Theoretical Computer Science 575 (2015) 10–16

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Variations on the bottleneck paths problem

Tong-Wook Shinn, Tadao Takaoka

Department of Computer Science and Software Engineering, University of Canterbury, Christchurch, New Zealand

ARTICLE INFO

Article history:
Received 24 March 2014
Received in revised form 26 September 2014
Accepted 28 October 2014
Available online 31 October 2014

Keywords:
Algorithm
Graph theory
Bottleneck paths
Shortest paths
APBP
APSP

ABSTRACT

We extend the well known bottleneck paths problem in two directions for directed graphs with unit edge costs and positive real edge capacities. Firstly we narrow the problem domain and compute the bottleneck of the entire network in $O(m \log n)$ time, where m and n are the number of edges and vertices in the graph, respectively. Secondly we enlarge the domain and compute the shortest paths for all possible bottleneck amounts. We call this problem the Shortest Paths for All Flows (SP-AF) problem. We present a combinatorial algorithm to solve the Single Source SP-AF problem in $O(mn)$ worst case time, followed by an algorithm to solve the All Pairs SP-AF problem in $O(\sqrt{t}n^{(\omega+9)/4})$ time, where t is the number of distinct edge capacities and $O(n^{\omega})$ is the time taken to multiply two n-by-n matrices over a ring.

1. Introduction

Let us start by considering a graph where each edge has a capacity. Then the bottleneck of a path is the minimum capacity out of all edge capacities on the path. In other words, the bottleneck is the maximum flow that can be pushed through the path. The Bottleneck Path (BP) from a vertex u to vertex v is the path that gives us the maximum bottleneck value out of all possible paths from u to v. The problem of finding the bottleneck paths from a single source vertex to all other vertices in the graph is known as the Single Source Bottleneck Paths (SSBP) problem, and the problem of finding the bottleneck paths for all possible pairs of vertices is known as the All Pairs Bottleneck Paths (APBP) problem [1–3]. The BP problems have important applications in various fields such as transportation and logistics, computer networking, etc.

We define the bottleneck of the entire graph as the minimum bottleneck from all bottleneck paths for all possible pairs of vertices. We refer to the problem of finding the bottleneck of the entire graph as the Graph Bottleneck (GB) problem. The GB can be considered to be the bottleneck of the entire graph, and highlights the edge that may experience the greatest amount of pressure when the given network is under heavy load. Also, edges with capacities that are smaller than the GB can be considered to be redundant, and solving the GB problem can highlight these edges.

It is straightforward to solve the GB problem by first solving the APBP problem then scanning the results for the minimum bottleneck value. We show that the GB problem can be solved efficiently without having to solve the APBP problem. Our algorithm has the asymptotic worst case time complexity of $O(m \log n)$, where m is the number of edges and n is the number of vertices on the graph.

Let us now consider a graph where each edge has a cost in addition to the capacity. Then there exists a shortest path from vertex u to vertex v that can satisfy a flow of amount up to f. If the flow demand from u to v is less than f,

E-mail address: wookie919@gmail.com (T.-W. Shinn).

http://dx.doi.org/10.1016/j.tcs.2014.10.049
0304-3975/© 2014 Elsevier B.V. All rights reserved.

however, there may exist a shorter path from u to v. This information is useful if we wish to minimize the distance for a given amount of flow. Thus our new problem is to compute the shortest paths for each possible flow value. We call this problem the Shortest Paths for All Flows (SP-AF) problem.

One motivation for solving the SP-AF problem is its application in computer networking. If we model a computer network as a graph such that each router/host is represented as a vertex and each link is represented as an edge, (i, j), then the bandwidth and latency of the link can be considered to be the capacity and the cost of the edge, respectively. If the required data flow from a source host to a destination host can be predetermined, then it is clearly beneficial to find the path with the lowest total latency that has enough bandwidth for the required flow amount, such that the data can be transferred as quickly as possible without causing any network congestion.

As is common in graph paths problems, we divide the SP-AF problem into the Single Source SP-AF (SSSP-AF) problem and the All Pairs SP-AF (APSP-AF) problem. Note that the APSP-AF problem is different from the All Pairs Bottleneck Shortest Paths (APBSP) problem [3], which is to compute the bottlenecks of the shortest paths for all pairs. We present a combinatorial algorithm for solving the SSSP-AF problem in $O(mn)$ time, and we present an algebraic algorithm for solving the APSP-AF problem in $O(\sqrt{t}n^{(\omega+9)/4})$ time, where t is the number of distinct edge capacities and $O(n^{\omega})$ is the time taken to multiply two n-by-n matrices over a ring. The current best algorithm for the matrix multiplication over a ring gives $\omega < 2.373$ [4].

Our algorithms are presented in the order of increasing complexity. Section 3 details the algorithm for solving the GB problem. In Sections 4.1 and 4.2 we present the algorithms for solving the SSSP-AF and APSP-AF problems, respectively.

2. Preliminaries

Let $G = (V, E)$ be a directed graph with unit edge costs and real positive edge capacities where V is the set of vertices (or nodes) and E is the set of edges. Let $n = |V|$ and $m = |E|$. Vertices are given by integers such that $\{1, 2, ..., n\} \in V$. Let $(i, j) \in E$ denote the edge from vertex i to vertex j. Let $c(i, j)$ denote the capacity of the edge (i, j). Let t be the number of distinct edge capacity values. We define the path length as the number of edges on the path and the path distance (or cost) as the sum of all edge costs on the path. For graphs with unit edge costs, the path length and the path distance are equivalent.

Let $R = \{r_{ij}\}$ be the Boolean reachability matrix, where $r_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise. $r_{ii} = 1$ for all $1 \le i \le n$. Let $D = \{d_{ij}\}$ be the distance matrix. Since we are only considering graphs with unit edge costs, $d_{ij} = 1$ if $(i, j) \in E$ and ∞ otherwise. $d_{ii} = 0$ for all $1 \le i \le n$. Let $B = \{b_{ij}\}$ be the bottleneck matrix, where $b_{ij} = c(i, j)$ if $(i, j) \in E$ and 0 otherwise. $b_{ii} = \infty$ for all $1 \le i \le n$.

We now define three types of matrix multiplication, namely the Boolean-product denoted by $\bullet$, the $(\min, +)$-product denoted by $\star$, and the $(\max, \min)$-product denoted by $*$:

$$R \bullet R = \bigvee_{k=1}^{n} \{r_{ik} \wedge r_{kj}\} \qquad D \star D = \min_{k=1}^{n}\{d_{ik} + d_{kj}\} \qquad B * B = \max_{k=1}^{n}\left\{\min\{b_{ik}, b_{kj}\}\right\}$$

We can observe that $r^2_{ij} = 1$ if j is reachable from i via any path of length up to two, d^2_{ij} is the shortest distance possible from i to j via any path of length up to two, and finally b^2_{ij} is the maximum bottleneck possible from i to j via any path of length up to two. Clearly R^{n-1} is the reflexive-transitive closure, D^{n-1} is the solution to the well known All Pairs Shortest Paths (APSP) problem, and B^{n-1} is the solution to the APBP problem.

3. The graph bottleneck problem

Suppose that the APBP problem has already been solved. Then we define the GB, denoted as Θ, as the minimum bottleneck value from all bottleneck paths. Clearly the most straightforward method of computing Θ is to solve the APBP problem then scanning the results in $O(n^2)$ time to find the minimum. The APBP problem can be solved in $O(mn + n^2 \log n)$ time by a simple modification to the well known Dijkstra's algorithm [5]. On dense graphs where $m = O(n^2)$, a sub-cubic algorithm is known for solving the APBP in $O(n^{2.688})$ time bound [1].

Example 1. $\Theta = 9$ for the graph in Fig. 1, which is the capacity of edges $(2, 5)$ and $(7, 8)$.

If we define Θ to be 0 for graphs that are not strongly connected, it is possible to solve the GB problem much faster than the straightforward method described above. We begin by assuming that the edge capacities are integers bounded by c. Let the threshold value h be initialized to $c/2$. Let $G' = (V, E')$ be such that E' only contains edges from E that have capacities greater than or equal to h. We repeatedly halve the possible range $[\alpha, \beta]$ for Θ by adjusting the threshold h through binary search, as shown in Algorithm 1.

Theorem 1. *The GB problem can be solved in $O(m \log n)$ worst case time complexity.*