Effective All-Pairs Dijkstra’s Algorithm for Computing Undirected Weighted Graph Metrics

Jianzhong Wang
Department of Mathematics and Statistics
Sam Houston State University
Huntsville, TX 77341–2206, USA
Email: jzwang@shsu.edu

Abstract—The graph metric of an undirected graph can be represented by a symmetric matrix in which each entry is the graph distance between the corresponding nodes, i.e., the shortest path distance between them. This article presents an improved all-pairs Dijkstra’s algorithm for computing the graph metric on an undirected weighted graph. Taking the advantage of the symmetric property of the graph metric, the developed algorithm avoids the repeated computations in the construction of shortest path tree that produces the metric, so that it saves about a half computing time as compared with the regular all-pairs Dijkstra’s algorithm.

Index Terms—Graph, graph metric, shortest path tree, Dijkstra’s algorithm, all-pairs shortest path problem.

I. INTRODUCTION

In computer science, the geometric structure of a data set is often described by a data graph, whose nodes represent the elements (often called points) of the set and whose edges define a neighborhood system on the set. A weight matrix is often introduced on the data graph to produce a weighted graph, in which a weight is assigned on each edge of the graph. In many applications, the weight matrices are nonnegative so that its entries define edge lengths, which describe the dissimilarity of the data set: a larger weight on an edge indicates less similarity between the end nodes of the edge, and vice versa. Finding the shortest paths between nodes in a weighted graph is called the shortest path problem, which is very important in of the metric graph theory and other areas [1], [2]. According to different purposes, the problem has the following variations: The single-source shortest path problem, which finds the shortest paths from a source node to all other nodes; the single-destination shortest path problem, which finds shortest paths from all nodes to a destination node; and the all-pairs shortest path problem, which finds shortest paths between every pair of nodes.

In an undirected weighted graph, the length of the shortest path between two nodes is called the graph distance between the points, and the collection of graph distances of all pairs of nodes defines a graph metric, which describes a geometric structure on the node set. For instance, if the data set resides on a manifold, then the graph distance can be considered as the discrete approximation of the geodesic distance on the manifold and the graph metric is the discretization of the corresponding Riemannian metric on the set. Graph metric plays an important role in the applications such as object search, data mining, data classification, data dimensionality reduction, and data feature learning. For instance, the kernel in the Isomap algorithm for dimensionality reduction [3] is constructed based on graph metrics.

In the computation of graph metric, to calculate the graph distance between two nodes that are not directly connected by an edge is time-consuming for we have to compute all possible path distances and then minimize the results. An exhaustive iterative method known as Floyd’s algorithm [4] is optimal for computing the graph metrics of dense graphs. It runs in $O(n^3)$ for a graph having $n$ nodes. In information science, we often employ sparse graphs. The cost for computing metric on an undirected weighted sparse graph can be dramatically reduced. In 1956 Dijkstra introduced an algorithm [5], which is called Dijkstra’s algorithm later, to solve the single-source shortest path problem for a weighted graph. The algorithm produces a shortest path tree so that the shortest path-lengths computed in advance are reusable for computing the shortest path-lengths of new pairs. If the number of edges in a graph is $m$, based on a min-priority queue, the single-source Dijkstra’s algorithm runs in $O(n \log(n) + m)$ for solving the single-source shortest path problem [6]. This is asymptotically the fastest known single-source shortest path algorithm for arbitrary directed graphs with unbounded nonnegative weights. For a sparse graph, since $m = O(n)$, the algorithms runs only in $O(n \log(n))$. The single-source Dijkstra’s algorithm has also been applied to the computation of graph metrics in [3], [7], where the (regular) all-pair Dijkstra’s algorithm simply runs the single-source one over all nodes, so that it runs $O(n^2 \log(n))$ for the computation of graph metric.

Notice that the weight matrix of an undirected graph is symmetric. The repeated computations exist when we run the regular all-pair Dijkstra’s algorithm for an undirected graph. In this paper, taking the advantage of the symmetry, we present an improved all-pair Dijkstra’s algorithm for computing the graph metric on an undirected graph. The new algorithm avoids all repeated computation so that it takes only a half of computing time as compared with the regular all-pair Dijkstra’s algorithm.

II. GRAPH METRICS

Let $G = [\mathcal{X}, E]$ be a graph associated to data set $\mathcal{X} = \{x_1, \cdots, x_n\}$, where $E$ is the edge set, which defines a...
neighborhood system on $\mathcal{X}$ such that the neighborhood of the node $\vec{x}_i$ is the set

$$N(i) = \{ \vec{x}_j : (i,j) \in E \}.$$ 

The adjacency matrix $A = [a_{ij}]$ of $G$ is defined as follows.

$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}.$$ 

Therefore, we have $N(i) = \{ \vec{x}_j : a_{ij} = 1 \}$. It is clear that adjacency matrix can replace edge set in the graph notation. A graph $G = [\mathcal{X}, A]$ is undirected if and only if $A$ is symmetric. If a graph $G = [\mathcal{X}, A]$ is equipped with an $n \times n$ weight matrix $W = [w_{ij}]$, we call it a weighted graph. In computer and information sciences, weight matrix is often used to define the lengths of edges, which measure the dissimilarity on the data set. Assume that $w_{ij}$ is nonnegative, possibly taking the infinity value: $w_{ij} \in \mathbb{R}^+ \cup \{ \infty \}$. Assume that it also satisfies the following conditions:

- $w_{ij} = 0$ if and only if $i = j$,
- $w_{i,j} = \infty$ if and only if $a_{ij} = 0, i \neq j$.

Then we call $w_{ij}$ the length of the edge $(i,j)$, provided $(i,j) \in E$. A weighted graph is denoted by $G = [\mathcal{X}, A, W]$. When $W$ is symmetric, the weighted graph $G$ is undirected. Otherwise, $G$ is directed. In a directed graph, the length of $(i,j)$ may not equal the length of $(j,i)$. The details of weighted graphs are referred to [8], [2].

In many applications, the weighted graph $G = [\mathcal{X}, A, W]$ on a given data set $\mathcal{X}$ is constructed as follows. Let each node in the set $\mathcal{X}$ be embedded in a $D$-dimensional vector space $\mathbb{R}^D$ so that the set $\mathcal{X}$ is shown as a point cloud in $\mathbb{R}^D$. Let $d$ be a distance function on $\mathbb{R}^D$, which is usually chosen as the Euclidean distance, but other distances can also be applied to meet certain purposes. (See [9, Chapter 3].) We denote by $d(\vec{x}_i, \vec{x}_j)$ the distance between two nodes $\vec{x}_i, \vec{x}_j \in \mathcal{X}$. Then a neighborhood system on $\mathcal{X}$ can be constructed by using either the $e$-neighborhood method or the $k$-neighborhood one. If the $e$-neighborhood method is applied, the neighborhood of a point $\vec{x}_i \in \mathcal{X}$ is the subset of $\mathcal{X}$ defined by

$$N(i) = \{ \vec{x}_j : d(\vec{x}_i, \vec{x}_j) < e \}.$$ 

If the $k$-neighborhood method is applied, the neighborhood of $\vec{x}_i$ contains the $k$ nearest points of $\vec{x}_i$, excluding itself. Let the $k$-neighborhood of $\vec{x}_i$ be denoted by $N(i)$. Then

$$\max_{\vec{x} \in N(i)} d(\vec{x}_i, \vec{x}) \leq \min_{\vec{x} \notin N(i), \vec{x} \in \mathcal{X}} d(\vec{x}_i, \vec{x}).$$

In both cases, the weight matrix $W = [w_{ij}]$ is given by

$$w_{ij} = \begin{cases} d(\vec{x}_i, \vec{x}_j) & \vec{x}_j \in N(i) \\ \infty & \vec{x}_j \notin N(i), i \neq j \\ 0 & i = j \end{cases}$$

and the adjacency matrix $A$ becomes the sigh matrix of $W$:

$$a_{ij} = \begin{cases} 0 & w_{ij} = 0 \text{ or } \infty \\ 1 & \text{otherwise} \end{cases}.$$ 

In this paper, we always assume that $W$ defines the edge lengths on $G$ as described in (1) so that the adjacency matrix of $G$ can be derived by (2). The weight matrix created by $e$-neighborhood method is symmetric. However, if it is created by $k$-neighborhood, it may be asymmetric. We prefer to construct a symmetric weight matrix on a data set since it is convenient for the spectral analysis. In order to force the weighted graph created by $k$-neighborhood to be undirected, we often replace $W$ by $\min(W, W^t)$, $\max(W, W^t)$, or $\frac{1}{2}(W + W^t)$. Either $e$-neighborhood method or $k$-neighborhood method produces a sparse adjacency matrix $A$. We call a graph sparse if its adjacency matrix is sparse.

On graph $G = [\mathcal{X}, A, W]$, let $\vec{b} \in \mathcal{X}$ be path-connected to $\vec{a} \in \mathcal{X}$. Assume $P = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)\}$ is a path from $\vec{a}$ to $\vec{b}$, where $\vec{x}_{i_0} = \vec{a}$, $\vec{x}_{i_k} = \vec{b}$, and all pairs $(i_s, i_{s+1}), 1 \leq s \leq k$, are elements of the edge set $E$ (i.e., $a_{i_s, i_{s+1}} = 1$). Then the path length of $P$ is $l(P) = \sum_{s=1}^{k} w_{i_s, i_{s+1}}$. We denote by $l(\vec{a}, \vec{b})$ the length of the shortest path from $\vec{a}$ to $\vec{b}$. If the graph is undirected, then the length of a path from $\vec{a}$ to $\vec{b}$ is equal to the length of its inverse path from $\vec{b}$ to $\vec{a}$. We then call it the path distance between $\vec{a}$ and $\vec{b}$ and denote it by $d(P)$. Correspondingly, the shortest path distance between $\vec{a}$ and $\vec{b}$ is called the graph distance between $\vec{a}$ and $\vec{b}$ and denoted by $d_G(\vec{a}, \vec{b})$. If $\vec{b}$ is disconnected to $\vec{a}$, then we define $d_G(\vec{a}, \vec{b}) = \infty$. The graph metric on an undirected graph $G = [\mathcal{X}, A, W]$ is the $n \times n$ symmetric matrix $M = [d_G(\vec{x}_i, \vec{x}_j)]$. We have to point out that the path lengths and path distances here may not satisfy the following distance axiom:

$$l(\vec{a}, \vec{c}) \leq l(\vec{a}, \vec{b}) + l(\vec{b}, \vec{c})$$

and

$$d_G(\vec{a}, \vec{b}) \leq d_G(\vec{a}, \vec{c}) + d_G(\vec{c}, \vec{b}).$$

This fact makes the shortest path search problem more time-consuming.

III. DIJKSTRA’S ALGORITHM

In this section, we briefly introduce the regular all-pairs Dijkstra’s algorithm for computing graph metrics. Let $G = [\mathcal{X}, A, W]$ be a given undirected graph. For convenience, we assume that the graph is connected so that the graph metric is bounded. To compute the graph distance between two nodes that is not directly connected by an edge, we need to compute all possible path distances and then minimize the results. An exhaustive iterative method known as Floyd’s algorithm [4] is optimal for computing the graph metric on dense graphs. Floyd’s algorithm runs in $O(n^3)$. For sparse graphs, the computing costs can be reduced by using certain shortest path reaching trees. In literature, there have been developed several such algorithms, among which Dijkstra’s algorithm [5], conceived by Dijkstra in 1959, has been widely adopted. Dijkstra’s algorithm was originally developed for solving the single-source shortest path problem for a directed weighted graph. To save the computing time, it produces a shortest path.
tree so that the shortest path lengths computed in advance are reuseable for computing the shortest path lengths of new pairs.

Let \( G = (X, A, W) \) be a given weighted graph (not necessarily undirected) and \( \bar{a} \in X \) be a fixed node. We now describe the single-source Dijkstra’s algorithm for computing the shortest path lengths from \( \bar{a} \) to all other nodes in \( G \). The algorithm employs two functions on the tensor product set \( X \times Z_0 \), where \( Z_0 \) is the set of discrete times with the initial time \( t = 0 \). The first function is the length function \( l(\bar{x}, k) \), which records the tentative path length from \( \bar{a} \) to \( \bar{x} \) at the time \( k \) and its initial value is assigned to

\[
l(\bar{x}, 0) = \begin{cases} 0 & \bar{x} = \bar{a} \\ \infty & \bar{x} \neq \bar{a} \end{cases}.
\]

The second function is called the state function, which records the state of \( \bar{x} \) at the time \( k \). When the algorithm is running, each node has to have one of three states: unlabeled, labeled and scanned [4]. At each step, unlabeled nodes are not in the computational processing, scanned nodes are already out of the processing, and only labeled nodes are in the processing. Hence, for the ternary function \( s(\bar{x}, k) \), we may choose its range to be the set \( \{-1, 0, 1\} \), where \( -1 \) is used to record the unlabeled state, \( 1 \) used to record the scanned state, and \( 0 \) used to record the labeled state. At time \( k > 0 \), \( s(\bar{x}, k) \) is evaluated as the following.

\[
s(\bar{x}, k) = \begin{cases} -1 & \text{if } l(\bar{x}, k) = \infty \\ 1 & \text{if } l(\bar{x}, k - 1) = l(\bar{a}, \bar{x}) \\ 0 & \text{otherwise} \end{cases}
\]

and its initial value is

\[
s(\bar{x}, 0) = \begin{cases} 0 & \bar{x} = \bar{a} \\ -1 & \bar{x} \neq \bar{a} \end{cases}.
\]

The single-source Dijkstra’s algorithm runs iteratively, updating the functions \( l(\bar{x}, k) \) and \( s(\bar{x}, k) \) at each time-step. More precisely, while \( \sum_{\bar{x} \in X} s(\bar{x}, k) < n \), it takes the following actions at the time \( k + 1 \).

1) Set \( V(k) = \{ \bar{x} : s(\bar{x}, k) = 0 \} \)

and find \( \bar{a}_k \in V(k) \) that has the minimal length in \( V(k) \):

\[
l(\bar{a}_k, k) = \min_{\bar{x} \in V(k)} l(\bar{x}, k).
\]

Denote by \( N(a_k) \) the neighborhood of \( a_k \) and set

\[
s(\bar{a}_k, k + 1) = 1.
\]

2) For \( \bar{x} \in V(k) \) with \( s(\bar{x}, k) \neq 1 \), set

\[
s(\bar{x}, k + 1) = 0
\]

and

\[
l(\bar{x}, k + 1) = \min(l(\bar{x}, k), l(\bar{a}_k, k) + l(\bar{a}_k, \bar{x})).
\]

3) For other nodes in \( X \), no changes are made. That is, set

\[
l(\bar{x}, k + 1) = l(\bar{x}, k) \quad \text{and} \quad s(\bar{x}, k + 1) = s(\bar{x}, k).
\]

The algorithm stops when it reaches \( \sum_{\bar{x} \in X} s(\bar{x}, m) = n \). Its pseudocode is presented in the following.

```plaintext
1 function ss_Dijkstra(G, source)
2   for each vertex v in G:
3       length[v] := infinity;
4   end for;
5   length[source] := 0;
6   while G is not empty:
7       u := node in G with smallest length;
8       if length[u] = infinity:
9           break;
10      end if;
11     remove u from G;
12     for each neighbor v of u in G:
13       tl := length[u] + length(v, u);
14       if tl < length[v]:
15           length[v] := tl;
16       end if;
17     end for;
18   end while;
19   return length[];
20 // end of ss_Dijkstra.
```

An illustration of the single-source Dijkstra’s algorithm runtime and the Matlab code of the algorithm can be found in [9, Section 8.3].

All-pairs Dijkstra’s algorithm computes the shortest path lengths between all pairs of a given weighted graph, which is either undirected or directed. The regular all-pairs Dijkstra’s algorithm simply applies the single-source algorithm over all nodes in the set \( X \): After the shortest path lengths from a source node to all nodes in \( X \) are calculated, the algorithm shift to the next node in the set \( X \) and do the similar computation, until all shortest path lengths are calculated. A more general all-pairs shortest path problem is to compute all shortest path lengths from the nodes of a source set \( S \) to the points of a target set \( T \). Then the output is a \( |S| \times |T| \) matrix. The algorithm for solving the general problem essentially has no difference with the algorithm above. A Matlab code for the all-pairs Dijkstra’s algorithm, written by M. G. Kay at North Carolina State University in 2000, can be found in the author’s web-site [10].

IV. IMPROVED DIJKSTRA’S ALGORITHM FOR COMPUTING UNDIRECTED GRAPH METRICS

As mentioned in Section II, in many applications, the data graphs are undirected weighted sparse graphs. To compute the graph metric on an undirected graph, we may take the advantage of its symmetry to avoid repeated computations. In this section, we present an improved all-pairs Dijkstra’s algorithm for the computation of graph metrics. The new algorithm consists of a forward phase and a backward phase. We construct a queue on the node set \( X \) in advance. For instance, we may set the queue in the natural way, say, \( (x_1, x_2, \cdots, x_n) \). Then, in the forward phase, when we run the single-source Dijkstra’s algorithm for the source node \( \bar{x}_k \), its
destination set is chosen to be the subset $X_k = \{x_k, \ldots, x_n\}$. Then, the shortest path from $x_k$ to $x_{k+j}$ in the subset $X_k$ is called the forward shortest path. In this phase, a path tree is made to indicate the forward shortest paths associated with the queue. Note that, in the forward phase, among all paths from $x_k$ to $x_{k+j}$, only those paths that go through the nodes in the subset $X_k$ are computed. The lengths of the forward shortest paths in the tree can be represented as an upper triangle matrix $F = [f_{ij}]$, where $f_{ij}$ is the forward shortest path length from $x_i$ to $x_j$. In order to find the shortest paths from $x_k$ to all nodes in $X_k$, in the backward phase we need to compute all path lengths of the paths from $x_k$ to the nodes in $X_k$ that go through the nodes in the set $X \setminus X_k$. However, utilizing the forward shortest path tree, the task can be completed in a single step:

$$d_G(x_k, x_{k+j}) = \min_{1 \leq i \leq k} (f_{i,k+j} + f_{i,k}), \quad 1 \leq j \leq n - k.$$ 

Since the new algorithm avoids any repeated computation, it saves about a half of computational time comparing with the regular all-pairs Dijkstra’s algorithm. We present the Matlab code of the new algorithm in the following.

```matlab
function D = ap_dijkstra(W)
% D: graph metric
% W: symmetric weighted matrix
% calculate graph metric
[n,m] = size(W);
if n ~= m
    error('W must be a square matrix');
elseif any(W(:)<0
    error('W must be non-negative');
end
W(1,:)=[]; W(:,1)=[];
% update weighted matrix
W(1,:)=W(:,1); W(:,1)=[];
D=D+D';
end
```

Since the new algorithm avoids any repeated computation, it saves about a half of computational time comparing with the regular all-pairs Dijkstra’s algorithm. We present the Matlab code of the new algorithm in the following.

```matlab
function D = ap_dijkstra(W)
% D: graph metric
% W: symmetric weighted matrix
% calculate graph metric
[n,m] = size(W);
if n ~= m
    error('W must be a square matrix');
elseif any(W(:)<0
    error('W must be non-negative');
end
W(1,:)=[]; W(:,1)=[];
% update weighted matrix
W(1,:)=W(:,1); W(:,1)=[];
D=D+D';
end
```

V. Conclusion

When the graph is sparse, the regular all-pairs Dijkstra’s algorithm run faster than Floyd’s algorithms. When the graph is undirected, we develop an improved all-pairs Dijkstra’s algorithm for the computation of the graph metrics. Taking the advantage of the symmetry of the undirected graphs, the new algorithm avoids repeated computation. The new algorithm consists of two phases. In the forward phase, a forward shortest path tree are built on the graph, while in the backward phase the algorithm utilizes the forward shortest path tree to build a complete shortest path tree so that the graph metric is obtained. The new algorithm saves about a half of the computational time of the regular one.

REFERENCES