the same as in Figure 1. The denominator represents w (u, v) and the numerator represents w (u, v) −r (u, v) in a link (u, v). First, the algorithm adds link AB, AC to L. For these links r (A, B) = 8 and r (A, C) = 2. The minimum value is 2 in this example, so link AC reaches node C. Then it adds C to S and d(C) = 2. Link AB proceeds two steps. It deletes AC from L and adds CB, CD, and CE to L. It updates r (A, B) = 6, r(C, B) = 3, r(C, D) = 9, and r(C, E) = 4. The minimum value is 3, so link CB reaches node B. It then adds B to S and d (B) = 5. Link AB, CD, and CE proceed three steps. It deletes AB and CB from L and adds BD to L. It updates r (B, D) = 1, r(C, D) = 6, and r(C, E) = 1. The minimum value is one, so link BD reaches node D, and link CE reaches node E simultaneously. It adds D and E to S and d (D) = 6, d (E) = 6. Link CD proceeds one step. The task is completed because S = V.

V. LIMITATIONS OF DIFFERENT ALGORITHMS

WHY BFS does not work for weighted Graphs?

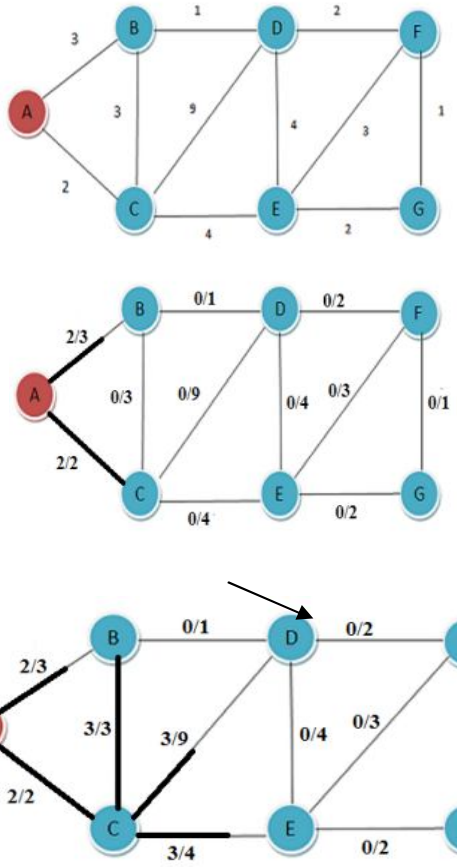
It is very chhalanging task to gurantee the node infront of the queue is the node closest to start node. It is not nearer in terms of weight of edges but in terms of number of connecting links used to reach. To overcome this difficulty if we use priority queue instead of queue we can easily get the vertices which are sorted in increasing distance value.

Why Dijkstra algorithm doesnot work?

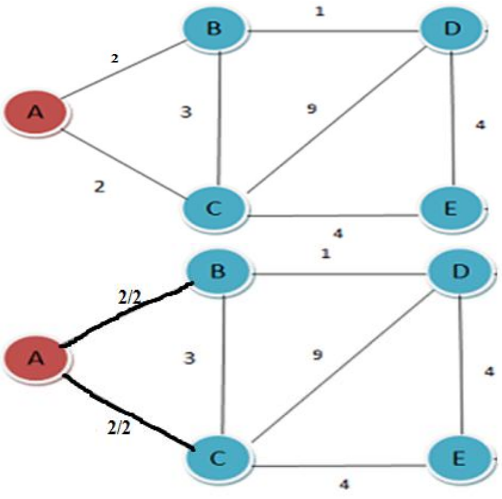
Dijkstar algorithm works when there is no negative weight, but it will not work for negative weights. Dijkstra and BFS algorithms are use the fact that the whole graph is known before. Dijkstra algorithm only search which has been seen before and it is not expecting the new strcture at latter stage which may affect to proceed. Bellman-Ford algorithm overcomes the shortcoming of BFS and Dijkstra algorithm.

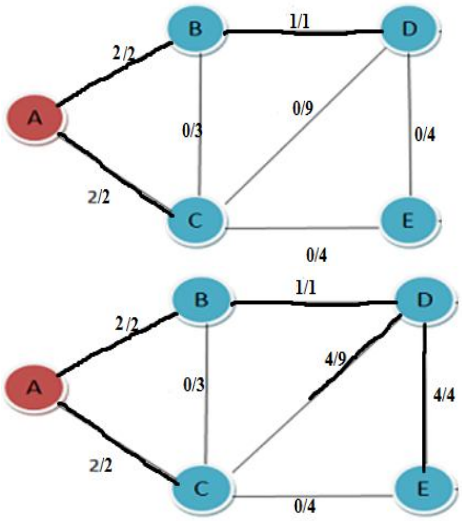
Why PSPS algorithm does not work?

It is found that this algorithm failed for finding shortest path in the folloging type of graph in parallel.It doesn't work in the line 7 of the algorithm.



If the two edges have same weights from the source node then which path we should go.





CONCLUSION AND FUTURE WORK

In this paper we have studied and analyzed different algorithms employed for parallel shortest path searching, and have identified various problems and limitations in different algorithms.

The inter-relationships between graph problems, software, and parallel hardware in the current state of the art are discussed and how those issues present inherent challenges in solving large-scale graph problems are identified. The Range of these challenges suggests a research agenda for the development of scalable high-performance software for graph searching problem. Basically our motivation to develop a more reliable and robust algorithm for parallel search in future

REFERENCES

[1] E. W. Dijkstra, (1959): *A note on two problems in connection with graph*

[2] Moshe Sniedovich, (2003): *Dijkstra's algorithm revisited: the dynamic programming connexion*: Department of Mathematics and Statistics The University of Melbourne, Australia.

[3] Gry J. Katz1, 2 and Joseph T. Kider Jr1 1University of Pennsylvania, 2 Lockheed Martinla :(2008) :*All-Pairs Shortest-Paths for Large Graphs on the GPU*

[4] Hiroyuki Ishikawa, Sho Shimizu, Yutaka Arakawa, Naoaki Yamanaka, Kosuke Shiba: (2007) :*New Parallel Shortest Path Searching Algorithm Based On Dynamically Reconfigurable Processor*

★★★