COAR: Combining Arc flag with Reach based method for shortest path computation

Thangalatha Legaz C
Assistant Professor
Department of Information Technology
Sri Manakula vinayagar Engineering College, Puducherry
thangalatha@gmail.com

Abstract

In general shortest path computation from one node to another in a directed graph is a very common task. This problem is fundamentally solved by the Dijkstra algorithm. There are many techniques available to speed up the original Dijkstra’s algorithm heuristically. Even then the optimality of the solution can still be guaranteed. In this project, the various combinations of speedup techniques are studied and analyzed. By analyzing the advantages of the arc flag method and reach based method, the combination of the arc flag and the reach based method provides better speed up than the previous combinations and therefore we improve the speedup of shortest path queries using this combination. The main focus of the work is to implement the combination of Arc flag and Reach based method to improve the speedup of shortest path queries. The experimental results were compared and analyzed for determining better performance improvements in random graphs and planar graphs. The road network for South India with the distance values between districts in Tamilnadu are also tested using arcflag method, reach based method and COAR.

1. Introduction

The classical algorithm for computing shortest paths in a directed graph with nonnegative edge weights is that of Dijkstra’s algorithm. The Dijkstra’s algorithm implemented with Fibonacci heaps is still the fastest known algorithm for the general case of arbitrary nonnegative edge lengths, taking $O(m+n \log n)$ worst-case time. For special cases (eg. undirected graphs, integral or uniformly distributed edge weights) better algorithms are known.

Let $G = (V, E)$ be a directed graph whose edges are weighted by a function $W: E \to R$. The weights are interpreted as the edges’ lengths in the sense that the length of a path is the sum of the weights of its edges. The single-source single-target (SSST) shortest-path problem consists in finding a path of minimum length from a given source $s \in V$ to a given target $t \in V$. The problem is only well defined for all pairs, if $G$ does not contain negative cycles. In the presence of negative weights, but not negative cycles, it is possible, using Johnson’s algorithm, to convert in $O(nm + n^2 \log n)$ time the original edge weights $W: E \to R$ to non-negative edge weights $W': E \to R^0$ that result in the same shortest paths. Hence, it can be safely assumed that the edge weights are non-negative. It can also be assumed that for all pairs $(s,t) \in V \times V$, the shortest path from $s$ to $t$ is unique.

2. Related Works

2.1 Basic Speed up Techniques

In general Dijkstra algorithm is used for computing the shortest path. In order to improve the speedup of shortest path queries many speed up techniques are introduced. The main focus is to reduce the response time for on-line queries. In this sense, a speed-up technique is considered as a technique to reduce the search space of Dijkstra’s algorithm e.g. by using pre computed information or inherent information contained in the data. Actually, often the underlying data contain geographic information, i.e a layout of the graph is provided. Furthermore, in many applications the graph can be assumed to be static, which allows a pre-processing.

Goal-directed search: Goal directed search modifies the priority of active nodes to change the order in which the nodes are processed. This search adds to the priority dist(u) a potential ‘pt’ depending on the target t of the search. The modified priority of a node is therefore dist(v)+pt(v) [1]. With an appropriate potential the search can be directed to the target thereby reducing the running time and at the same time finding a shortest path [1].

Bidirectional search: Bidirectional search simultaneously performs two searches: a “normal”, or forward, starting at the source node, and reverse, or backward, starting at the destination node. The algorithm can be terminated when a node has been
labeled permanent by both the forward and backward searches. The shortest path can be obtained by node u with minimum value df(u) +db(u) i.e., combining the path obtained in the forward search with the one obtained in backward search

Multilevel approach: Multilevel approach which has a preprocessing step in which the input Graph G is decomposed into levels and enriched with extra edges which represent shortest path between certain nodes [4]. The decomposition of the graph depends on the selected nodes Si, at level i such thatS0:=V \supseteq S_1 \supseteq \ldots \supseteq S_i. The decomposition of the node sets can be based on various criteria. Selecting desired number of nodes with the highest degree in the graph, works out to be an appropriate criterion.

Shortest-path containers: These containers provide a necessary condition for each edge, whether or not it has to be respected during the search. More precisely, the set of all nodes that can be reached on a shortest path using this edge is stored. For each edge e, determine all nodes Se that are reachable via e on a shortest path. For each edge e, store a geometric object around Se. Consider only edges with target inside its geometric object.

Arc flag Method: Arc Flags partitions the node set in p regions with a function r: V -> {1,…, p}. Then an arc flag, i.e. a p-bit-vector where each bit represents one region is used as edge label. For an edge e, a region is marked in the p bit-vector of e if it contains a node v with v € S(e) where S(e) contains the edges (each edge is the beginning of any shortest path to at least one region) with flag vector set to true. The modified dijkstra algorithm with arc flags takes all those outgoing arcs of u into account where flag entry corresponding to the target region is true, and all other outgoing arcs will be ignored. Then the overall space requirement for the preprocessed data is \(4(p \cdot m)\). But an advantage of bit vectors as edge labels is the insight that the preprocessing does not need to compute all-pairs shortest paths.

Reach based Method: Reach based method [5] computes a centrality measure called ‘reach’. The definition of reach metric in reach based method is same as that weight function. A reach for a graph function G(V,E) is a function: E -> R mapping an edge e in its real value m(e). Reach based Dijkstra uses a function called test(v) before inserting the vertex into the priority queue. If test(v) returns true the vertex is inserted into the queue, otherwise the vertex is not inserted into the queue. The value of test(v) is found to be true if r(v,G)>=m(P) of r(v,G) >d(v,t), where P is computed path from s to v at the time of insertion into the queue and d(v,t) is the distance from vertex to target t, false otherwise.

The combination of the above mentioned speedup techniques are discussed below. It’s to be noted that Goal-directed search and shortest-path containers, as well as other approaches, are only applicable if a layout of the graph is provided. The multilevel approach and shortest-path containers both require preprocessing, calculating additional edges and containers, respectively.

The combination of the four techniques is very natural, since all of the techniques modify the search space of Dijkstra’s algorithm independently of each other; Goal-directed search directs the search space towards the target of the search by modifying edge lengths; bidirectional search maintains two search spaces; the multi-level graph approach runs a common Dijkstra’s algorithm on a subgraph of the augmented input graph; and with shortest-path containers, search space can be pruned by ignoring such edges that absolutely do not contribute to a shortest path.

An important point to be considered is whether the search space of a combination is better (i.e., smaller) than the one of a single speedup technique. Another point concerns the additional effort needed to reduce the search space. For example, with goal-directed search, edge weights have to be calculated during the search. This additional effort usually increases the running time per visited edge by a small constant factor. Considering a combination of techniques, the constant factor per edge is higher than with a single technique, so there is a trade-off between reduction of search space and the additional running time per edge in the search space.

The single techniques are, as mentioned above, heuristics in the sense that the reduction of the search space cannot be proved, in general. Hence, the same holds particularly for a combination of such techniques and the method of choice to prove the performance improvement is an extensive experiment study. Since even the single speedup techniques do not work equally well on all kinds of graphs, several graph types are to be considered.

3. Combining Speedup Technique

In this section, the key notion of combining each pair of techniques is outlined and it is noted that extending these to combinations, including three or all six techniques, are not difficult.

**Goal-Directed Search and Bidirectional Search:** Combining goal-directed and bidirectional search is not as obvious as it may seem. Simple application of a goal-directed search forward and backward yields a wrong termination condition. In certain situations the search in each direction almost reaches the sources of the other direction. This often results in a slower algorithm.

To overcome these deficiencies, it is preferable to use the very same edge weights \(l'(v, w) = l(v, w) - \lambda(v) + \lambda(w)\) for both the
forward and the backward search. With these weights, the forward search is directed to the target \( t \) and the backward search has no preferred direction, but favours edges that are directed towards \( t \). This proceeding always computes shortest paths, as an s-t path is shortest independent of whether \( l \) or \( l' \) is used for the edge weights.

**Goal-Directed Search and Multilevel Approach:**
The multilevel approach determines, for each query, a subgraph of the multilevel graph on which Dijkstra’s algorithm is finally run. The computation of this subgraph does not affect edge lengths and thus a goal-directed search can be simply performed on it.

**Goal-Directed Search and Shortest-Path Containers:** Similar to the multilevel approach, the shortest-path containers approach determines for a given query a subgraph of the original graph. Again, edge lengths are irrelevant for the computation of the subgraph and goal-directed search can be applied readily.

**Bidirectional Search and Multilevel Approach:**
A bidirectional search can be applied to the subgraph defined by the multilevel approach. The subgraph can be computed on the fly during Dijkstra’s algorithm: for each node considered, the set of necessary outgoing edges is determined. To perform a bidirectional search on the multilevel subgraph, a symmetric, backward version of the subgraph computation has to be implemented: for each node considered in the backward search, the incoming edges that are part of the subgraph have to be determined. Shortest paths are guaranteed, since bidirectional search is run on a subgraph that preserves optimality, and, by the additional edges, only contains supplementary information consistent with the original graph.

**Bidirectional Search and Shortest-Path Container:** In order to take advantage of shortest-path containers in both directions of a bidirectional search a second set of containers is needed. For each edge \( e \in E \), the set \( S_b(e) \) is computed containing those nodes from which a shortest path ending with \( e \) exists. For each edge \( e \in E \) the bounding box of \( S_b(e) \) is stored in an associated array \( C_b \) with index set \( E \). The forward search checks whether the target is contained in \( C(e) \), the backward search, checks whether the source is in \( C_b(e) \). It can be verified that by construction only such edges are pruned that do not form part of any partial shortest path and thus of any shortest s-t path.

**Multilevel Approach and Shortest-Path containers:** The multilevel approach enriches a given graph with additional edges. Each new edge \((u_1, u_2,...,u_k)\) in \( G \). Such a new edge \((u_1, u_2)\) is annotated with \( C(u_1,u_2) \), the associated bounding box of the first edge on this path. This consistent labelling of new edges, which represent shortcuts in the original graph, ensures still shortest paths.

**4. Problem Definition**

Let \( G = (V, E) \) be a weighted graph together with a weight function \( l : E \rightarrow R \). Call a set of nodes \( C \in V \) a container. A container \( C \) associated with an arc \((u, v)\) is called consistent, if for all shortest paths from \( u \) to \( t \) that start with the arc \((u, v)\), the target node \( t \) is in \( C \). Then use a function \( r : V \rightarrow \{1, \ldots, p\} \) that assigns to each node the number of its region, then for each arc \( a \), a vector \( f_a : \{1, \ldots, p\} \rightarrow \{\text{true, false}\} \) with \( p \) entries, each of which corresponds to a region. The vector \( f_a \) will be called the arc-flag vector. The entries in the vector will be called the arc-flags. For each arc \( a \), we set the arc-flag \( f_a(i) \) to true if \( a \) is the beginning of a shortest path to at least one node in region \( i \in \{1, \ldots, p\} \) and false otherwise.

The partitioned region obtained using arc flag method is used for the Reach based method. The partitioned region obtained using arc flag method is given as input for the reach based method where the partitioned region is considered to be a weighted graph with positive weights. In this method a non negative reach metric \( m : E \rightarrow R \) is computed at the preprocessing step. Then the reach of vertex \( v \) on a path \( P \) in the partitioned regions starting at vertex \( s \) and ending at vertex \( t \) is computed, i.e \( r(v,P) \), is \( \min\{m(s,v,P),(m,t,P)\} \) and the reach of \( v \) in partitioned region \( r(v,G) \), is the maximum value of \( r(v,Q) \) over all least-cost paths \( Q \) in \( G \) containing \( v \). In general a reach for a graph function \( G(V,E) \) is a function: \( E \rightarrow R \) mapping an edge \( e \) in its real value \( m(e) \).

After the Preprocessing step the test(v) function is called in the computation phase before inserting the vertex into the priority queue used in the dijkstra algorithm. If the test(v) returns true that vertex \( v \) is then inserted otherwise ignored. Therefore at the reach based method, only the partitioned regions obtained at the arc flag method is used which reduces the computation time for finding the shortest path for the given input graph and also the execution time for processing the queries.
5. COAR - Combining Arcflag with Reach based Method

**COAR – ALGORITHM**

1. Get the input graph G = (V,E)
   //Perform Arc flag method to obtain partitioned region
2. Partition the input graph into arcs using two level partitioning method
   // Assign flag vector for each arc
3. Assign flag values for each node a in the region
4. Flag value of ‘a’ is fa := true iff ‘a’ is the beginning of any shortest path to atleast one region
   else fa := false
5. Construct regions using the partitioned region by considering only the arcs for which flag value is true, ignore other regions
   //Perform Reach based method on the partitioned regions
6. Compute reach metric (Reach on a path R(P) and Reach on a graph R(G'))
7. R(P) := min \{ m(s,v,p), m(v,t,p) \}
   R(G') := max \{ r(v,Q) \} where Q in G'
8. Perform a test function test(v)
9. Use values in step 3 to obtain result for the test function
   Test (v) := True
   iff r( v, G') ≥ m(P) or r( v, G') ≥ d( v, t)
   Test (v) := False otherwise
   // Computation phase
10. For partitioned region, assign to every node a distance value.
11. Set it to zero for our initial node and to infinity for all other nodes.
12. Mark all nodes as unvisited. Set initial node as current. D[s] ← 0/source node
13. For each v ∈ V – {s}
   Do d[v] ← ∞, S ← Ø
   Q ← v // Q is priority queue maintaining v-s
14. For all neighbor nodes v of V
   set new_dist := dist(u) + w(u,v) {
     if new_dist < dist(v) {
       if dist(v) = ∞ then goto step 4 then
       If test(v) := true
       Insert neighbor node v in Q with priority new_dist
     else set priority of neighbor node v in Q to new_dist
       else set dist(v) := new_dist
     }
15. Repeat step 7 to step 9 until shortest path is obtained
16. Stop

The combination of arc flag and reach based method is done in such a way that first run a complete Arc flag method on a input graph G=(V,E), so that the graph is partitioned into regions and are assigned flag vector for each arc which is considered to be the preprocessing phase and those regions with flag vector set to true is considered and other regions are ignored in the computation phase. The partitioned region is obtained at the end of arc flag method is taken as input for reach based method. Next, the reach based method is applied on the partitioned regions obtained using arc flag method where the partitioned region alone is concentrated and the other regions are not taken into account, which reduces the computation time for finding the shortest path for the given input graph G.

6. Experimental Analysis

The Arc flag and Reach based method were implemented in Visual C++(2008) with the help of LEDA library version 6.2 (Library of Efficient Data Types and Algorithms) [10]. The graph and priority queue data structures as well as other utilities such precise time measurement function provided by LEDA were used in the implementation. The combination of Arc flag and Reach based method were also implemented so that the performance of the combination can be compared with the query performance of Arc flag and Reach based method individually. The code was compiled using Microsoft ® 32-bit Visual C++ Compiler (version 2008) and the experiments were performed on an Intel Core i5 machine (2.53 GHz) with 4 GB RAM running Windows 7 64-bit operating system.

The random and planar graph generators provided by LEDA were used for generating graphs on which the modules were tested. The number of vertices visited during the shortest path computation and runtime were measured and used as metrics for comparing with the Dijkstra’s Algorithm. The time required for preprocessing and shortest path computation was accurately measured by using the functionality offered by LEDA.

The South Indian map is used for testing the combination of Arcflag and Reach based method in order to test the improvement of speedup of shortest path queries. The several districts of tamilnadu and its distances values are used for testing the COAR.
6.1 Analysis of Speed-up

The Speed-up is computed for arcflag, reach based method and COAR. Here the Speed-up denotes the ratio between average runtime of the Speedup technique and the dijkstra’s average runtime. The effect of using Arcflag method and Reach based method during shortest path Computation on random graphs, planar graphs and road network is analysed below. The Combination of Arcflag with Reach based method is also tested for the same input of 10,000 nodes. The tabulated values for arcflag, reach based method and COAR are listed in Table 1. The speedup values with respect to runtime is tabulated in Table 1.

For the road network, the districts of tamilnadu are taken as nodes and their distance values as the edge weights. The node count of upto 150 districts are used as input for the road network to test for arcflag, reach based method and COAR.

Table 1. Comparison of speed-up on arcflag, reach based and COAR

<table>
<thead>
<tr>
<th>Input Graph</th>
<th>COAR</th>
<th>Arcflag</th>
<th>Reach based method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random graph</td>
<td>1.08</td>
<td>1.47</td>
<td>1.52</td>
</tr>
<tr>
<td>Planar graph</td>
<td>1.5</td>
<td>0.97</td>
<td>0.94</td>
</tr>
<tr>
<td>Road network</td>
<td>2.17</td>
<td>2.06</td>
<td>3.03</td>
</tr>
</tbody>
</table>

The effect of using arcflag, reach based and COAR for shortest path computation can be clearly understood from the Figure 1, 2, 3 which shows the speedup of arcflag method, reach based method and COAR.

Figure 1. Speedup on Random graph

Figure 2. Speed-up on Planar graph

Figure 3. Speed-up on road network

From figure 1, the Speed-up on random graph is better for reach based method and the speedup for arcflag and COAR on random graph is nearly equal. From figure 2, the Speed-up on planar graph is better for COAR and speedup for reach based and arcflag is nearly equal. From figure 3, the Speed-up on road network is better for reach based method and the speedup for COAR is nearly equal to arcflag method.
6.2 Analysis of Vertex visit count

The effect of Vertex visit count on arcflag method, reach based method and COAR is analyzed below. The speed-up with respect to vertex visit count is tabulated in Table 2. The Speed-up referred here is the ratio between average vertex visit count of speedup technique and the dijkstra’s average vertex visit count.

The Arcflag method, Reach based method and COAR are tested on Random and Planar graph with nodes of 10,000. For Road network, nearly 150 districts from tamilnadu are used.

Table 2. Comparison of Vertex visit count on arcflag, reach based and COAR

<table>
<thead>
<tr>
<th>Input Graph</th>
<th>COAR</th>
<th>Arcflag</th>
<th>Reach based method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random graph</td>
<td>1.14</td>
<td>1.04</td>
<td>1.91</td>
</tr>
<tr>
<td>Planar graph</td>
<td>1.5</td>
<td>1.58</td>
<td>0.22</td>
</tr>
<tr>
<td>Road network</td>
<td>1.28</td>
<td>1.26</td>
<td>2.07</td>
</tr>
</tbody>
</table>

The following figures show the Speed-up with respect to vertex visit count for arcflag method, reach based method and COAR with Dijkstra’s algorithm.

From figure 4, the number of vertices visited during shortest path computation is reduced better in the reach based method rather than the other two methods. The number of vertices visited in arcflag and COAR for shortest path computation is nearly equal.

Figure 5 shows the Speed-up of reach based method on planar graph does not have more effect on shortest path computation, whereas the arcflag method and COAR on planar graph for shortest path computation is nearly equal which shows the improved speed-up on dijkstra’s algorithm.

Figure 5, shows the Speed-up of arcflag method, reach based method and COAR on road network. The reach based method improves the speedup of shortest path computation better than the arcflag method and COAR. Also the number of vertices visited is reduced better in the reach based method. The speedup of 2.07 is obtained for reach based method on road network.

7. CONCLUSION

The existing speedup techniques though possess some advantages they also have some disadvantages which proves that the implementation of those technique with the road networks leads to only small variation in the runtime of online queries. The method of implementing the Arc flag without combining with any other speedup technique has the drawback: Partitioning of the node set plays a vital role in this
method, i.e., it purely depends on the partitioning strategy adapted. The additional space utilized for storing the arc flag vector is not always compromising. Reach based method, the parameters used for search is reach value and reach bound, i.e., the centrality measure of the vertex called ‘reach’. Because of this the search space is pruned. This method reduces the number of insertions of vertices into priority queue. The pre-processing time is very high. Therefore by combining the Arc flag method and Reach based method, the speedup of shortest path queries can be improved since both the techniques uses pre-processing step. Also the Arc flag method in this combination is implemented by using 2D grid method which is considered to be the efficient partitioning method.

The combination of Arc flag and Reach based method is implemented so that the partitioned regions obtained from the Arc flag method is given as input to the Reach based method where reach metrics are computed only for the regions with flag value true. Therefore this combination will improve the speedup of shortest path queries.

8. REFERENCES


