Shortest Path Computation in Large Graphs using Bidirectional Strategy and Genetic Algorithms

Shom C. Abraham
Manipal Institute of Technology

Girish Dutt Shukla
Manipal Institute of Technology

ABSTRACT

The shortest path problem in graphs is a fundamental optimization problem which has stimulated research for several decades. Numerous real-world applications are modeled as graphs and shortest path computation is a frequent operation performed on them. Many graphs happen to be very large like road networks or routing networks. Shortest path computation on them is a challenge because of the low performance due to its large nature. Already existing graph algorithms are not suitable for large graphs.

In this paper, an attempt is made to solve the problem of finding an efficient point-to-point shortest path algorithm for graphs of larger sizes. First run the A * algorithm with binary heap implementation from both the directions. The nodes extracted from both directions are saved and then genetic algorithm is used to find the shortest path. The bi-directional strategy reduces the search space and the genetic algorithm optimizes the search problem to give best result. The final results illustrates that this novel approach with the optimization strategies achieves high scalability and performance.

1. INTRODUCTION

Shortest path problems are one of the most fundamental combinatorial optimization problems with many applications. 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.

Several real world applications use graphs, some of which are very large. An example of a large graph is road network which contains thousands of nodes and millions of links. Conventional methods of least cost path discovery gives poor performance in such large graphs. Hence, the requirement is to design and implement an efficient point-to-point shortest path algorithm for large graphs.

2. BACKGROUND

2.1 Min Heaps

A min heap is a left complete binary tree which satisfies the property key (parent) ≤ key (child) for all the nodes. Due to this property, the node with the lowest key will always be present at the root of the tree. Hence, extraction of minimum key node is an O (1) operation. After the extraction of root node, the tree may not satisfy the min-heap property for which the non-leaf nodes would have to undergo percolate down operation. This operation is an O (n) operation.

2.2 Dijkstra’s Algorithm

Dijkstra’s algorithm is a solution to the single-source shortest path problem in graph theory. Here, all edges must have nonnegative weights. Input is a weighted graph G= {E, V} and source vertex vεV, such that all edge weights are nonnegative. Output is the lengths of shortest paths (or the shortest paths themselves) from a given source vertex vεV to all other vertices.

```
distance[source] ←0
for all vertex v εV–{source}
      do  distance[v] ←∞
SET←ϕ   (S, the set of visited vertices is initially empty)
QUEUE←V  (Q, the queue initially contains all vertices)

while QUEUE ≠ ϕ
do   u ← min distance(QUEUE,dist)
     SET←SETυ{u}
     for all vertex, v ε neighbors[u]
           do  if   distance[v] > distance[u] + weight(u, v)
               then      distance[v] ←distance[u] + weight(u, v)

return distance[ ].
```

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of O (|V|^2).The adjacency list with a binary heap or priority queue implementation will produce a running time of O (|E| log |V|).

2.3 A* Algorithm

A* algorithm is a most popular algorithm for path finding as it is like Dijkstra’s algorithm in finding the shortest path and like greedy best first search in using a heuristic to guide itself. It favours vertices that are close to starting point (like Dijkstra’s algorithm) as well as vertices that are close to the goal (like greedy best first search). If g(n) represents the exact cost of path from starting point to any vertex n, and h(n) represents the heuristic estimated cost from vertex n to the goal, then A* algorithm examines the vertex n that has the lowest $f(n)= g(n) + h(n)$.

2.4 Genetic Algorithms

Genetic algorithms aim to develop solutions to optimization problems using techniques which are inspired by natural evolution. This involves inheritance, mutation, selection, and crossover.

Algorithm is initialized with a set of solutions called population. Solutions from one population are taken and used to form a new population. It is expected to get a new population which will be better than the old one. Population which is selected to form new population (offspring) is selected based on its fitness. A fitness function is developed to calculate the fitness value.

This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied. Genetic Algorithm Operators:

1. Encoding of a chromosome
The chromosome represents the information about solution. The widely used encoding is a binary string.

2. Crossover
Crossover selects parent individuals and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent.

3. Mutation
After a crossover is performed, mutation takes place. Mutation changes randomly the new offspring.

Genetic Algorithm Pseudocode
Create initial population

Calculate the fitness value of each individual in the population

Do

1. Select individuals with high fitness values to reproduce
2. New generation is formed through crossover and mutation to get offspring.
3. Calculate the fitness value of the offspring
4. Replace the individuals with least fitness value of population with newly generated offspring

While the condition is satisfied.

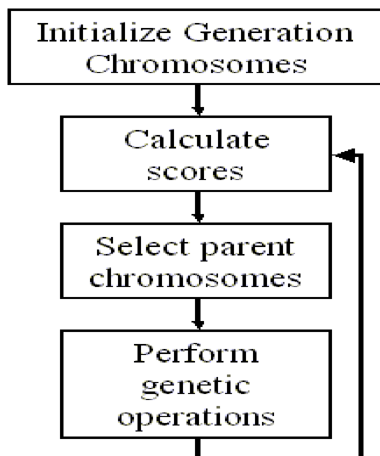

Fig 1: Flowchart of Genetic Algorithm

3. PROPOSED ALGORITHM

Since, the graph under consideration is a large one run the goal directed shortest path algorithm from both the directions (i.e. source and target). The following algorithm is run in each of the directions.

FUNCTION weight (u, v)

BEGIN.

1. new_weight = actual weight (u,v) + Heuristic value
2. return new_weight

END.

FUNCTION sp_algorithm(HEAP , SET)

BEGIN.

1. u ← min distance(HEAP)
   SET ←SET ʋ{u}
   for all vertex, v ε neighbors[u]
   do if distance[v] > distance[u] + weight(u, v)
   then distance[v] ←distance[u] +weight(u, v)
2. return SET.

END.

In the following function 'main' maintain two min-heaps, A and B for the two directions sorted by distances from source and target respectively. The nodes extracted from the min-heaps during the run of above algorithm are stored in sets A' and B' respectively. The process is not stopped until a common vertex v is found in both sets.

FUNCTION main

BEGIN.

1. Fill vertices in min-heap A sorted by distance from source.
   a. distance[source] ←0
   b. for all vertex v εV−{source}
      do distance[v] ←∞
   c. A ←V (A, the min-heap initially contains all vertices according to the min-heap property)
2. Similarly, fill vertices in min-heap B sorted by distance from target.
   a. distance[target] ←0
   b. for all vertex v εV−{target}
      do distance[target] ←∞
   c. B ←V (A, the min-heap initially contains all vertices according to the min-heap property)
3. Let A' and B' be the sets to hold the extracted nodes from the run of sp_algorithm. It is initially maintained to be empty.
   a. A'←ϕ
   b. ,B'←ϕ
4. REPEAT
   a. Call function sp_algorithm(A, A').
   b. Call function sp_algorithm(B, B').

   UNTIL (∃v AND vεA' AND v εB')
5. Call function genetic_algorithm(A', B').

END.

The shortest path from source, s to target, t does not necessarily run through the vertex v. It goes from something in A' to something in B'. Hence apply step 5 of the main function where the genetic algorithm is used to find the shortest path. The following section discusses the construction of genetic algorithm to achieve the purpose.

The chromosome is encoded by a string of positive integers that represent the IDs of nodes through which the path passes. Each position of the string represents an order of a vertex. Create an initial set of population with the chromosomes containing the vertices extracted in sets A' and B' as genes. Create x number of chromosomes randomly by selecting the genes from A' and B' as the shortest path from source to target will be a permutation of the vertices in sets A' and B' such that the cost function is minimized.

From the population randomly choose individuals for mating. Apply single point crossover to produce offspring from the parents which is best explained by the image below.

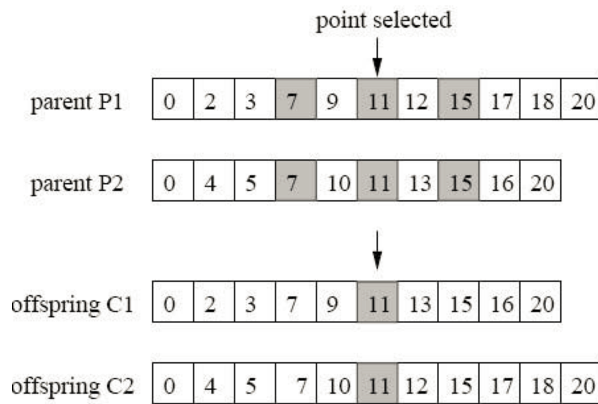

Fig.2 Single point crossover operation

If the offspring produced are having greater fitness value, then they replace the parents in the population. For mutation, wjust swap two genes, to ensure the genetic diversity in the population. This process of choosing parents for crossover, mutation and replacement with new children are continued until a chromosome containing genes which decodes to a path from source to destination is obtained. The following pseudo code summarizes the genetic algorithm applied to the above problem.

FUNCTION genetic_algorithm(SET A', SET B')

BEGIN.

1. Randomly pick vertices from A' and B' to generate x number of chromosomes.
2. do
   a. Pick two parents for mating.
   b. Apply single point crossover function.
   c. Compute the fitness value of children .
   d. If children have greater fitness value, replace the the chromosomes having lower fitness value in the population with them.

   While there exists a gene sequence from source to destination with the intermediate nodes connecting them.

END.

4. ANALYSIS AND RESULTS

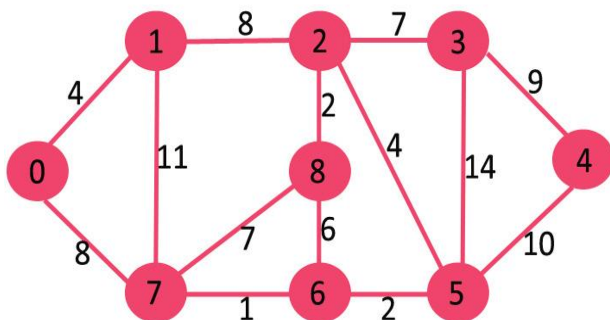

Fig.2 Graph example

Let's say the problem is to find shortest path from source vertex, 0 to a destination vertex, 4. On running sp_algorithm on the above graph, from both forward and backward directions the vertex sets A' and B' are obtained as below.

A'= [0, 1, 7, 6, 5]

B'= [4, 3, 5, 2, 6]

At the 5th iteration, the algorithm stops as it finds a vertex (5) as the condition ∃v AND vεA' AND v εB' is satisfied. Ignore all the vertices after the common vertex (5) in B' and create a random population out of the vertices 0, 1, 7, 6, 5 and 4, 3, 5 with 0 as the starting gene and 4 as the ending gene of the chromosome. Let's say the following two chromosomes represent a part of population.

Parent 1= [7, 0, 6, 5, 4, 3, 1]

Parent 2= [0, 7, 6, 5, 1, 3, 4]

Let's say, the algorithm chooses 6 as the crossover point, then the offspring created are

Child 1= [7, 0, 6, 5, 1, 3, 4]

Child 2= [0, 7, 6, 5, 4, 3, 1]

As the fitness value of Child 2 will be greater than those in the population (there exists a path sequence from source to target), the least promising individual will be replaced by child 2 in the population and that will be the final result. Hence, the shortest path is of distance 21 from 0-7-6-5-4.

As shown above, the genetic algorithm is guaranteed to find an optimal solution in some generation as it is inspired by the natural process of evolution.

5. CONCLUSION

This paper proposes an efficient point to point shortest path algorithm for large undirected graphs. The problem requirements have been met using modified version of Dijkstra's algorithm running it bi-directionally and using naturally inspired genetic algorithms for further steps. The speed factor is taken into account by using heap for the shortest path algorithm implementation. Experimental results ascertains that the proposed technique outperform commonly used Dijkstra's algorithm with adjacency matrix implementation qualitatively and quantitatively.

The work can be extended in future by replacing the current genetic algorithm strategy with much efficient crossover and mutation procedures to get optimal results.

6. REFERENCES

[1] Dijkstra, E. W. (1959), A Note on Two Problems in Connexion with Graphs, Numerishe Mathematic 1 , 269-271

[2] Shom C. Abraham. (2013) Least Cost Path Discovery over Graphs Defined for Large Volumes of Data Satisfying Node and Link Constraints. International Journal of Computer Applications 82(12):15-18.

[3] Bellman, R. (1958), On a Routing Problem, Quart. Appl. Math. 16, 87-90.

[4] M. Ericsson, M.G.C. Resende, and P.M. Pardalos , A Genetic Algorithm for the Weight Setting Problem in OSPF Routing

[5] Goldberg, D. E., Genetic Algorithms in Search ,Optimization, and Machine Learning, Addison-Wesley,

Reading, 1989

[6] Cai, X., Klocks, T. and Wong, C.K. (1997), Time-Varying Shortest Path Problems with Constraints, Networks, 29 , 141-149

[7] Dreyfus, S. E. (1969), An Appraisal of Some Shortest-Path Algorithms, Operations Research, 17, 395-412

[8] Floyd, R.W. (1962), Algorithm 97: shortest path. Comm. ACM 5345.

[9] Klein, P.N. and Subramanian, S. (1997), A Randomized Parallel Algorithm for Single-Source Shortest Paths, Journal of Algorithms, Vol. 25, No. 2, pp. 205-220.

[10] C. W. Ahn and R. S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, *IEEE Trans. Evol.Comput.*, vol. 6, no. 6, pp. 566–579, Dec. 2002.

[11] Cherkassky, B. V., Goldberg, A. V. and Radzik, T. (1996), Shortest path algorithms: Theory and experimental evaluation, Mathematical Programming 73, 129-174