Fluid Flow Algorithm to Solve Travel Salesman Problem

Mudassir Khalil[1] Jian-Ping Li[2] Rafaqat Hussain[3] Kamlesh Kumar[4]

Abstract

Travel Salesman Problem (TSP) is well known combinatorial optimization problem. Hamiltonian path is required to solve TSP. This Hamiltonian path can be achieved by many methods. In this paper a novel and efficient algorithm is discussed. The name of algorithm is Fluid Flow algorithm. It works similar as flow of fluid when expands from a source. It starts from smallest triangle and expands by keeping a closed path and captures its nearest possible point. It continues expansion until all points included in that path. In this paper the efficiency of proposed algorithm is compared with other algorithms and experimental results have proved that our proposed algorithm is more efficient than Ant Colony and Genetic algorithms.

Keywords: Travel Salesman Problem (TSP), Big Data, TSP Big Instances, Fluid Flow Algorithm. Ant Colony Algorithm, Genetic Algorithm

1 Introduction

To solve Travel Salesman Problem (TSP) with given in- stances one has to pass all points of that instance and come back to the original point. The path taken should be the minimum from all possible paths. This is called Travel salesman problem (TSP). TSP is very famous optimization problem. In optimization problems best solution is needed to find from many available solutions. TSP is included in NP-Complete class [1]. NP-complete problem are those which cannot be solved in polynomial time by a deterministic algorithm, but a non-deterministic algorithm can do it in polynomial time [2]. TSP has a long history and always been an inspirational topic for researchers. The reason for it is that TSP problem is very easy to understand. This statement is true when given instance is small but when instance are too high, this problem becomes complex to solve and complexity increases exponentially. There are many algorithms to solve TSP. Some are exact and some are heuristic. Exact algorithms are those which can provide exact solution of given problem but the time taken to solve the problem is too high. Heuristic algorithms cannot always be exact but can provide near to exact solution. Heuristic algorithms are fast as compared to exact algorithm. For TSP heuristic algorithm save so much time as compared with exact algorithm, in some cases years can be saved by using heuristic algorithms. There are two types of TSP problems, symmetric TSP and asymmetric TSP [3]. Symmetric TSP have same path from one point to other and for its return, but asymmetric have different paths between two points. Here difference between paths is in terms of length of paths. In this paper we will deal only with symmetric TSP. There are many algorithms to solve symmetric TSP. In this paper a brand new algorithm which is called Fluid Flow (FF) to solve TSP fast and accurately is provided.

[1] University of Electronic Science and Technology China, Chengdu, China engr.mudassirkhalil@gmail.com
[2] University of Electronic Science and Technology China , Chengdu, China jpli2222@uestc.edu.cn
[3] Shah Abdul Latif University Khairpur, Pakistan rafaqat.arain@salu.edu.pk
[4] Sindh Madressatul Islam University Karachi, Pakistan kamlesh@smiu.edu.pk

Fluid Flow is a newly developed algorithm, very easy to understand and easy to implement. It starts by calculation of a small triangle. To get this triangle first calculate a distance vector between points of given problem. Then find a smallest value in the distance vector. This would be the smallest path between two points. Now calculate the nearest point from these two points and make a triangle by combining straight lines between these three points. This would be a desire triangle in given instance. After having triangle next is to find nearest point from its edges. When that point is found include that point into triangle path. That would become four edges path (quadrilateral). Again previous step is repeated with quadrilateral path and include nearest point into quadrilateral path. This step is repeated until all points are included in path. In the end resultant path would be the path which has all points in it. This is a brief introduction of fluid flow algorithm. The name fluid flow comes from fluids with similarity in movement when expands from a source.

2 Literature Review

There are many algorithms to solve TSP problem. From currently available algorithms very famous are, greedy or nearest neighbor, Christofides, K-OPT, Lin Kernighan and various branch-and-bound algorithms. Nearest neighbor or Greedy algorithm is simplest algorithm. This algorithm finds its neighbor which is near to current point and move towards that point. Greedy algorithm is dependent on its initial point, if start point changes result of greedy algorithm may be change also. Greedy algorithm is less efficient but very easy to implement [4]. Christofides Algorithm is approximate algorithm to solve TSP. It gives solution which is 3/2 factor of exact solution [5]. To solve TSP problem a C language program called Concorde is mostly used. This program can solve bigger instances in several minutes. Concorde given as one of the best exact TSP solver available so far while reviewing both heuristic and exact solutions of the TSP by Hahsler and Hornik In 2007 [6] [7]. K-opt mostly used technique for TSP problem. It can be 2-opt, 3-opt or can be more. Mostly used forms are 2-opt and 3-opt [8]. Lin-Kernighan uses 2-opt and 3-opt to solve TSP problem and is one of the best algorithm to solve TSP. This is because it is an adoptive algorithm. Lin-Kernighan decides at every step how many edges need to swap [9], [10], [11]. Genetic Algorithm and Ant Colony Algorithm both are heuristic methods to solve TSP problem very quickly [12], [13]. These algorithms do not find exact solution of problem but can find near to exact solution.

Ant colony algorithm works similar to ants, which always find smaller path between their food and nest.

3 TSP Proposed Algorithm

To solve TSP big instances FF algorithm work efficiently. This algorithm consists of two parts. First part divide problem in to smaller parts. Second part start form lowest possible triangle in all divided parts and expand all triangles by combining near points. This FF algorithm will help first part to join sub problems into big one. We take Euclidean distances to perform test results, which are given in examples below. Detail of algorithms is in following.

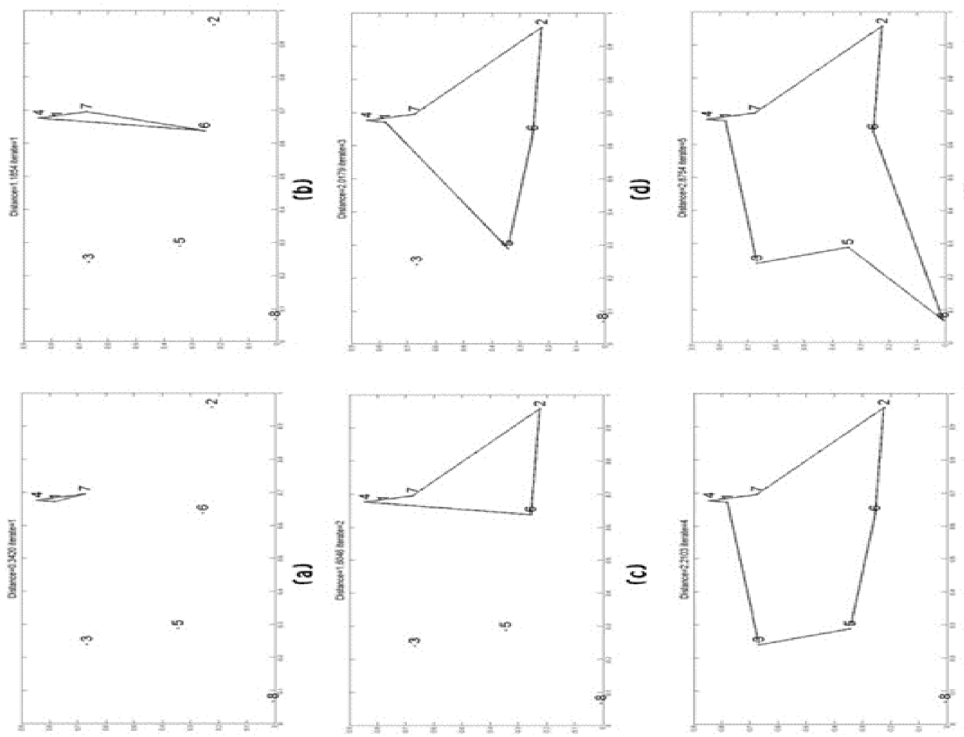

Figure 1: 8 points solution by use of FF algorithm

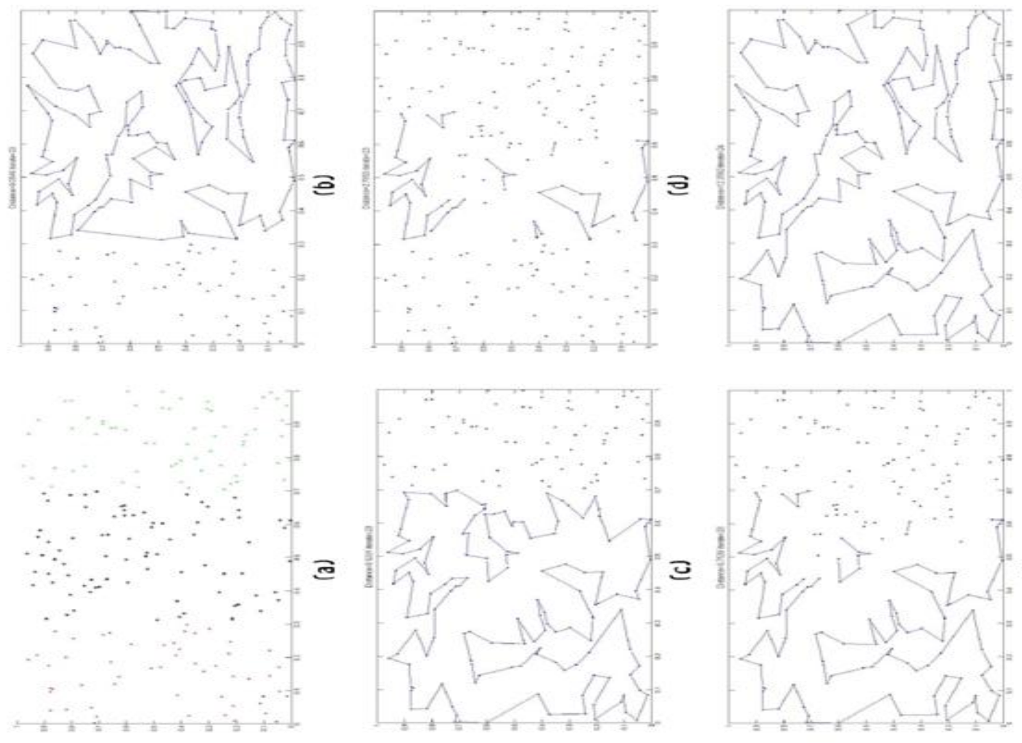

Figure 2: Solution of divided parts in FF algorithm

A. Fluid Flow (FF) Algorithm

In this paper Fluid Flow algorithm takes input which is given in TSP lib [14]. This input is Euclidean distance of cities. Every city is considered as point (x, y) on plan Algorithm first calculates Distance Matrix of all points using Eq 1.

$$d(x,y) = d(y,x) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2} \tag{1}$$

Here k can be any number greater than zero. Fig 1 explains the basic idea of fluid flow algorithm using 8 points example. In Fig 1(a) point 1, 4 and 7 which are close to each other combine to make a triangle. At this point path contains three points which are 1, 4 and 7. In Fig 1(b) closest point 6 to that triangle is included in to path. In Fig 1(c) (d) (e) points 2, 5 and 3 are included respectively into path. In Fig 1(e) last point which is point 8 is included in path as there is no point left so the path contains all point is solution of given problem. To find the nearest point form edge of path Specific distance Sd is used which is given in Eq 2.

$$rangeY = \frac{\max(Y) - \min(Y)}{10}$$

$$rangeX = \frac{\max(X) - \min(X)}{10}$$

$$S_d = \frac{rangeX + rangeY}{L} \tag{2}$$

In Fluid flow algorithm first there is division of bigger problem into small problems. For this Fig 2 procedure is used. In Fig 2(a) a big problem is divided in three parts which are displayed differently by design and colors. In Fig 2(b) first left and middle parts are solve by using procedure discussed with Fig 1 example. Similarly right and middle part is solve which is displayed in Fig 2(c). In Fig 2(d) intersection of Fig 2 (b) (c) is shown. This intersected path is combined with solution of left part which is shown in Fig 2 (e). In the end solution of right part is combined with result of Fig 2 (e). This gives final solution of problem given in Fig 2 (f). Following is a algorithmic form of Fluid Flow algorithm.

4 Algorithm

```
Input: C    // XY coordinates of cities.
Output:  Path     // Set of edges
Require: dMat           // Distance Matrix calculate using Eq. 1
Require: Sd // Specific Distance calculate using Eq.2 Path
            // Minimum cost triangle
while iterate == True do
   Cities = C - Path
   nearPoints = nearPoints (Cities, Path, Sd)
   for all nearPoints do
      for all Path do
            Dist  // minimum distance from path to   point
            if minDist > Dist then
                  minDist = Dist
            else
                  minDist = minDist
            end if
      end for
      Path = Path + minDist
   end for

   if path == C  then
      iterate = false
   end if
end while

function  nearPoints (Cities,P ath,Sd)
   for all Path do
      for  all  Cities do
            Distance    // Min distance from nearest point
            if Distance < Sd then
                  nearPoint = City
            end if
      end for
   end for
end function
```

5 Experimental Results

In these experiment author uses Genetic algorithm, Ant colony algorithm and FF algorithm because these all are heuristic algorithms. These experiments are performed on Matlab 2010 by use of 1.6 GHz core i5 machine have 4GB RAM. For Ant Colony test results there were 200 ants selected and 20 iterations are performed on each sample test. In table 1 execution time and length of final path for three algorithms are given on various instances. In table 2 approximation ratio of FF algorithm compares with Ant Colony and Genetic algorithms is given.

6 Conclusion and Future Work

FF algorithm is intelligent in its work, because it works similar to flow of fluid which always finds shortest path. Another advantage of proposed algorithm is, it is very easy to understand. FF algorithm works efficiently compare to other algorithms because in other algorithms solution is dependent on comparison with all other points. In FF algorithm this path finding is done with fraction of possible p3oints, which saved much of execution time. Experimental results prove this algorithm is more efficient than Genetic and Ant colony algorithm. FF algorithm is an approximate algorithm but not exact algorithm. In future work and another possible step would be added to divide the larger problem to multiple smaller problems, and at the end, results will be combined. This would be like divide and conquer algorithm. This division can be implemented on distributed system, which will increase efficiency of proposed algorithm.

Table 1: Results using FF Algorithm

Instance Name	FF		Genetic Algorithm		Ant Colony Algorithm	
	Speed (Sec)	Distance	Speed (sec)	Distance	Speed (sec)	Distance
berlin52	0.51	8506	8.34	8506	4.6	8986
ulysses22	0.08	77	6.3	77.3	1.588	77.22
eil51	0.52	460	8.5	436.6	4.6	535
eil76	0.676	595.7	10.04	556.5	8.12	694
kroA100	1.191	25639	12.2	22273	13.42	33894
kroA150	3.31	30591	28.5	44369	24.3	51453
kroA200	6.34	33318	21.4	35572	47.25	46815
	1.09	24541	12.5	23779	13.17	31538
kroB150	2.99	29675	17.06	28873	27.14	44712
kroB200	5.57	34295	20.57	36699	47.35	49817
kroC100	1.422	22904	12.2	23710	12.826	34333
kroD100	1.03	23662	12	21777	12.68	33263
kroE100	1.18	23956	12.4	23150	12.97	33589
pr1002	137	318986	167	670590	-	-
gr666	274	3780	40.7	4739	572	6395
rat575	65	8076	36	10464	371	14155

Table 2: Approximation ratio of FF Algorithm compares with Ant Colony and Genetic Algorithms

Instance Name	Oprx ratio vs Genetic Algorithm		Oprx ratio vs Ant Colony Algorithm	
	Speed	Dist	Speed	Dist
berlin52	1.930556	0.985061	1.064815	1.040649
ulysses22	5.833333	0.99217	1.47037	0.991144
eil51	3.373016	0.871457	1.825397	1.067864
eil76	2.73123	0.916804	2.208923	1.143328
kroA100	2.884161	0.838308	3.172577	1.275697
kroA150	2.141247	1.361388	1.825695	1.578749
kroA200	1.30967	0.953213	2.891677	1.254488
kroB100	2.948113	0.930284	3.106132	1.233833
kroB150	1.313318	0.941562	2.089299	1.458079
kroB200	1.32113	1.019558	3.041105	1.383998
kroC100	2.758933	0.990227	2.900498	1.433887
kroD100	3.053435	0.920336	3.226463	1.405756
kroE100	3.031785	0.966355	3.171149	1.402112
pr1002	1.113333	2.065986	-	-
gr666	0.108824	1.22772	1.529412	1.656736
rat575	0.202247	1.249284	2.08427	1.689947

Acknowledgements

This paper was supported by the National Natural Science Foundation of China (Grant No. 61370073), the National High Technology Research and Development Program of China (Grant No. 2007AA01Z423), the project of Science and Technology Department of Sichuan Province.

References

[1] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman &amp; Co., 1990.

[2] S. A. Mulder and D. C. Wunsch, II, "Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks," Neural Netw., vol. 16, no. 5-6, pp. 827–832, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1016/ S0893-6080(03)00130-8

[3] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," Operations Research Letters, vol. 2, no. 4, pp. 161 – 163, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167637783900482

[4] D. S. Johnson, Local optimization and the Traveling Salesman Problem. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 446–461. [Online]. Available: http://dx.doi.org/10.1007/BFb0032050

[5] M. Bl¨aser, K. Panagiotou, and B. V. R. Rao, A Probabilistic Analysis of Christofides' Algorithm.

[6] Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 225–236. [Online]. Available: http: //dx.doi.org/10.1007/978-3-642-31155-0 20

[7] M. Hahsler and K. Hornik, "Tsp – Infrastructure for the traveling salesperson problem," Journal of Statistical Software, vol. 23, no. 2, pp. 1–21, December 2007. [Online]. Available: http://www.jstatsoft.org/ v23/i02/

[8] S. A. Mulder and D. C. Wunsch, II, "Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks," Neural Netw., vol. 16, no. 5-6, pp. 827–832, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1016/ S0893-6080(03)00130-8

[9] K. Helsgaun, "An effective implementation of k-opt moves for the linkernighantsp," in Roskilde University, 2007. Case, 2006, p. 109.

[10] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," European Journal of Operational Research, vol. 126, pp. 106–130, 2000.

[11] H.-K. Tsai, J.-M. Yang, Y.-F. Tsai, and C.-Y. Kao, "An evolutionary algorithm for large traveling salesman problems," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 34, no. 4, pp. 1718–1729, Aug 2004.

[12] R. Baraglia, J. I. Hidalgo, and R. Perego, "A hybrid heuristic for the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 5, no. 6, pp. 613–622, Dec 2001.

[13] C. L. Valenzuela and A. J. Jones, "Evolutionary divide and conquer (i): A novel genetic approach to the tsp," Evolutionary Computation, vol. 1, no. 4, pp. 313–333, Dec 1993.

[14] Y. Zhou, "Runtime analysis of an ant colony optimization algorithm for tsp instances," IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 1083–1092, Oct 2009.

[15] "TSPLIB," 2008, [Online]. Available: http://comopt.ifi.uniheidelberg.de/software/TSPLIB95/