TRAVELING SALESMAN PROBLEM IN DISTRIBUTED ENVIRONMENT

Lau Nguyen Dinh and Tran Quoc Chien

University of Da Nang, Danang City, Vietnam
launhi@gmail.com
dhsp@dng.vnn.vn

ABSTRACT

In this paper, we focus on developing parallel algorithms for solving the traveling salesman problem (TSP) based on Nicos Christofides algorithm released in 1976. The parallel algorithm is built in the distributed environment with multi-processors (Master-Slave). The algorithm is installed on the computer cluster system of National University of Education in Hanoi, Vietnam (ccs1.hnue.edu.vn) and uses the library PJ (Parallel Java). The results are evaluated and compared with other works.

KEYWORDS

TSP, processor, parallel, distributed, cycle

1. INTRODUCTION

Traveling salesman problem (TSP) is a well known problem. The problem is solved in different ways. Especially in 1976, Nicos Christofides introduced new algorithms called Christofedes' algorithm [3]. In 2003, Ignatios Vakalis built Christofedes' algorithms on MPI environment [4]. In this paper, we build Christofides' traveling salesman problem in distributed environment. Sequential algorithms are built thoroughly with illustrative examples. In addition, parallel algorithms are experimented in different graphs.

2. CHRISTOFIDES' TRAVELING SALESMAN PROBLEM ALGORITHM

Let G=(V,E) be a graph and let P=V_1, V_2,…, V_k be a path in G. This path is called a Hamiltonian path if and only P is containing every vertex in V. P is a Hamitonian cycle if and only if V_1=V_k and P is a Hamiltonian path. Where G is a directed graph, the terms directed Hamiltonian path and directed Hamiltonian cycle are used. The problem of determining a shortest directed in a weighted directed graph G is called the Traveling Salesman Problem (TSP) [1].

Consider an n x n distance matrix D with positive entries; for example, the distance between the cities the traveling salesman is visiting. We assume D is symmetric, meaning that d_{ij}=d_{ji} for all i and j and d_{ii}=0 for i=1,2,…,n. We claim that [d_{ij}] satisfies the triangle inequality if

$$d_{ij}+d_{jk}\geq d_{ik} \text{ for all } 1\leq i,\ j,\ k \leq n \qquad (1)$$

What the triangle inequality constraint essentially says is that going from city i to city k through city j can not be cheaper than going directed from city i to city k. This is a reasonable assumption, sine the imposed visit to city j appears to be an additional constraint, meaning that can only increase the cost. As a rule of thumb, whenever the entries of the distance matrix represent cost, the triangle inequality is satisfied.

Notice that the graph in this variant of the problem undirected. If we remove any edge from an optimal path for such a graph, we have a spanning tree for the graph. Thus, we can use a algorithm to obtain a minimum spanning tree, then by going twice around the spanning tree, we can convert it to a path that visits every city. Recalling the transformation from Hamiltonian cycle to traveling salesman problem. Christofides [3] introduced a heuristic algorithm based on the minimum spanning tree for this problem.

Definition 2.1. Hamiltonian Cycle is a cycle in an undirected graph that passes through each node exactly once [7].

Definition 2.2. Given an undirected complete weighted graph, TSP is the problem of finding a minimum cost Hamiltonian Cycle [7].

Christofides' Traveling Salesman Problem (Algorithm 1)

Step 1: Find the minimum spanning tree T using the distance matrix D.

Step 2: Find the nodes of T having odd degree and find the shortest complete matching M in the completed graph consisting of these nodes only. Let G' be the graph with nodes {1,2,…,n} and edges in T and M.

Step 3: Find a Hamiltonian cycle in G'.

3.1: Find an Euler cycle C_0=(x,y,z,…,x) in G'.

3.2: Starting at vertex x, we trace C_0 and delete the vertex that has visited before in turn. Then remaining vertices, in the original order in C_0, determine a Hamilton cycle C, which is a required approximation optimal cycle.

The Prim's algorithm can be used in Step 1.

The number of odd-degree nodes in a graph is even. It's easy to see why this is the case: The sum of the degrees of all nodes in a graph is twice the number of edges in the graph, because each edge increases the degree of both its attached nodes by one. Thus, the sum of degrees of all nodes is even. For a sum of integers to be even it must have an even number of odd terms, so we have an even number of odd-degree nodes.

A matching is a subset of a graph's edges that do not share any nodes as endpoints. A perfect matching is a matching containing all the nodes in a graph (a graph may have many perfect matchings). A minimum cost perfect matching is a perfect matching for which the sum of edge weights is minimum. A minimum cost perfect matching of a graph can be found in polynomial time.

Finding a shortest complete matching in a graph is a version of the *minimal weight matching* problem, in which the total weight of the edges obtained from the matching is minimal. Edmonds and Johnson (1970) [5]; William Cook and André Rohe [6] have presented an efficient algorithm for finding minimum weight perfect in any weighted graph.

The Fleury's algorithm [19] can be used in Step 3.1 for finding Euler cycle.

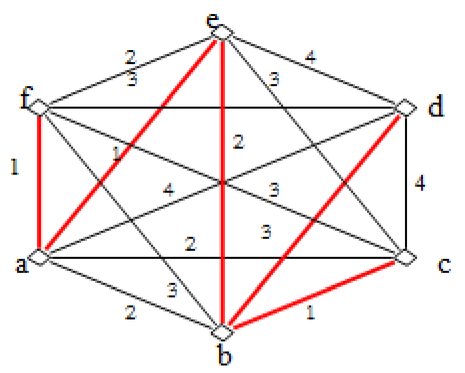


Figure 1. G(V,E) graph

$$D = \begin{Bmatrix} 0 & 2 & 2 & 4 & 1 & 1 \\ 2 & 0 & 1 & 3 & 2 & 3 \\ 2 & 1 & 0 & 4 & 3 & 3 \\ 4 & 3 & 4 & 0 & 4 & 3 \\ 1 & 2 & 3 & 4 & 0 & 2 \\ 1 & 3 & 3 & 3 & 2 & 0 \end{Bmatrix}$$

Figure 2. Distance matrix *D*

Determining whether a graph contains a Hamiltonian cycle is a computationally difficult problem. In fact, the fastest algorithm known has a worst-case time complexity of $O(n^2 2^n)$ in the case of n points. Therefore, the TSP exhibits an exponential worst-case complexity of $O(n^2 2^n)$. Proof [4].

Example: G(V,E) graph is illustrated in Figure 1

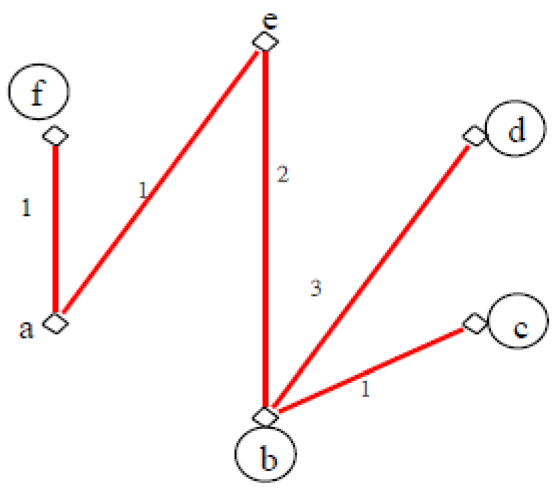


Figure 3. The minimum spanning tree T with the odd-degree vertices encircled

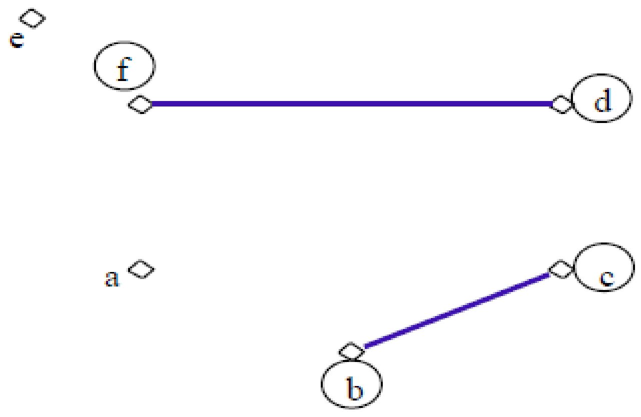


Figure 4. Shows the shortest complete matching M of these odd-degree verteces.

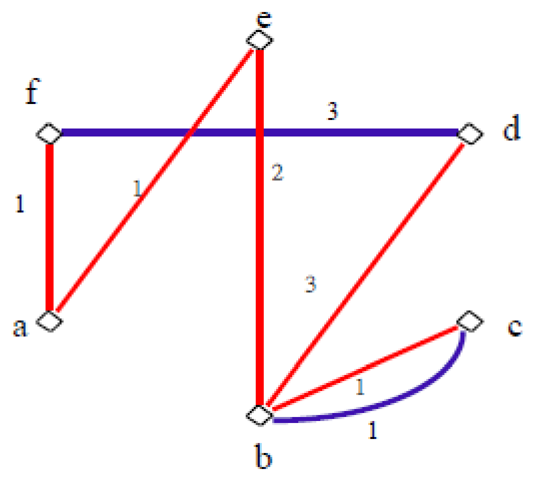


Figure 5. G'(V, E'), E' is edges in T and M

We have Euler cycle C_0 = (a, e, b, c, b, d, f, a). Deleting a repeated vertex b from C_0 results in a Hamilton cycle

C = (a, e, b, c, d, f, a) in G with w(C) = 12. Because the edge (c, d) of C is not in G', C corresponds a salesman route P = (a, e, b, c, b, d, f, a) with w(P) = 12 which visits each vertex of G at least once (Figure 6).

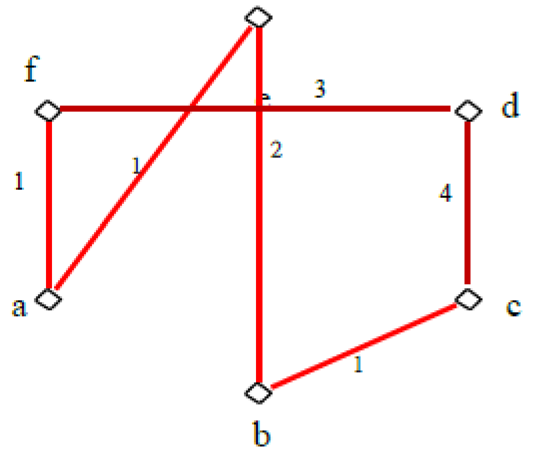


Figure 6. Traveling Salesman tour

For large n, the sequential version of a TSP algorithm becomes impractical. Thus, the need arises to examine a parallel approach in obtaining exact solutions to the TSP problem.

3. THE PARALLEL TRAVELING SALESMAN ALGORITHM

We carry out parallel algorithms on k processors. The parallel is performed in step 1 of the algorithm TSP. Slave processors perform to find MST T. Master processor performs step 2 and step 3 of the algorithm TSP.

Prim's algorithm (Algorithm 2):

```
Input: Let G(V,E) be a graph. V={1,2,…,n}
Output: Minimum Spanning Tree T(B, E') (T be a graph).
Step 1: Initialize T(1,∅). (B={1}, E'={∅})
Step 2: Condition to terminate.
        If T has n-1 edges, then T becomes Minimum Spanning Tree. Otherwise, then go to Step 3.
Step 3: (Addition)
        Symbol S is a set as following:
        $S=\{(i, j)\in E|i\in B \text{ and } j\notin B\}$
        Find edge $(u, v)\in S$ so that:
        $d_{uv}=\min\{d_{ij} \mid (i, j)\in S\}$
        If $d_{uv} <\infty$ then v is added to B and (u,v) to E', return to Step 2.  Otherwise, $S=\emptyset$ stop.
```

Parallel Traveling Salesman Problem algorithm (Algorithm 3)

```
Step 1: Create k numbers of Slave processes.
Step 2: Master node send n/k vertex and weight
        matrix D(n x n/k) to Slave.
Step 3: k Slave receives n/k vertex and D(n x n/k)
        from the master node.
Step 4: Master node performs:
        If T(B, E') has n-1 edges, then T becomes Minimum Spanning Tree.
        Otherwise, then go to Step 5.
Step 5: k Slave performs to find:
        - $S_p=\{(i, j)\in E_p|i\in B_p \text{ and } j\notin B_p\}$
        - Find edge $(u, v)_p \in M_p$ so that:
            $d_{uv}^{p} = \min\left\{d_{ij}^{p} \middle| (i,j) \in S_p \ (p=1,2,\ldots k)\right\}$    (2)
        - Send $d_{uv}^{p}$ to Master node.
Step 6: Master node finds $d_{xy}=\min\{ d_{uv}^{p} | \ p=1,2,\ldots k\}$
        If $d_{xy} <\infty$ then y is added to B and (u,v) to E',
        Master node sends y vertex and (x,y) edge to k Slave. return to Step 2.  Otherwise, stop.
Step 7: Master node receives T which is MST, then go to Step 8.
```

Step 8: Master node finds the shortest complete matching M

Step 9: Master node finds the Hamiltonian cycle in G'.

The main loop of the Prim algorithm is executed (*n-1*) times. In each n iteration it scans through all the *m* edges and tests whether the current edge joins a tree with a nontree vertex and whether this is a smallest edge found so far. Thus, the enclosed loop takes time $O(n)$, yielding the worst-case time complexity of the Prim algorithm as $O(n^2)$. Total parallel time $O(n^2/k + n \log k)$.

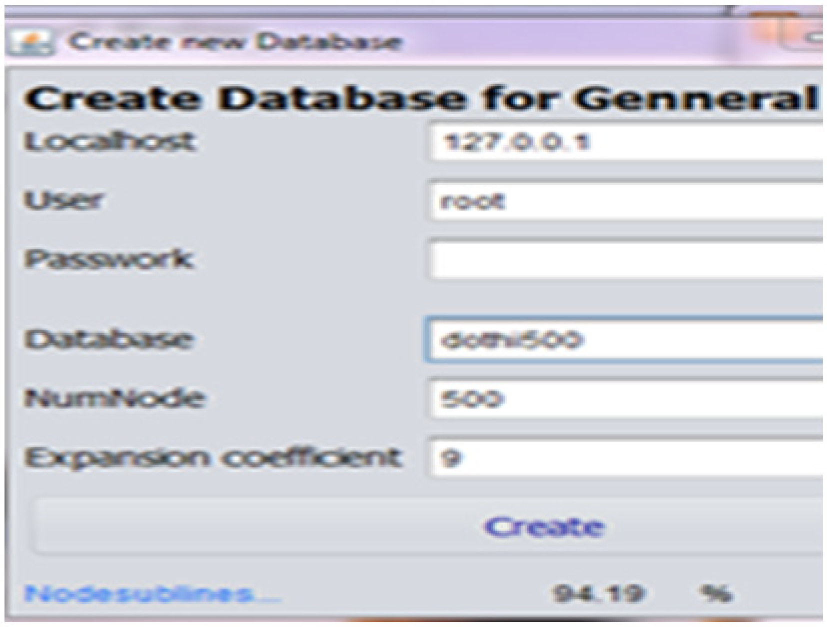
Therefore, algorithm 3 reduces more computation time than algorithm 1.

Parallel computing- Brief Overview:

The development of a wide range of parallel machines with large processing capacities high reliability, and low costs, have brought parallel processing into reality as an efficient way for implementing techniques to solve large scale optimization problems. A good choice of the programming environment is essential to the development of a parallel program.

The processes that are executed on parallel machines, are based on different memory organization methods: shared memory; or distributed memory. In shared memory machines, all processors are able to address the whole memory space. The processors can communicate through operations performed by the parallel tasks on the shared memory. Each task shares a common address space. The advantage of this approach is that the communication can be easy and fast. However, the system is limited by the number of paths between the memory and the processors.

An alternative to the shared memory organization is the distributed memory paradigm. In the framework of the distributed memory organization, the memory is physically distributed among the processors. Each processor can only access its own memory, and communication between processors is performed by messages passed through a communication network. A number of parallel programming tools are available to implement parallel programs for distributed memory environments.


Create new Database

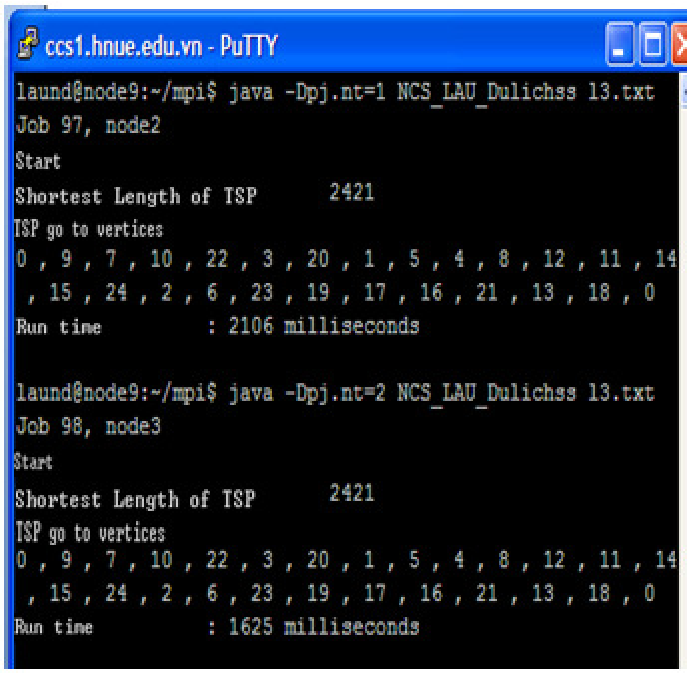
Create Database for Genneral

Localhost	127.0.0.1
User	root
Passwork	
Database	dothi500
NumNode	500
Expansion coefficient	9

Nodesublines... 94.19 %

Figure 7. Create database (Graph)

So, in this paper we choose the system's computing cluster of Hanoi National University of Education (ccs1.hnue.edu.vn) and use *Parallel java library*_PJ [8], [9].



ccs1.hnue.edu.vn - PuTTY

```
laund@node9:~/mpi$ java -Dpj.nt=1 NCS_LAU_Dulichss 13.txt
Job 97, node2
Start
Shortest Length of TSP      2421
TSP go to vertices
0 , 9 , 7 , 10 , 22 , 3 , 20 , 1 , 5 , 4 , 8 , 12 , 11 , 14
 , 15 , 24 , 2 , 6 , 23 , 19 , 17 , 16 , 21 , 13 , 18 , 0
Run time          : 2106 milliseconds

laund@node9:~/mpi$ java -Dpj.nt=2 NCS_LAU_Dulichss 13.txt
Job 98, node3
Start
Shortest Length of TSP      2421
TSP go to vertices
0 , 9 , 7 , 10 , 22 , 3 , 20 , 1 , 5 , 4 , 8 , 12 , 11 , 14
 , 15 , 24 , 2 , 6 , 23 , 19 , 17 , 16 , 21 , 13 , 18 , 0
Run time          : 1625 milliseconds
```

Figure 8. Parallel Computing Cluster (ccs1.hnue.edu.vn)

Parallel TSP algorithm is built on ccs1.hnue.edu.vn. The program written in Java and use *Parallel java library* (*PJ*). We experimentally sampled nodes as follows: The graph corresponds to 20000 nodes and 30000 nodes. The simulation result is shown in figure 9 and figure 10. This result demonstrates that the runtime of parallel algorithms is better than sequential algorithm.

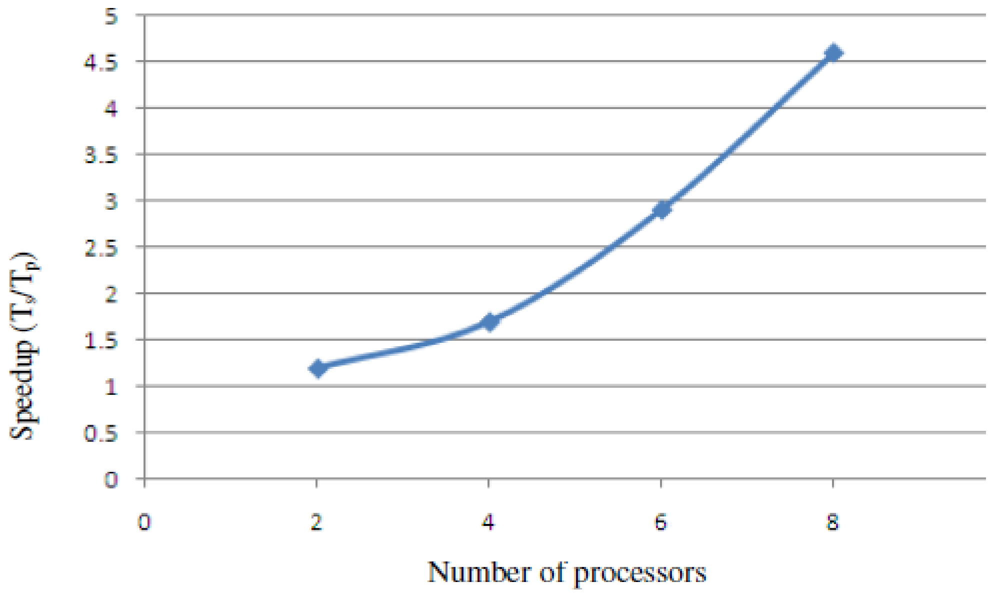


Figure 9. Chart performs the speedup of graph having 20000 nodes

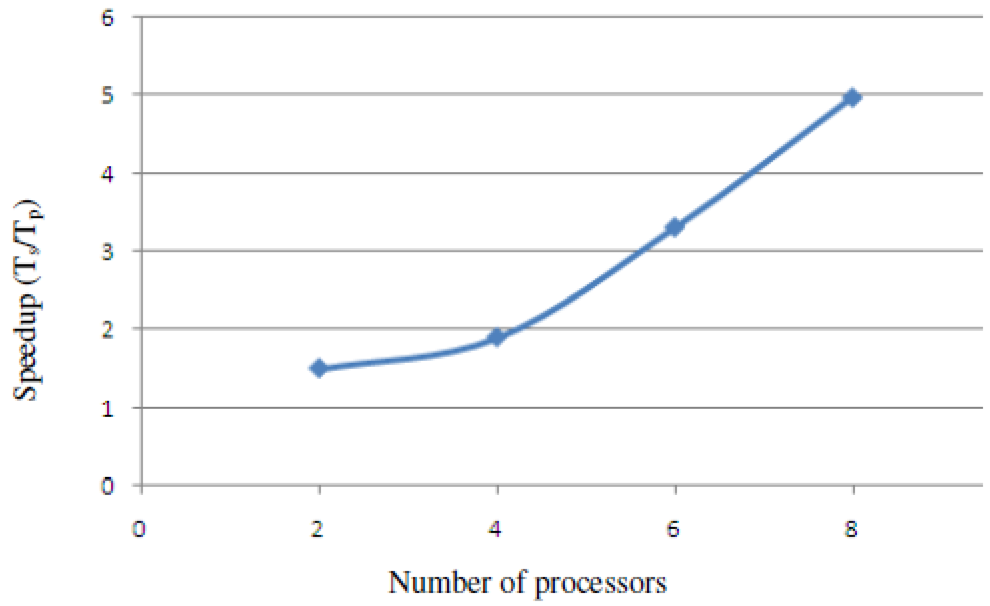


Figure 10. Chart performs the speedup of graph having 30000 nodes

4. CONCLUSION

The detail result of this paper is building sequential and parallel Traveling Salesman Problem algorithm.In addition, to take more advantage of multi-core architecture of the parallel computing system and reduce the computing time of this algorithm, we build this algorithm on multiple processors. Parallel algorithms in this paper are processed at step 1 of the algorithm 1. Ignatios Vakalis 2003 [4] built parallel algorithms by simultaneously looking for DFS (Depth First Search) in step 3 of algorithm 1 to resolve TSP. Random graphs (Figure 7) are created as our database to test the algorithms. As in [4] a small number of vertices graph (less than 12 vertices) are tested. Our algorithms are installed in computer cluster using Parallel Java (PJ) whereas in [4] using MPI. Therefore, our paper has made great contribution to building parallel algorithms using many different libraries.

REFERENCES

[1] Seyed H. Roosta, (1999) Parallel Processing and Parallel Algorithms, Theory and Computation, Springer.
[2] J. Little, K. Murty, D. Sweeney, C. Karel, (1963) “An algorithm for the traveling salesman problem”, Operations Research, No 11 , pp. 972–989.
[3] Nicos Christofides, (1976) Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem, Carnegie-Mellon University Management Sciences Research Report 388, Pittsburgh, PA.
[4] Ignatios Vakalis, (2003) Traveling Salesperson Problem: A Parallel Approach, NSF Grant No. 9952806, Computational Science Across the Curriculum Project, Capital University, 2003.
[5] J. Edmonds and E.L. Johnson, (1970) "Matching: a well-solved class of integer linear programs", in: Combinatorial structures and their applications (Gordon and Breach, New York, 89-92.
[6] William Cook, André Rohe, (1999) “Computing Minimum-Weight Perfect Matchings”, INFORMS Jounral on Computing, Voll.11, No.2, pp 138-148.
[7] Serge Plotkin, (2010) CS261 - Optimization Paradigms Lecture Notes for 2009-2010 Academic.

[8] Alan Kaminsky. (2007) “Parallel Java: A unified API for shared memory and cluster parallel programming in 100% Java”, 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, CA, USA.

[9] Jonathan Jude, (2008) “Fast Guide to using the RIT PJ Parallel Java Library: An Introduction to Java Parallel Programming using an API”, ISBN 978-3-8370-2439-5.

[10] Chien Tran Quoc, Lau Nguyen Dinh, Trinh Nguyen Thi Tu, (2013) “Sequential and Parallel Algorithm by Postflow-Pull Methods to Find Maximum Flow”, Proceedings 2013 13th International Conference on Computational Science and Its Applications, ISBN:978-0-7695-5045-9/13 $26.00 © 2013 IEEE, DOI 10.1109/ICCSA.2013.36, published by IEEE- CPS pp 178-181.

[11] Lau Nguyen Dinh, Thanh Le Manh, Chien Tran Quoc, (2013) “Sequential and Parallel Algorithm by Pre-Push Methods to Find Maximum Flow”, Vietnam Academy of Science and Technology AND Posts & Telecommunications Institute of Technology, special issue works Electic, Tel, IT; 51(4A) ISSN: 0866 708X, pp 109-125.

[12] Lau Nguyen Dinh, Chien Tran Quoc and Manh Le Thanh, (2014) “Parallel algorithm to divide optimal linear flow on extended traffic network”, Research, Development and Application on Information & Communication Technology, Ministry of Information & Communication of Vietnam, No 3, V-1.

[13] Lau Nguyen Dinh, Chien Tran Quoc, Thanh Le Manh, (2014) “Improved Computing Performance for Algorithm Finding the Shortest Path in Extended Graph”, proceedings of the 2014 international conference on foundations of computer science (FCS’14), July 21-24, 2014 Las Vegas Nevada, USA, Copyright © 2014 CSREA Press, ISBN: 1-60132-270-4, Printed in the United States of America, pp 14-20.

[14] Chien Tran Quoc, Thanh Le Manh, Lau Nguyen Dinh, (2013) “Sequential and parallel algorithm by combined the push and pull methods to find maximum flow”, Proceeding of national Conference on Fundamental and Applied Infromation Technology Research (FAIR), Hue, Vietnam, 20-21/6/2013.ISBN: 978-604-913-165-3, 538-549.

[15] Chien Tran Quoc, Thanh Le Manh, Lau Nguyen Dinh, (2013) “Parallel algorithm to find maximum flow costlimits on extended traffic network”, Proceeding national Conference XVI "Some selected issues of Information Technology and Communications” Danang 14-15/11/2013, ISBN: 978-604-67-0251-1, 314-321.

[16] Lau Nguyen Dinh, Tran Ngoc Viet, (2012) “Parallelizing algorithm finding the shortest paths of all vertices on computer cluster system”, Proceedings national Conference XVth "Some selected issues of Ìnormation Technology and Communications” Ha Noi, 03-04-2012, 403-409.

[17] Lau Nguyen Dinh, Tran Ngoc Viet, (2012) “A parallel algorithm finding the shortest paths of multiple pairs of source and destination vertices in a graph”, Journal of science and technology - University of DaNang 9 (58), pp. 30-34.

[18] Lau Nguyen Dinh, Tran Ngoc Viet, (2012) “Parallelizing algorithm dijkstra’s finding the shortest paths from a vertex to all vertices”, Journal of science, University of Hue, 74B, 5, pp. 81-92.

[19] Chien Tran Quoc, Graph algorithm: theory and application, 2007

AUTHORS

1. Dr. LAU NGUYEN DINH

Born in 1978 in Dien Ban, Quang Nam, Vietnam. He graduated from Maths_IT faculty of Hue university of science in 2000. He got master of science (IT) at Danang university of technology and hold Ph.D Degree in 2015 at Danang university of technology. His main major: Applicable mathematics in transport, parallel and distributed process, discrete mathemetics, graph theory, grid Computing and distributed programming.

2. Ass. Prof. DrSc. CHIEN TRAN QUOC

Born in 1953 in Dien Ban, Quang Nam, Vietnam. He graduated from Maths_IT faculty. He got Ph.D Degree of maths in 1985 in Charles university of Prague, Czech Republic and hold Doctor of Science in Charles university of Prague, Czech Republic in 1991. He received the tittle of Ass. Pro in 1992. He work for university of Danang, Vietnam. His main major: Maths and computing, applicable mathematics in transport, maximum flow, parallel and distributed process, discrete mathemetics, graph theory, grid Computing, distributed programming.