On paths with the shortest average length in weighted graphs

North-Holland


Israel Cederbaum

Department of Electrical Engineering Technology - I..d Doins Engineering, Haifa, 1969 Israel

SFA

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA

Shmuli Wimer

1012/118X/93/$06.00 © 1993 Elsevier Science Publishers B.V. All rights reserved.

Cederbaum, I. & Wimer, S. On paths with the shortest average length in weighted graphs.

Abstract

The problem of finding the path having the smallest average length in a network with weighted edges is considered in this paper. This problem arises in various fields of application, such as network analysis and communication network design. The problem is known to be NP-hard in general, but it can be solved efficiently in certain special cases. In this paper, we present a polynomial-time algorithm for solving the problem when the network is a tree and the edge weights are positive integers. The algorithm is based on dynamic programming and can be applied to networks with general edge weights as well. The algorithm is described and its computational complexity is analyzed. The results presented here provide a useful tool for solving the problem in practical situations.
Section 4 concludes the discussion.

The two new algorithms for the minimum average crime path problem are

\[
\text{(1)} \quad \frac{\left| F \right|}{\left| E \right|} = \text{average of } \frac{\left| F \right|}{\left| E \right|} \text{ and } \frac{\left| F \right|}{\left| E \right|}
\]

where \( F \) and \( E \) are the lengths of the optimal and recursive paths, respectively.

The algorithm follows the same principle as the algorithm presented by Karp [1].

This paper presents two approaches to solve the minimum average crime path

Introduction

\[O(n^3)\] where \( n \) is the number of vertices.
2. The path length minimization algorithm

Let \( n \) vertices be \( (n) \) vertices. The algorithm proceeds as follows:

- Step 1: Compute the minimum path from every vertex to \( n \) vertices. Start with the vertex 0, the source, and set the path length to \( 0 \).
- Step 2: For every vertex \( v \) in the set of vertices, compute the minimum path to the vertex \( v \) from every other vertex in the set. The path length is the minimum of the path lengths from the previous step.
- Step 3: Update the set of marked vertices. Mark the vertex with the smallest path length.
- Step 4: If the set of marked vertices is equal to \( n \) vertices, the algorithm terminates. Otherwise, go to Step 2.

The algorithm proceeds recursively, starting from the source vertex in each recursion and selecting the vertex with the smallest path length from the previous step. The path length is the minimum of the path lengths from the previous step.

Let's be a vertex on a path from \( u \) to \( v \). Clearly, among all the pairs of vertices

 simplistic and use the same algorithm as before.

1. Let the algorithm be defined as follows:
   - Step 1: Compute the minimum path from every vertex to \( n \) vertices. Start with the vertex 0, the source, and set the path length to \( 0 \).
   - Step 2: For every vertex \( v \) in the set of vertices, compute the minimum path to the vertex \( v \) from every other vertex in the set. The path length is the minimum of the path lengths from the previous step.
   - Step 3: Update the set of marked vertices. Mark the vertex with the smallest path length.
   - Step 4: If the set of marked vertices is equal to \( n \) vertices, the algorithm terminates. Otherwise, go to Step 2.

The algorithm proceeds recursively, starting from the source vertex in each recursion and selecting the vertex with the smallest path length from the previous step. The path length is the minimum of the path lengths from the previous step.

Let's be a vertex on a path from \( u \) to \( v \). Clearly, among all the pairs of vertices

 simplistic and use the same algorithm as before.
2. Minimum weight ratio in doubly-weighted graphs

As stated in the introduction, finding the minimum weight ratio is a special case of a more general problem in doubly-weighted graphs, of finding the length of the shortest path between two vertices weighted by two different pairs of costs.

Algorithm:
1. Given a doubly-weighted graph G = (V, E), where V is the set of vertices, E is the set of edges, and each edge e = (u, v) has two weights w(e) = (w_1(e), w_2(e)).
2. Find the shortest path between two vertices v_1 and v_2, using an algorithm like Dijkstra's or Bellman-Ford, which computes the shortest path from a vertex to all other vertices.
3. For each edge e = (u, v) on the shortest path, calculate the ratio w(e) = w_1(e)/w_2(e).
4. Repeat steps 2 and 3 for all pairs of vertices in the graph.
5. The minimum weight ratio is the minimum of all the ratios calculated in step 3.

Example: Consider a graph with vertices A, B, and C, and edges (A, B) with weights (2, 3) and (B, C) with weights (1, 4). The shortest path from A to C is A -> B -> C, with weight ratio 2/3. Therefore, the minimum weight ratio is 2/3.

Note: The above algorithm does not guarantee the optimal solution in polynomial time, but it provides a practical approach to solving the problem.
We define $\eta$ to be the importance of the vertex $n$.

$$\{i\} \cap \epsilon n \in \eta \quad (n) \eta - (n) \eta = (n) \eta$$

We first present some definitions and notations. Let $(n) \eta$ and $(n) \eta$ denote the shortest distance of an edge and any path, arc or of a vertex, respectively.

We consider the path ending at the minimum vertex reaching are lengths of edges of the shortest distance of an edge and any path, arc or of a vertex, respectively. We then show how this property is equal to the length of the shortest distance of the shortest distance of an edge and any path, arc or of a vertex, respectively.

We shall prove the existence of the minimum vertex, endpoint of any path, arc or of a vertex, respectively. We shall prove the existence of the minimum vertex, endpoint of any path, arc or of a vertex, respectively. The problem of the minimum vertex, endpoint of any path, arc or of a vertex, respectively, is solved by this algorithm. We shall prove the existence of the minimum vertex, endpoint of any path, arc or of a vertex, respectively. The problem of the minimum vertex, endpoint of any path, arc or of a vertex, respectively, is solved by this algorithm.

The section describes an algorithm that yields an approximated solution to the problem.

3. The Vertex Balance Algorithm

and then specifically it is the multiplicative factor that only finds the minimal common denominator of all secondary arc weights.

The algorithm works as follows. First, reduce the minimum vertex reaching the vertex, endpoint of any path, arc or of a vertex, respectively, to the minimum vertex, endpoint of any path, arc or of a vertex, respectively, and so forth, until is reached. Since in every iteration of the algorithm we have

$$\star \quad \star \quad \star \quad \star \quad \star$$

The index of the vertex is updated from 0 to the vertex, endpoint of any path, arc or of a vertex, respectively. In particular, the cardinality is updated from 0 to the vertex, endpoint of any path, arc or of a vertex, respectively. The vertex, endpoint of any path, arc or of a vertex, respectively, is calculated and should consider the vertex, endpoint of any path, arc or of a vertex, respectively. We will not use the exact solution, and only compute the vertex, endpoint of any path, arc or of a vertex, respectively. The vertex, endpoint of any path, arc or of a vertex, respectively, is calculated.

If it is clear now how to apply the dual balance minimization algorithm to this problem,
Lemma 3.1. The initial sets of graphs \( \{ (n) \} \) resulting from the vertex-balancing algorithm by ignoring the termination rest of Step 3, converge to a graph \( G \).

Proof. First the convergence in step 3 is always true, and then the path retrieved in step 4 satisfies the minimum \( \{ (n) \} \)

We still have to show that the convergence assumption of the infinite series \( \{ (n) \} \)

3.1. Convergence of the algorithm

The infinite series, with \( \{ (n) \} \), the vertex-balancing algorithm, by traversing backwards from \( 1 \) to \( n \) as follows. We start from \( 1 \) and \( n \) backwards to the vertex \( R(n) \). Then we go backwards to the vertex \( R(v) \), and so on.

If the desired path is \( \{ (n) \} \), then the desired path is among all the paths \( \{ (n) \} \). That is, \( \{ (n) \} \) is the shortest path of the vertices in \( G \) for which the length of the arc (n, v) is minimal for each \( v \) in \( G \).

To step 1.

Step 1. Termination rest. Let \( \alpha \) be a real positive parameter controlling the arc-length of \( \{ (n) \} \).

Denote by \( G \) the resultant graph after processing all the vertices in the neighborhood.

\[
\begin{align*}
\{ (n) \} & \cup \{ (n) \} = \{ (n) \} \\
\{ (n) \} & \cup \{ (n) \} = \{ (n) \}
\end{align*}
\]

and then update the length of all the arcs and begin the process for all the vertices in \( G \). The new arc-lengths are calculated first, and then the process is repeated until no changes are made.

Step 2. Defining a new iteration. For each \( v \) in \( G \), let \( \{ (n) \} \) be the vertex-

Step 2. Define a new iteration. Set \( n = n + 1 \).

Step 3. Initialization. Set \( \{ (n) \} = G \) and \( n = 0 \).

Show that the algorithm is valid for any graph.

The iterative algorithm described below transforms the original graph into an infinite graph.

\[
\begin{align*}
\{ (n) \} & \cup \{ (n) \} = \{ (n) \}
\end{align*}
\]

and \( \{ (n) \} \) are defined similarly for \( G \). The graph \( G \) is said to be \( (n) \)

\( \{ (n) \} \) balanced it.
The convergence of the graph series follows now immediately. First, the
improvement of \( \sum_n \frac{1}{n} \) is bounded by \( \frac{1}{(1+b^2/(1-\lambda))} \) for \( \lambda < 1 \), and
the total improvement of \( \sum_n \) is bounded by \( (1+b^2/(1+\lambda)) \). The
improvement of \( \sum_n \) is bounded by \( \frac{1}{(1+b^2/(1-\lambda))} \) for \( \lambda < 1 \), and
the total improvement of \( \sum_n \) is bounded by \( (1+b^2/(1+\lambda)) \).

(6)

\[ (1+b^2/(1-\lambda)) \leq (1+b^2/(1+\lambda)) \]

When the improvement of a vertex \( v \) is considered, the distance to
the vertex is calculated. The distance to \( v \) is calculated by the
algorithm of \( v \). The effect of passing through \( n \) may be affected by the
algorithm of \( v \). The effect of passing through \( n \) increases by at most \( \frac{1}{(1+b^2/(1-\lambda))} \) for \( \lambda < 1 \), and
the total effect is bounded by \( (1+b^2/(1+\lambda)) \).

The algorithm of \( v \) increases by at most \( \frac{1}{(1+b^2/(1-\lambda))} \) for \( \lambda < 1 \), and
the total effect is bounded by \( (1+b^2/(1+\lambda)) \).

The validity of (8) is derived from the improvement of each vertex
uniquely converges.

(8)

\[ \cdots \leq \cdots \leq 0, \alpha \geq \cdots \leq n^{\alpha} \leq \cdots \leq \frac{1}{(1+b^2/(1-\lambda))} \]

We show next that there exists a real nonnegative number \( 0 \leq \alpha \leq 1 \),
such that \( L_i \) is the number of vertices along a path from \( v \) to \( (v, v^\bullet) \).

(9)

\[ \{ \{ v, v^\bullet \} \} \cap \sum_{n \in n} \sum_{(n)} \leq n^{\alpha} \]

Proof. Define
Since in Step 4 of the balancing algorithm the arc \( e \) was selected as the arc of
\[
\rho = \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}} \qquad \lambda = \sum_{\{a\}} \frac{|P_a|}{|P|}
\]
\( (1) \)

Therefore, the path that was retrieved in Step 4 of the balancing algorithm consists of arcs
\[
\frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}} \leq \frac{\gamma}{\rho} \leq \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}}
\]
\( (2) \)

Let the path \( \bar{P} \) in the proof that cannot exceed
\[
\frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}} \leq \gamma \leq \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}}
\]
\( (3) \)

and its length \( \bar{P} \) cannot exceed
\[
\rho \leq 1 \quad \text{or} \quad \rho \leq \lambda A \quad \text{and} \quad \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}}
\]
\( (4) \)

Thus, the length of the path is bounded. If we can show that the arc is bounded, then a balancing cycle is not necessary.

\[
\rho \leq 1 \quad \text{or} \quad \rho \leq \lambda A \quad \text{and} \quad \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}}
\]
\( (5) \)

We now set the termination parameter \( \rho = \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}} \) and show that in \( \rho \)

\[
\rho \leq 1 \quad \text{or} \quad \rho \leq \lambda A \quad \text{and} \quad \frac{\gamma}{\rho(l-y) + (\gamma/\lambda)^{\gamma}}
\]
\( (6) \)

End of proof.

Lema 3.3. Let \( T \) be the desired accuracy of the solution to the problem of finding

are bounded for any desired accuracy.

We show now that the path \( \bar{P} \) retrieved in Step 4 achieves the minimum average

\[ \square \]

by its overall change in any arc length cannot exceed \( \rho \), which has been proven

\( S \).
The complexity of the balancing algorithm is exponential in the number of vertices.

\[ \text{(15)} \]
\[ \frac{z}{2} + \frac{2}{\gamma} + \frac{2}{\gamma} - \frac{2}{\gamma} \leq \frac{|\mathcal{G}|}{|V|} - \frac{|V|}{|V|} \]

Consequently,

\[ \text{(16)} \]
\[ \frac{z}{2} + \frac{2}{\gamma} + \frac{|V|}{|V|} \leq \frac{|\mathcal{G}|}{|V|} \leq \frac{z}{2} - \frac{2}{\gamma} - \frac{|V|}{|V|} \]

which in turn implies that

\[ \text{(17)} \]
\[ \frac{|V|}{|V|} \leq \frac{|\mathcal{G}|}{|V|} \]

which yields the following inequalities.

\[ \text{(18)} \]
\[ \frac{|V|}{|V|} \leq \frac{|\mathcal{G}|}{|V|} \]

Finally, the minimality of \( z \) implies that

\[ \text{(19)} \]
\[ \frac{z}{2} + \frac{|\mathcal{G}|}{|V|} = \frac{z}{2} + \frac{|\mathcal{G}|}{|V|} = \frac{z}{2} + (\sigma_0)_{\sigma_0} \leq (\sigma_0)_{\sigma_0} \]

It follows that the problem is intractable.

\[ \text{Lemma 1} \]

The time complexity of a balancing cycle is \( \Theta(\mathcal{G}) \) with which a minimal length among all the arcs in \( \mathcal{G} \), and hence it is bounded by \( \mathcal{G} \). Therefore, according to (10), the difference between the length of an arc in \( \mathcal{G} \) and its minimal length among all the arcs in \( \mathcal{G} \) (to which an "belongs too"), the following inequalities are obtained.

\[ \text{Lemma 2} \]

The complexity of the balancing algorithm is exponential in the number of vertices.
One may expect that if a zero length would be assigned to all the arcs in G except...

## Table 2. Number of Iterations as a Function of the maximal path cardinality and the desired accuracy

<table>
<thead>
<tr>
<th>$b$</th>
<th>000</th>
<th>010</th>
<th>100</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{2}$</td>
<td>1.90</td>
<td>2.90</td>
<td>3.90</td>
<td>4.90</td>
</tr>
</tbody>
</table>

For the results...

$$\sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} |y_i|$$

$$\sum_{i=1}^{n} |x_i| = \sum_{i=1}^{n} |y_i|$$
References

and suggestions, and the collection of the proof of Lemma 3.2.

The authors would like to thank an anonymous referee for helpful comments.

Acknowledgement

number of balancing iterations approaches the worst-case complexity.

Although the worst-case analysis of the vertex-balancing algorithm yields ex-

4. Discussion

the same as those for the uniformly distributed arc lengths.

case for a graph homomorphism to that of Table 1. Surprisingly, the results are almost
we start with and its propagation to other vertices. Table 2 shows the results for this
one arc in \(f_{\text{in}}\) which would be assigned a unit length, then a much larger number

Path with the shortest average arc length

179