All Pairs Maximum Path Algorithms for an Average Path Value

Gordon Beavers, University of Arkansas
Wing-Ning Li,University of Arkansas

Abstract

Let $G(V, E)$ be a weighted connected simple graph. Given a pair of vertices $u, v \in V$, let $Max(u, v)$ denote the maximum *average path value* over all simple paths from u to v. For a given simple path from u to v the *average path value*, $apv_P(u, v) = min(P)/length(P)$ where $min(P)$ is the weight of the minimum weight edge in the path P and $length(P)$ is the number of edges in P. This notion of *average path value* has been used in the analysis of social networks. Algorithms are presented for the calculation of *average path value*.

Key Words: Combinatorial optimization, optimum path weight, average path weight, graph algorithms.

1 Introduction

Let $G(V, E)$ be a weighted connected simple graph. Given a pair of vertices $u, v \in V$, let $Max(u, v)$ denote the maximum *average path value* over all simple paths from u to v. For a given simple path P from u to v the *average path value*, $apv_P(u, v) = min(P)/length(P)$ where $min(P)$ is the weight of the minimum weight edge in the path P and $length(P)$ is the number of edges in P. This notion of *average path value* has been used in the analysis of social networks [1].

Algorithms are provided for undirected graphs. In most cases it is obvious how the algorithm may be modified for directed simple graphs. Single source maximum average path algorithms are provided which provide the method for all pairs maximum average path algorithms. For dense graphs a matrix multiplication method can be used for all pairs maximum paths. The size of the set of edge weights relative to the size of the set of vertices is a parameter that can affect the run time of the algorithms. Edge weights are restricted to non-negative real numbers.

The principle of optimality is exemplified by the shortest paths problem in that subpaths of shortests paths are themselves shortest paths. When the principle of optimality holds more efficient greedy or dynamic programming algorithms can be used. The optimality principle fails to hold for this notion of average as the following example demonstrates. Nevertheless we have been able to provide polynomial time algorithms for this type of average.

Optimality fails because there are two criteria for determining the optimal path: a smaller number of edges and a larger least weight edge. Either of these criteria can dominate the other. We present two examples (Figure

1) in which the optimal path does not use the optimal subpath. In the first example the optimal path is the one with the smaller minimum, but a smaller number of edges, that is, the optimal path from a to f is $adef$ while the optimal path from a to e is $abce$. In the second example the optimal subpath has more edges but a larger minimum, that is, the optimal path from a to c utilizes the subpath containing 12 edges with a minimum weight of those edges being 23, while the optimal path from a to b utilizes the path containing 5 edges with a minimum weight of those edges being 10.

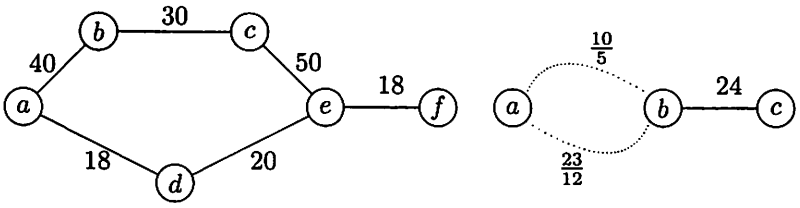

Figure 1: Examples where optimal path do not use optimal subpaths.

The structure of the paper is as follows. In section 2 a version of breadth first search, developed specifically for this problem, is presented that considers the weight of an edge as a criterion for inclusion in the breadth first tree. The new version is called WEIGHTED -BFS. In section 3 a special case of the problem, when the edge weights are equal, is considered for which a more efficient algorithm is given. Section 4 considers algorithms for simple connected graphs. The results are summarized in the conclusion.

2 Breadth First Search

We provide a version of breadth first search that determines both the number of edges in a shortest path from a source vertex s to another vertex u, $d(u)$, and the weight of the minimum weight edge in the BFS path. The procedure has an added parameter w_0. Only edges of weight at least as large as w_0 are considered. The run time is $O(|V| + |E|)$.

WEIGHTED-BFS(G, s, w_0) returns shortest paths (least number of edges) from s to all other vertices such that all edges in the paths have weight at least as large as w_0. WEIGHTED-BFS$(G, s, 0)$ is equivalent to standard breadth first search if all edge weights in the graph are non-negative.

WEIGHTED-BFS(G, s, w_0)
1. for each u
2. $\quad d(u) = \infty$
3. $\quad \pi(u) = NIL$
4. $\quad color(u) = white$
5. $\quad min(u) = 0$
6. $d(s) = 0$
7. $min(s) = 0$
8. $color(s) = gray$
9. ENQUEUE(Q, s)
10. while $Q \neq \emptyset$
11. $\quad u =$DEQUEUE(Q)
12. $\quad$ for each $v \in adj(u)$
13. $\quad\quad$ if $color(v) == white$ and $w(u, v) \geq w_0$
14. $\quad\quad\quad color(v) = gray$
15. $\quad\quad\quad d(v) = d(u) + 1$
16. $\quad\quad\quad \pi(v) = u$
17. $\quad\quad\quad min(v) = \min(min(u), w(u, v))$
18. $\quad\quad\quad$ ENQUEUE(Q, v)

WEIGHTED-BFS(G, s, w_0) is basically the breadth first search algorithm found in **Introduction to Algorithms** [2] by Cormen et. al. modified to consider only edges having a weight of at least w_0 where $w_0 \geq 0$, and with an added variable $min(u)$ which is the value of the minimum weight edge in the shortest path. The modification essentially causes the BFS algorithm to run on a subgraph of G containing only the edges of G with weight at least w_0. See [2] for proof of correctness and justification of the run time.

3 Equal Edge Weights Case

In this special case, all the edges of the graph have the same weight. Let $d(u)$ denote the length (number of edges) of the shortest path (least number of edges) from the source vertex s to the vertex u. Let w be the weight of each edge. Let $apv(u)$ denote the average path value for the BFS path from the source s to vertex u. If $w \geq 0$ the following algorithm solves the special case.

EQUALWEIGHTAPV(G, s)
1. WEIGHTED-BFS$(G, s, 0)$
2. for each $v \in V$
3. $\quad apv(v) = \frac{w}{d(v)}$

The correctness of the algorithm follows from the fact that if edge weights are non-negative and equal then the maximum average path is the shortest path in terms of the number of edges. WEIGHTED-BFS can be used to solve the single source all destinations shortest path problem in time $O(|V| + |E|)$. WEIGHTED-BFS is called $|V|$ times to get all pairs shortest path lengths with a run time of $\Theta(|V|(|V| + |E|))$. Therefore, for sparse graphs, that is for graphs with $|E| = O(|V|)$, the overall time complexity of the algorithm is $\Theta(|V|)$ or $\Theta(|V|^2)$ for all pairs. For dense graphs, using a matrix multiplication algorithm [3] to determine shortest paths (least number of edges) between pairs of vertices, results in a run time of $O(|V|^{2.376} \lg |V|)$, which becomes the overall time complexity for dense graphs. For the general case, we will not likely do better in time complexity than the special case.

4 Algorithms for Simple Graphs

As the following algorithm progresses, a path $s \rightsquigarrow u$ is chosen just in case every shorter path from u to v contains an edge of sufficiently lesser weight to cause the value of the function $apv(u)$ to decrease. Let k denote the number of distinct edge weights. The uppper bound for k is $|E|$ and is achieved when edge weights are distinct. The runtime of the algorithm is $O(k(|V| + |E|))$ for a single source and thus $O(k|V|(|V| + |E|))$ for all pairs. This algorithm checks each edge for being the minimum weight edge in a path yielding $Max(s, u)$. When the algorithm terminates $apv(u) = Max(s, u)$. The algorithms works by progressively removing edges of lesser weight from consideration in BFS paths.

```
SHRINKINGMETHOD(G, s)
1. sort edge weights in increasing order (no duplicates)
2. for each vertex u ∈ V
3.         apv(u) = 0
4. for each weight w in sorted order
5.                 WEIGHTED-BFS(G, s, w)
6.                 for each vertex u
8.                         temp = min(u)/d(u)
9.                         if temp > apv(u)
10.                                apv(u) = temp
```

Consider an optimal path P from u to v. Let w' be the weight of the least weight edge in P. Then P is a path with the least numbers of edges from u to v containing only edges of weight w' or greater. The correctness of the algorithm follows from the observation that the algorithm determines

shortest paths (in terms of the number of edges) containing only edges of weight at least w for every edge weight w. WEIGHTED-BFS(G, s, w') may determine a path different from P, but that path will have the same length and same average path value as P. The following theorem formalizes this reasoning.

Theorem When SHRINKINGMETHOD(G, s) terminates $MAx(s, u) = apv(u)$.
Proof We show that if $Max(s, u) = w'/k$ and there is a path of length k with minimum weight edge w', then at the stage of the algorithm when w is set to w' $apv(u) = w'/k$ and the value of $apv(u)$ remains unchanged for the remainder of the algorithm.

Suppose that $Max(s, u) = w'/k$, there is a path of length k with minimum weight edge w', and that when w is set to w' $apv(u) \neq w'/k$. Since there is a path of length k with a minimum weight edge of weight w' that will be discovered at this stage (if it has not been discovered before) this can only occur if $apv(u) > w'/k$ which means that $apv(u)$ has been set to the larger value at an earlier stage of the algorithm and thus $Max(s, u) \neq w'/k$ contrary to the assumption.

Once $apv(u)$ has achieved the maximum value, it is not altered. *QED*

The following matrix multiplication method runs faster on dense graphs with a runtime of $O(kn^{2.376} \lg n)$ [3], where k is the number of distinct edge weights and n is the number of vertices. In the following code $n = |V|$, Adj is the matrix of edge weights, APV is the current best average path value, and D is the matrix of the number of edges in a shortest path between pairs of vertices.

```
ALL PAIRS MAX PATHS(W)
1. sort edge weights in increasing order
2. initialize matrices Adj and APV
3. for each edge weight w in sorted order
3.       update matrix Adj (remove edges of weight less than w)
4.       D =MATRIXSHORTESTPATHS(Adj)
5.       for i = 1 to n
6.             for j = 1 to n
7.                   if w/d(i,j) > apv(i,j)
8.                         apv(i,j) = w/d(i,j)
9. return APV
```

The correctness of the algorithm can be argued as for the previous algorithm in that the only difference is that a matrix method is being used to find the shortest path (least number of edges) between vertices, rather than BFS.

5 Conclusion

This paper has provided algorithms to find the maximum average path values between vertices of a weighted simple connected graph $G(V, E)$, where given a pair of vertices $u, v \in V$, $Max(u, v)$ denotes the maximum *average path value* over all simple paths from u to v. For a given simple path P from u to v the *average path value*, $apv_P(u, v) = min(P)/length(P)$ where $min(P)$ is the weight of the minimum weight edge in the path P and $length(P)$ is the number of edges in P.

6 References

[1] S. Yang and D. Knoke, *Optimal connections: strength and distance in valued graphs*, **Social Networks23** (2001) 285-295.

[2] T. Cormen, C. Leiserson, R. Rivest and C. Stein, **Introduction to Algorithms**, Second Edition, McGraw-Hill (2001).

[3] R. Seidel, *On the all-pairs-shortest-path problem in unweighted undirected graphs*, **J. Comput. Sys. Sci. 51** (1995) 400-403.