Optimization of Travelling Cost through a Vertex-Weighted Undirected Graph

Jade Cheng

October 2008

Introduction

The paper is concerned with the management of travelling to a single-source vertex with an optimized cost through a vertex-weighted undirected graph. We provide a comprehensive problem formulation, algorithm design, and complexity analysis for this task. We prove the correctness of applying a Breadth First Search algorithm, and we further optimize the algorithm by identifying and eliminating vertices for which the root selection is inappropriate. After optimization, the average run-time is reduced, but the worst case run-time, however, still exists as $O(V \times E)$.

Problem Formulation

Given: A function $f(T,r) = \sum q(v)\,|p(r,v)|$, where $G = (V,E)$ is a given undirected graph, $v \in V$ is associated with a weight $q(v) \in N$ that represents a frequency of visiting v, $T = (V', E')$ is a sub-graph of G, $V' \subseteq V$, and $E' \subseteq E$, $p(r,v)$ denotes the path from the root r to vertex $v \in V'$ in T, and $|p(r,v)|$ denotes the length of the path $p(r,v)$ from r to v in T.

Sought: A set of edges and vertices (V_{min}, E_{min}) form a tree structure $T_{min} = (V_{min}, E_{min})$ $G(V,E)$, and the tree structure is rooted at the vertex r_{min} such that the function $f(T_{min}, r_{min}) = \sum q(v_{min})|p(r_{min}, v_{min})| \le f(T,r) = \sum q(v)|p(r,v)|$, and all r in V (minimization).

$$\begin{aligned} f\left(T_{min_i}, r_i\right) &= \sum q(v_{min})|p(r_i, v_{min})| \\ &= q(v_1)|p(r_i,v_1)| + q(v_2)|p(r_i,v_2)| + \cdots + q(v_{V-1})|p(r_i,v_{V-1})| \end{aligned} \tag{1}$$

In order to consider the overall optimal solution of equation (1), it would be nice if we could directly use the optimal solutions for all individual components, $q(v_1)|p(r_i,v_1)|$, $q(v_2)|p(r_i,v_2)|$, …, and $q(v_{V-1})|p(r_i,v_{V-1})|$. This is the problem known as the single-source shortest-path tree problem. There are known algorithms such as BFS to solve it. Proving two assumptions suffices for this purpose of adapting these known algorithms.

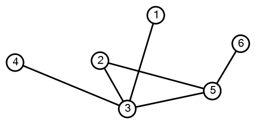
1. The individual minimal solutions for all components for equation (1) occur at the same time under the same conditions. When equation (1) is optimized, all of its components are optimized at the same time as well. No vertex needs to sacrifice its own shortest path for the overall optimization.
2. If the first assumption is proved, the output graph T_{min_i} is still a connected graph and forms a spanning tree structure.

After we proved the correctness of applying the BFS algorithm, we also optimized the algorithm by identifying and eliminating the vertices for which the root selection is inappropriate. This optimization is demonstrated in the pseudo-code on lines 7 to 16. We modified the tree traversal part of our algorithm by keeping track of the hop distances of from each vertex to a certain root. This part is demonstrate on lines 17 to 47 and in particular line 32.

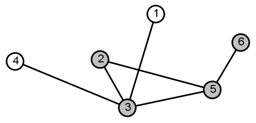
Algorithm Design and Optimization

```
JADE-BFS-ALGORITHM(Matrix M)
1     int maxWeight ← 0;
2     int solutionSum ← 0;
3     Matrix input ← the input adjacency M;
4     Matrix solution ← populate with 0;
5     VertexNode root ← null;
6     Queue candidateSet ← Ø
7     for every v ∈ V {
8         if v.getWeight > maxWeight {
9             maxWeight ← v.getWeight
10        }
11    }
12    for every v ∈ V {
13        if v.getDegree ≠ 1  or  v.getWeight = mWeight {
14            candidateSet.push(v);
15         }
16    }
17    while candiateSet ≠ Ø {
18        Matrix localSolution ← populate with 0;
19        int localSum ← 0;
20        Queue temp ← Ø;
21        VertexNode vroot ← candidateSet.pop;
22        temp.push(vroot);
23        vroot.setHops(0);
24        for every v ∈ V and v ≠ vroot {
25            v.setHops(∞)
26        }
27        while temp ≠ Ø {
28            VertexNode u ← temp.pop;
29            for every v ∈ V and v ≠ vroot {
30                if input.adjacent(v, u) {
31                    if v.getHops = ∞ {
32                        v.setHops(1 + u.getHops);
33                        localSolution.mark(v, u);
34                        temp.push(v);
35                    }
36                }
37            }
38        }
39        for every v ∈ V {
40            localSum ← localSum + v.getWeight × v.getHops;
41        }
42       if localSum < solutionSum {
43            solutionSum ← localSum
44            solution ← localSolution
45            root ← vroot
46        }
47    }
```

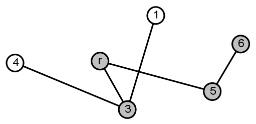
Simplified Example



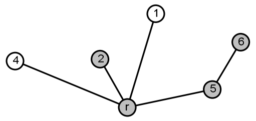
Input Graph



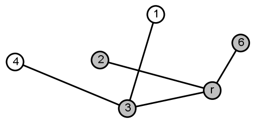
Candidates for Root Vertex



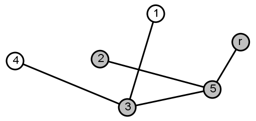
Solution for Graphs Rooted at 2
$\sum q(v)|p(r,v)| = 30$



Solution for Graphs Rooted at 3
$\sum q(v)|p(r,v)| = 24$



Solution for Graphs Rooted at 5
$\sum q(v)|p(r,v)| = 21$



Solution for Graphs Rooted at 6
$\sum q(v)|p(r,v)| = 32$

Algorithm Complexity Analysis

Time Analysis: $O(V) + O(V) + O\big(V \times (V + E)\big) = O(V \times E)$

Space Analysis: $V \times \frac{V}{2} + V \times 3 + V \times \frac{V}{2} + E = V^2 + 3V + E$