mathematics

Article

A Bellman–Ford Algorithm for the Path-Length-Weighted Distance in Graphs

Roger Arnau, José M. Calabuig, Luis M. García-Raffi, Enrique A. Sánchez Pérez * and Sergi Sanjuan

Instituto Universitario de Matemática Pura y Aplicada, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain; ararnnot@upvnet.upv.es (R.A.); jmcalabu@mat.upv.es (J.M.C.); lmgarcia@mat.upv.es (L.M.G.-R.); ssansil@upvnet.upv.es (S.S.)
* Correspondence: easancpe@mat.upv.es; Tel.: +34-963866000

Abstract: Consider a finite directed graph without cycles in which the arrows are weighted by positive weights. We present an algorithm for the computation of a new distance, called path-length-weighted distance, which has proven useful for graph analysis in the context of fraud detection. The idea is that the new distance explicitly takes into account the size of the paths in the calculations. It has the distinct characteristic that, when calculated along the same path, it may result in a shorter distance between far-apart vertices than between adjacent ones. This property can be particularly useful for modeling scenarios where the connections between vertices are obscured by numerous intermediate vertices, such as in cases of financial fraud. For example, to hide dirty money from financial authorities, fraudsters often use multiple institutions, banks, and intermediaries between the source of the money and its final recipient. Our distance would serve to make such situations explicit. Thus, although our algorithm is based on arguments similar to those at work for the Bellman–Ford and Dijkstra methods, it is in fact essentially different, since the calculation formula contains a weight that explicitly depends on the number of intermediate vertices. This fact totally conditions the algorithm, because longer paths could provide shorter distances—contrary to the classical algorithms mentioned above. We lay out the appropriate framework for its computation, showing the constraints and requirements for its use, along with some illustrative examples.

Keywords: graph; distance; Bellman–Ford; algorithm; path-length-weighted

MSC: 05C38; 90C35

check for updates

Citation: Arnau, R.; Calabuig, J.M.; García-Raffi, L.M.; Sánchez Pérez, E.A.; Sanjuan, S. A Bellman–Ford Algorithm for the Path-Length-Weighted Distance in Graphs. *Mathematics* **2024**, *12*, 2590. https://doi.org/10.3390/math12162590

Academic Editor: Andrea Scozzari

Received: 15 July 2024
Revised: 16 August 2024
Accepted: 20 August 2024
Published: 22 August 2024

1. Introduction

Algorithms for calculating the (weighted) path distance between vertices in a graph appeared in the middle of the 20th century, which were motivated by the growing interest at the time in the applications of the mathematical analysis of graphs. The Bellman–Ford algorithm is the main reference of these early studies [1]. Dijkstra's algorithm for solving the same problem appeared at about the same time [2], and it differs from the other, being more efficient depending on the particular problem. More specifically, Dijkstra's algorithm is best used when you have non-negative weights and need an efficient solution for static scenarios. Bellman–Ford, on the other hand, is more versatile for situations where edge weights can be negative and where it is important to detect negative cycles, making it suitable for more complex or dynamic environments (see [3], p. 604). Thus, Dijkstra's algorithm can be used for finding the shortest driving route between two cities or for finding the shortest path for data packets in a network. However the Bellman–Ford algorithm can be used for analyzing currency exchange rates to detect arbitrage opportunities or for analyzing supply chains with possible gains and losses during transport.

After these original works, the growing interest in the subject (due to the numerous applications that graph theory has found in many fields) has given rise to a great deal of research on graph analysis, which often includes the study of these structures when considered as metric spaces. The idea of also considering a graph as a metric space goes back to the beginnings of the theory of graphs. Metric notions began to appear explicitly in mathematical works in the second half of the last century. The main metric that was considered (and, in a sense, the only one until the latter part of the century) was the so-called path distance [4–6]: for undirected and connected graphs, this metric evaluated between two vertices (nodes) is defined as the length of the shortest path between them (see, for example, [7] §.2.2.2). One of the first advances in the metric analysis of graphs was the introduction of weights in the definition of the path distance, assigning weights to the individual paths connecting two consecutive nodes and calculating the infimum of the sum of these weights. Some recent papers on the subject using weighted distances that have inspired this paper are [8,9].

As in the case of other notions of fundamental graph theory, the relevant theoretical ideas appeared together with research topics from other scientific fields, such as sociology [10–12]. The definition of different metrics and algorithms to compute them increased greatly in the last decade of the last century, often proposed by problems from other disciplines such as chemistry or crystallography (see [6,13] and the references therein). In this sense, there is a particular case that deserves attention, that is, the resistance distance [14,15]. It is a concept derived from electrical circuit theory and applied to graph theory. It measures the "effective resistance" between two nodes (vertices) in a graph, treating the graph as an electrical network where each edge is a resistor. The resistance distance between two nodes i and j in a graph can be calculated using the Laplacian matrix L of the graph. Specifically, if L^+ denotes the Moore–Penrose pseudoinverse of L, the resistance distance $dr(i,j)$ is given by

$$dr(i,j) = L^+_{ii} + L^+_{jj} - L^+_{ij},$$

where L^+_{ij} is the i,j element of L^+. This metric is a measure of the ease with which electric current flows between two points, with a higher resistance distance indicating a more "difficult" or "costly" connection in the graph's structure. This concept has applications in various fields such as data analysis, network reliability, and clustering tasks. Related also to some ideas in theoretical chemistry [13] and social network analysis [12], this definition turned out to be a useful tool in the study of molecular configurations in chemistry, although it has also been used in network analysis and other fields [16–18]. In general, metrics could play a relevant role when studying properties such as robustness (see, for example, the survey [19]; see also [20]). The interested reader can find more information on the applications of metric graphs in the books [7,21,22].

In the same direction, in this paper, we provide an algorithm to compute a new distance that also appeared in an applied context in connection with the automatic analysis of fraud in economic networks (see [23]). In this paper, we show that this metric allows one to consider vertices that are far away when the distance is measured using the path distance, becoming close with respect to this metric. It is especially useful in the economic analysis of fraud in business networks, as often the strategy used to hide such fraud is to consider that the analysis tools to be used against them are based on the path distance: closer vertices in terms of fewer intermediate nodes means that they are more related vertices. However, financial fraudsters always try to introduce many intermediate companies (intermediate nodes) to avoid detection.

Technically, our metric is defined as a weighted metric, but this is achieved by dividing the sums of weights appearing in the infimum that provides the value of the metric using a new term that depends on the number of steps involved in the summation. We will show that this change in the weighting process forces us to radically change the algorithm to calculate it, since in this new case, longer paths could give shorter distances. However, in the present work, we consider the case of acyclic directed graphs in order to avoid restrictions that make it impossible to define a weighted path metric, since continuous passage through a cycle could always give a null value to the metric (which would mean that it is not a metric). There are other ways to avoid this (see, for example, Proposition 4.1 in [23]), but in our case, we decided to compute the distance between vertices by restricting the set of possible paths in the infimum that gives it. The way to do this is to avoid cycles and to consider directed graphs. As a result, what we compute is not a metric on the whole graph but only a distance between two vertices chosen in it.

Let us give now some basic definitions about graphs and metric spaces. Let us start by explaining some concepts related to the general definition of what a metric is, which will be adapted later according to the graph theoretic framework. Let $\mathbb{R}^+$ be the non-negative real numbers. An (extended) quasi-metric on a set Ω is a function $d : \Omega \times \Omega \to \mathbb{R}^+ \cup \{\infty\}$ such that for all $a, b, c \in \Omega$, the axioms

1. $d(a,b) = 0 = d(b,a)$ if and only if $a = b$, and
2. $d(a,b) \leq d(a,c) + d(c,b)$

hold. The resulting quasi-metric structure (Ω, d) is called a quasi-metric space. For the specific framework of this paper, a useful summary of the notions of distance in graphs, with sufficient explanation and many examples, is given in Chapter 15 in [24].

In this paper, we will deal with the so called path-length-weighted distance, which was introduced in [23] (§.4).

It should be noted that the version defined there was given for non-directed graphs, and so the definition we use is slightly different. However, we will also define an extended quasi-metric in this case by giving the value $d(a,b) = \infty$ when there is no path for going from a to b, and considering only allowed paths between vertices.

Since we are interested in how to compute the distance and not in theoretical questions about its metric space structure, we will focus our attention on the computational algorithm. All the notions on graph theory that are needed can be found in books on this subject, such as, for example, [7]. We will introduce some of them in the next section.

2. Main Definitions and Context

Take a weighted directed acyclic graph $G = (V, E)$, where V is the set of vertices (nodes), and E is the class of edges among them. Let us define what we call a proximity function. This is an extended function $\phi : V \times V \to \mathbb{R}^+ \cup \{\infty\}$ that describes by means of a non-negative real number a relation among node a and node b. It is supposed that ϕ represents some sort of distance among the nodes, but—and this is crucial—it is not assumed to be a distance (that is, none of the axioms of the metric are assumed). In the case where a and b are not connected in the graph, we will write $0, 1$, or ∞ depending on how we write the algorithm, that is, how we decide to codify the links in the graph. So, we will consider structures (V, E, ϕ) when the graph is provided with a proximity function ϕ.

Consider a non-increasing sequence $W := (W_i)_{i=1}^{\infty}$ of positive real numbers that will be assumed to be the weights associated to the lengths of the paths considered in the definition of a distance starting from the proximity function ϕ. Although all the graphs are finite, we will often define W as an infinite sequence using only its first elements. Given two vertices $a, b \in V$, we define the set of all possible paths from a to b as

$$\mathcal{P}(a,b) = \{P = (x_0, x_1, \ldots, x_n) : \; x_0 = a, \; x_n = b, (x_{i-1}, x_i) \in E, \; n \in \mathbb{N}\}.$$

The length of a path $P = (x_0, x_1, \ldots, x_n)$ is denoted by $l(P) = n$, and its total distance is denoted by

$$d(P) = W_n \cdot s(P),$$

where $s(P) = \sum_{i=1}^{n} \phi(x_{i-1}, x_i)$ is the *sum of path weights* of the path P. Given two points $a, b \in V$, we define an extended quasi-metric in V as the function on $V \times V$ by

$$d_\phi(a,b) = \inf\{d(P) : \; P \in \mathcal{P}(a,b)\},$$

with the convention that $\inf\{\emptyset\} = \infty$. As in the original definition in [23] (§.4), we will call it the path-length-weighted quasi-metric. The reader can find there the proof that it defines a metric. Given two paths $P = (x_0, x_1, \ldots, x_n) \in \mathcal{P}(a,b)$ and $Q = (y_0, y_1, \ldots, y_m) \in \mathcal{P}(b,c)$, we define the concatenation of both as

$$P \sqcup Q = (x_0, x_1, \ldots x_n, y_1, \ldots, y_m).$$

Write $V = \{v_i : i = 0, ..., n\}$ for the vertices of the graph. As in the classical math distance case, we are interested in this paper in computing the distance from any of the points in V to v_0.

3. Previous to the Algorithm

Consider the weighted directed graph without cycles $G = (V, E)$ and fix a vertex to which we want to measure the distance from any other vertex of the graph. Write $V = \{v_i : i = 0, ..., n\}$, and suppose that we want to compute the distance from any other node of the graph to v_0.

In both the Bellman–Ford and Dijkstra algorithms, the shortest distances from the source node v_0 to all other nodes in a graph are computed by selecting the closest unvisited node to the current node. In this process, all edges leaving it to unvisited nodes are examined, updating the distance from these nodes if the distance through the selected node is less than the currently known distance. Thus, the updated distance is given by the path from this node to the current node and the path of least distance between the current node and the source.

We introduce the following formal operation $\boxplus$ to describe the updated distance calculation in a simpler way. Fix a vertex $v_0 \in V$, and let us write $v_{selected}$ for the vertex from which we want to compute the distance to v_0, and $v_{current}$ is another vertex that belongs to one of the paths from $v_{selected}$ to v_0. Let us define

$$(d(R), l(R)) \boxplus (d(P), l(P)) := \left(W_{l(P)+l(R)} \cdot d(R) + \frac{W_{l(P)+l(R)}}{W_{l(P)}} \cdot d(P),\ l(P) + l(R)\right), \tag{1}$$

where $R \in \mathcal{P}(v_{selected}, v_{current})$ and $P \in \mathcal{P}(v_{current}, v_0)$ define the path with the shortest distance between the current node and the source.

This weighted sum substitutes the normal addition used in the Bellman–Ford algorithm, and it allows us to change the weights when the length of the path increases. Unlike the Bellman–Ford algorithm, this algorithm has the additional complexity in that the path with the shortest distance between two nodes does not have to be the one that passes through the nodes with the shortest distances, since the number of steps must be taken into account. As illustrated in Example 1, replacing the normal sum of the Bellman–Ford algorithm with the weighted sum is not sufficient, as it is crucial to take the path length into account in each distance calculation to ensure that no path possibility is discarded.

Example 1. *Consider the graph in Figure 1. We will calculate the path-length-weighted distance from v_5 to v_0. For this purpose, we consider the weights associated with the lengths of the paths as $W := \left(\frac{1}{t}\right)_{t=1}^{\infty}$. This means that the distance between nodes is the average of the path weights.*

The first step in calculating the distance has to take into account, of course, the only way forward in the graph: going from v_5 to v_0. For the following steps, we clearly have that the path with the smallest distance between the nodes v_4 and v_0 is

$$P := (v_4, v_3, v_0),$$

with a length $l(P) = 2$ and a distance $d(P) = \frac{3+1}{2} = 2 = d_\phi(v_4, v_0)$.

Since v_5 is only directly connected to v_4, when we apply a Bellman–Ford-type algorithm, the computation of $d_\phi(v_5, v_0)$ is determined by the path $Q = (v_5, v_4) \sqcup P$, which results in

$$(d(P), l(Q)) = (\phi(v_5, v_4), 1) \boxplus (d(P), l(P)) = \left(\frac{\phi(v_5, v_4) + 3 + 1}{3}, l(P) + 1 \right) = (8, 3).$$

In spite of this, the path with the shortest distance between the nodes v_5 and v_0 is

$$Q' = (v_5, v_4, v_2, v_1, v_0),$$

with a length $l(Q') = 4$ and an associate distance $d(Q') = \frac{20+3+4+2}{4} = 7.25 < 8 = d(Q)$.

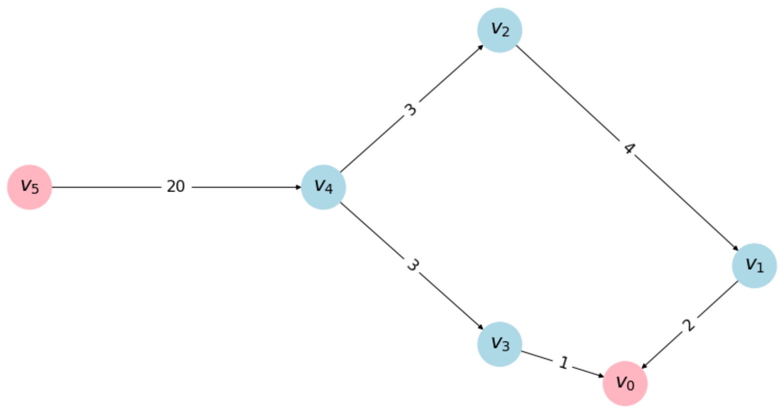

Figure 1. A graph in which a Bellman–Ford-type algorithm used to calculate the smallest distance between two nodes would not work. The weight of each edge is written on the corresponding arrow.

Consequently, as shown in Example 1, the shortest distance from an unvisited node to the current node for the metric presented in this article does not have to be calculated from the path with the shortest distance between the current node and the source. Given three vertices $a, b, c \in V$, this implies that the path with the shortest distance from a to c passing through b is not necessarily the union of the paths with the shortest distance from a to b and from b to c. In order to not consider all possible paths in the calculation of the distances, multi-objective optimization can be used, which allows us to restrict attention to the set of possible paths with the smallest distance and to make tradeoffs within this set.

Assume that, at a particular step of the algorithm, two or more paths $P, Q \in \mathcal{P}(a, b)$ are found between the vertices $a, b \in V$. Define on $\mathcal{P}(a, b)$ an order given by $P \preceq Q$ if and only if

$$l(P) \geq l(Q) \text{ and } s(P) \leq s(Q). \tag{2}$$

It is straightforward to see that it is an order relation. Note also that it implies that $d(P) \leq d(Q)$. The following result shows that it is sufficient to explore the minimal paths of $\mathcal{P}(a, b)$ (in the sense defined by $\preceq$), i.e., those residing in the following Pareto front,

$$\mathcal{P}^*(a, b) = \{P \in \mathcal{P}(a, b) : \{P' \in \mathcal{P}(a, b) : P' \preceq P, P \neq P'\} = \emptyset\}. \tag{3}$$

This front is illustrated in Figure 2.

Proposition 1. *Consider a graph G and the corresponding elements defined above. Fix $a, b \in V$, and let $P, P' \in \mathcal{P}(a, b)$ such that $P \preceq P'$. Then, for any $c \in V$ and $R \in \mathcal{P}(b, c)$, we have that the paths $Q = P \sqcup R$ and $Q' = P' \sqcup R$ in $\mathcal{P}(a, c)$ satisfy $Q \preceq Q'$, and, in particular, $d(Q) \leq d(Q')$.*

This means that, once you have found a better path (in terms of $\preceq$), you can discard the worst one (and stop the exploration), as it will not be used to find any distance between the last vertices.

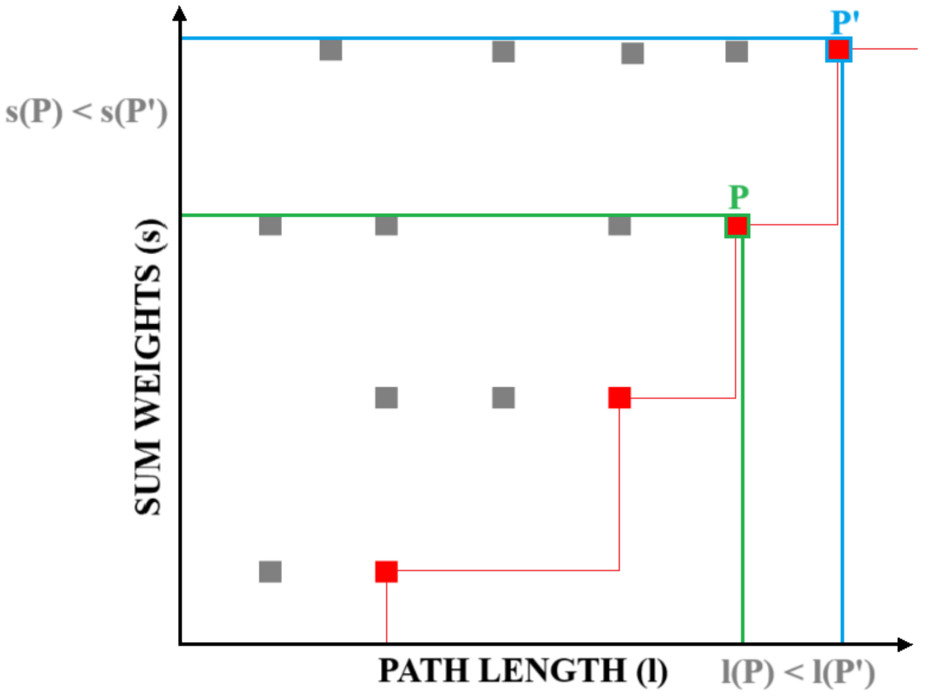

Figure 2. Example of a Pareto front associated with the possible paths in a graph. The framed dots represent all possible paths from one node to another. The red dots represent the set of paths needed to calculate distances, with $\mathcal{P}^*$ being the set of all of them.

Proof. Consider $P = (x_0, x_1, \ldots, x_n)$, $P' = (x'_0, x'_1, \ldots, x'_m)$, and $R = (y_0, y_1, \ldots, y_r)$ in the statement. Then, we have that

$$d(Q) = W_{l(Q)} \cdot \left(\sum_{i=1}^{n} \phi(x_{i-1}, x_i) + \sum_{i=1}^{r} \phi(y_{i-1}, y_i) \right),$$

$$d(Q') = W_{l(Q')} \cdot \left(\sum_{i=1}^{m} \phi(x'_{i-1}, x'_i) + \sum_{i=1}^{r} \phi(y_{i-1}, y_i) \right).$$

Recall that $P \preceq P'$. Then, $l(Q) = l(P) + l(R) \geq l(P') + l(R) = l(Q')$, and so $W_{l(Q)} \leq W_{l(Q')}$. Moreover,

$$\sum_{i=1}^{n} \phi(x_{i-1}, x_i) = s(P) \leq s(P') = \sum_{i=1}^{m} \phi(x'_{i-1}, x'_i).$$

Thus, we obtain $Q \preceq Q'$. $\square$

Remark 1. *It is easy to see that Proposition 1 also works if the composition of the paths is done in the opposite way. That is, the same conclusion holds if $Q = R \sqcup P$ and $Q' = R \sqcup P'$ belong to $\mathcal{P}(a, c)$, with $R \in \mathcal{P}(a, b)$ and $P, P' \in \mathcal{P}(b, c)$.*

4. The Algorithm

Taking into account the ideas of the previous section, in this part of the paper, we explain the algorithm to compute the path-length-weighted distance in a graph. To make the new algorithm more efficient, the sum of the weights—$s(P)$—will be used in the calculations instead of the distance—$d(P)$—which will be calculated in the last step. The counting parameter m will be used to indicate the step number in the algorithm, that is, the length of the path. Let $V = \{v_i : i = 0, \ldots, n\}$ denote the set of all nodes, where v_0 represents the source node. In this context, we define D_i, with $v_i \in V$, as the set of all computed tuples as (s_i, l_i), where (to simplify notation) s_i denotes the sum of all weights of a path $P \in \mathcal{P}(v_i, v_0)$, and l_i represents its length. We introduce the following formal operation $\oplus$ to describe the process of adding a new node in terms of the sum of all weights. Once a couple of adjacent vertices v_i, v_j are fixed, we write

$$\phi(v_j, v_i) \oplus (s_i, l_i) := \big(s_i + \phi(v_j, v_i), l_i + 1\big). \tag{4}$$

This definition is closely related to (but of course not the same as) that of $\boxplus$ given in Equation (1), but, as we noted above, it is written in terms of s rather than d. Finally, we set the (ordered) set $d_\phi = \{d_\phi(v_i, v_0) : v_i \in V\}$.

The new algorithm that we propose follows the next steps. The parameter $m \in \mathbb{N}$ gives the size of the paths that are considered at each step connecting any vertex v_i with the source node v_0:

(1) Initialization: For $m = 0$, initialize the algorithm by doing $D_i = \{(\infty, 0)\}$ for $i = 1, ..., n-1$, and $D_0 = \{(0,0)\}$. We have that $l_i = 0$ for all i, since we are considering steps of length equal to 0 at the beginning. Now, consider the "set of vertices connected to v_0 by a path of length 0", that is, $\{v_0\}$. Also, put, according to the definition, $s_0 = l_0 = 0$.

(2) Search for all paths: Put $m = 1$. Consider the set of adjacent vertices to v_0,

$$A_0 := \{v_j \in V \setminus \{v_0\} : \phi(v_j, v_0) \in \mathbb{R}^+\},$$

that is, all the vertices that are connected to v_0. For $v_j \in A_0$, consider the quantities

$$\phi(v_j, v_0) \oplus (s_0, l_0) = (\phi(v_j, v_0), 1)$$

given by Equation (4) and add them to the set D_j that has been initialized as only containing $(\infty, 0)$, that is,

$$D_j = D_j \cup \{(\phi(v_j, v_0), 1)\}.$$

(3) Path filtering: The main idea of the algorithm is to update the sets D_j as m grows when repeating step (2). For a given m following 1 (make $m = 2$ for the immediate step), we have that D_j is the set of tuples (s, l) of all paths $P \in \mathcal{P}(v_j, v_0)$ of lengths less than or equal to m. We need to reduce the size of these sets by eliminating some non-essential elements. According to Proposition 1, Equation (3) will be used to discard the tuples belonging to those paths that give a greater distance than others. We do that by replacing

$$D_j \quad \text{by} \quad \mathcal{P}^*(D_j),$$

which is defined for any D_j as

$$\mathcal{P}^*(D_j) = \big\{(s, l) \in D_j : \{(s', l') \in D_j : \ (s', l') \neq (s, l), \ s' \leq s \text{ and } l' \geq l\} = \emptyset\big\}.$$

Note that in the case that a given v_j has any path of size m go to v_0, this step will remove the tuple $(\infty, 0)$ from the set D_j.

(4) Repeat: Repeat steps (2) and (3) until m is greater than the maximum length of all paths in the graph connecting v_0.

(5) Distance computation: Finally, once the final versions of the sets D_i have been calculated, we proceed to the calculation of the distances. Then, for each $v_i \in V$, the distance from the node v_i to the source node v_0 is given by

$$d_\phi(v_i, v_0) = \min_{(s_i, l_i) \in D_i} \{W_{l_i} \cdot s_i\}.$$

Note that this way of calculating the distance, although it can be laborious, guarantees that it works for any non-increasing sequence of weights W.

The reader can find below Algorithm 1, which is a visual representation illustrating the steps of the proposed algorithm. This diagram serves as a helpful visual aid to complement the step-by-step explanation provided above.

Algorithm 1: Path length weight (ϕ, W, n, v_0)

```
Input: $\phi, n, W, v_0$
$V \leftarrow \{v_i : i \in \{0, \dots n\}\}$;
$D \leftarrow \{D[v_i] = \{[\text{inf}, 0]\} : i \in \{1, \dots n\}\}$;
$D[v_0] \leftarrow \{[0, 0]\}$;
$Q \leftarrow \{v_0\}$;
$m \leftarrow 1$;
while $(m \leq n)$ and $(Q \neq \emptyset)$ do
    for $v_i$ in $Q$ do
        $A_i \leftarrow \{v_j \in V - \{v_i\} : \phi[v_j, v_i] \neq \text{inf}\}$;
        for $v_j$ in $A_i$ do
            for $[s_i, l_i]$ in $D[v_i]$ do
                $s \leftarrow s_i + \phi[v_j, v_i]$;
                if $[s, l_i + 1] \notin D[v_j]$ then
                    Add $[s, l_i + 1]$ to $D[v_j]$;
                end
            end
            $D[v_j] \leftarrow \mathcal{P}^*(D[v_j])$;
        end
    end
    $Q \leftarrow \{v_i \in V - Q : [\text{inf}, 0] \notin D[v_i]\}$;
    $m \leftarrow m + 1$;
end
$d \leftarrow (0, ..., n)$
for $v_i$ in $V$ do
    $d[v_i] \leftarrow \min(W[l_i] \cdot s_i : [s_i, l_i] \in D[v_i])$
end
return $d$;
```

5. Examples of Graphs

In this section, we will present some examples to show the properties of the explained algorithm. We will explore some different cases in order to demonstrate that the conditions of the requirements are necessary to obtain a suitable algorithm. Also, our idea is to present different classes of graphs, which can be associated using topological properties, to show that our technique works in different situations.

We start with an example of a tree-type graph. After that, we will show that our algorithm also works for a centered-star-type graph. Finally, we will also present a case of a graph with no singular topological structure to prove the generality of our procedure.

To visualize the graphs, the Kamada–Kawai layout algorithm [25] has been used, which is a method for drawing two-dimensional graphs where the nodes are placed so that the distances in the drawing are proportional to the shortest distances between them in the original graph. In this way, it is possible to observe how the drawing of the graph changes depending on the distance used. The numbers shown on the arrows are the corresponding values of ϕ, except where other notation is explained. Let us remark that the "distances" that are written in the figures are the ones that can be computed: recall again that we are working with directed graphs with no cycles, so it may happen that we can compute the "distance" from v_1 to v_2 but not the "distance" from v_2 to v_1. In this case, we write in the corresponding edge the one that can be computed:

(1) Tree-type graph: In this type of graph, all nodes are connected to each other by exactly one single path, and there are no cycles (closed paths). For the calculation of the distance, we use the weights associated with the inverse of the lengths of the paths—this is $W := \left(\frac{1}{t}\right)_{t=1}^{\infty}$. The comparison among Figures 3 and 4 shows the particular nature of the metric we have defined, in which two points v_0 and v_i that are more distanced in the tree (with more intermediate nodes) are, however, nearer than one of the vertices that crosses the path that connects v_0 with v_i. The reader can see this, for example, in the comparison of the position and distances between v_0 and v_{11} (the path-length-weighted distance is 2.333), and between v_0 and v_{13} (the path-length-weighted distance is 2.250). Moreover, the distances between v_0 and v_1 and between v_0 and v_3 are equal, although it is necessary to pass through v_1 to reach v_0 from v_3. This effect is especially remarkable in the case of trees, where the usual path distance between a vertex and the root increases with the number of branches separating them. We have seen that for our distance, however, this is not necessarily true.

(2) Centered-star-type graph: In these graphs, there is a central node—v_0 in Figure 5—which is directly connected to all other nodes in the graph. Compared to Figure 6, where the weights $W = \left(\frac{1}{t}\right)_{t=1}^{\infty}$ were used, Figure 7 shows how the graph changes when applying the distance calculation with the weights $W = \left(\frac{1}{t^2}\right)_{t=1}^{\infty}$ In this case, the distance decreases as the path length increases. This is again a relevant difference with the usual weighted path distance case, and it could be used to model different network behaviors where proximity in terms of number of intermediate nodes does not adequately represent the desired relationships between nodes.

(3) Graph with no singular topological structure: Let us now consider a graph that does not have an easily definable shape or pattern, such as a tree or a star. The example that we chose is shown in Figure 8. It is a non-connected graph. This graph can be separated into two connected subgraphs—that is, its two connected components—to calculate the distances of the nodes. In spite of this, as can be seen in Figure 9, our algorithm also worked without the need to separate these graphs into their connected components. The weights $W = \left(\frac{1}{t}\right)_{t=1}^{\infty}$ were again considered.

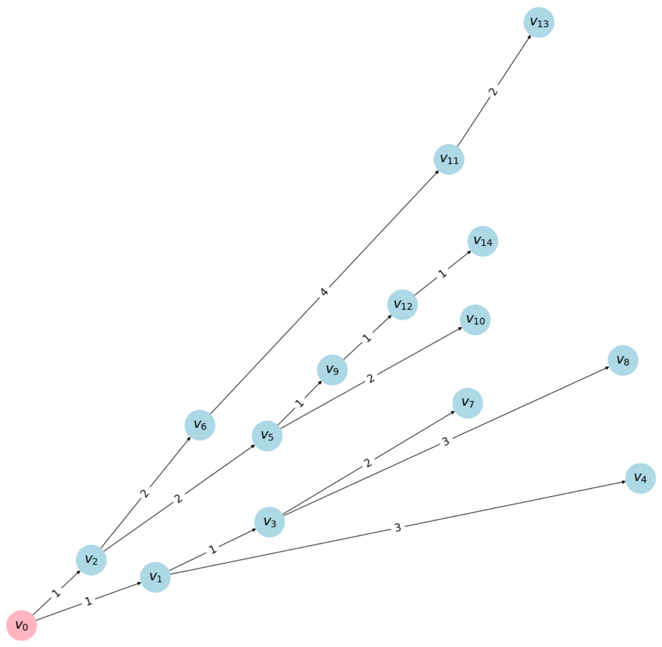

Figure 3. Example of a tree-type graph.

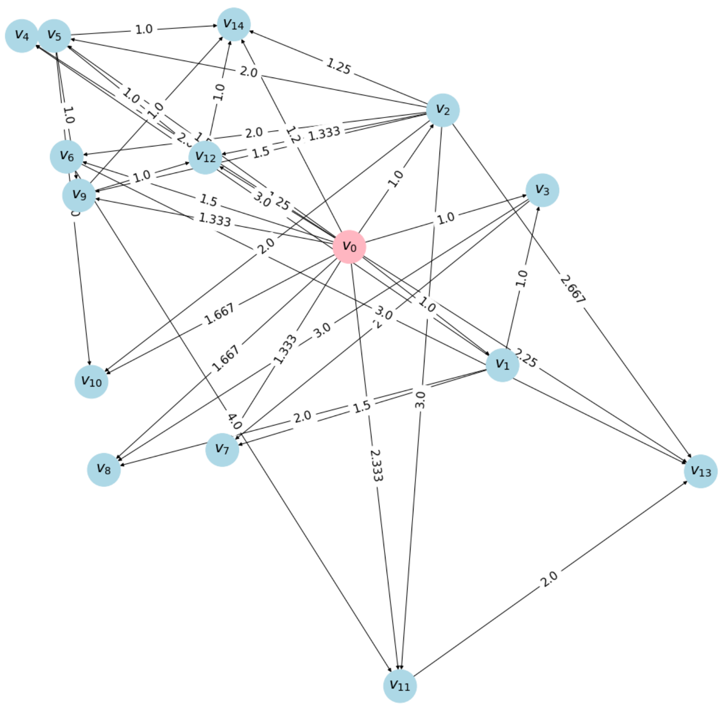

Figure 4. Distances calculated from the graph in Figure 3. In this case, the numbers appearing in the arrows are the quasi-metrics and not the values of ϕ.

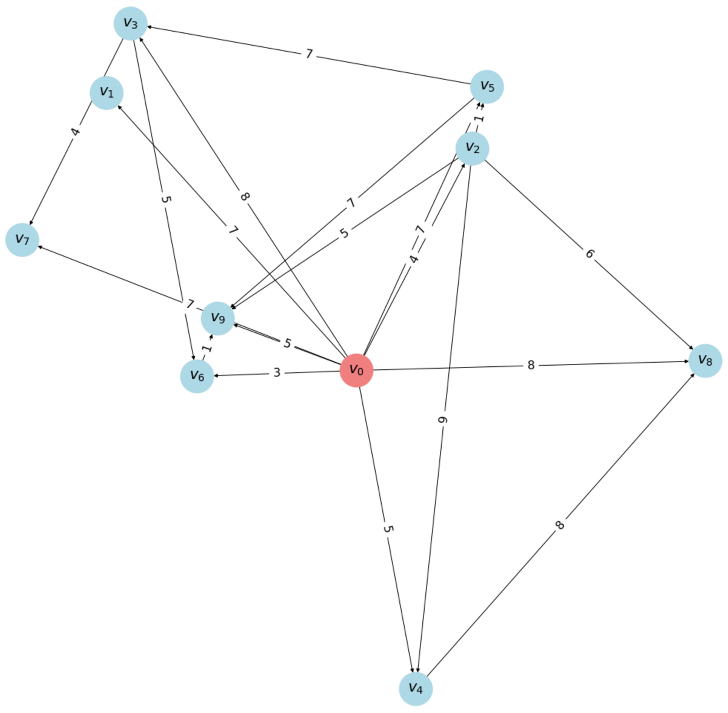

Figure 5. Example of a centered-star-type graph.

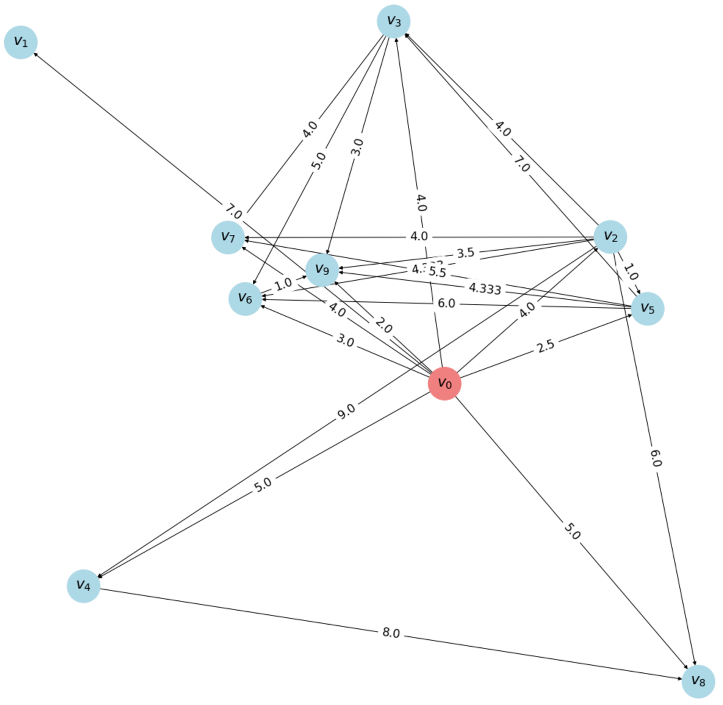

Figure 6. Distances calculated for the graph in Figure 5 for $W = \left(\frac{1}{t}\right)_{t=1}^{\infty}$.

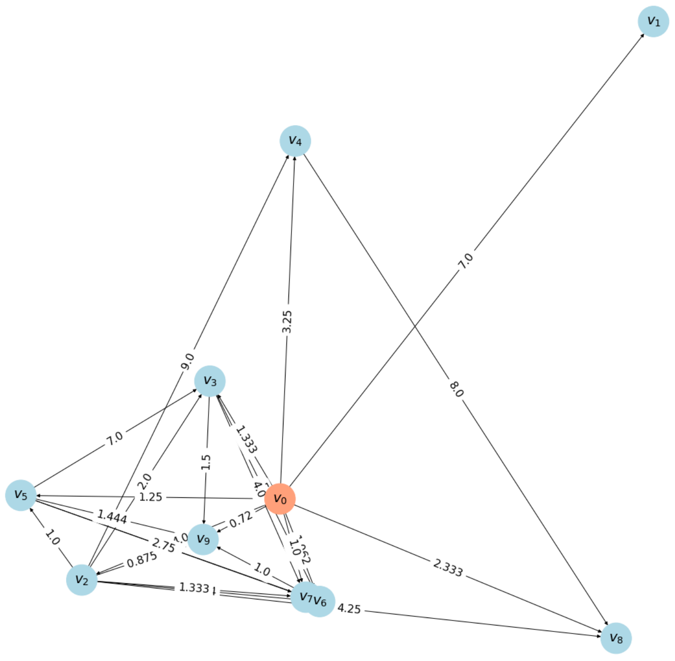

Figure 7. Distances calculated for the graph in Figure 5 using the square of the inverse of the path lengths $W = \left(\frac{1}{t^2}\right)_{t=1}^{\infty}$.

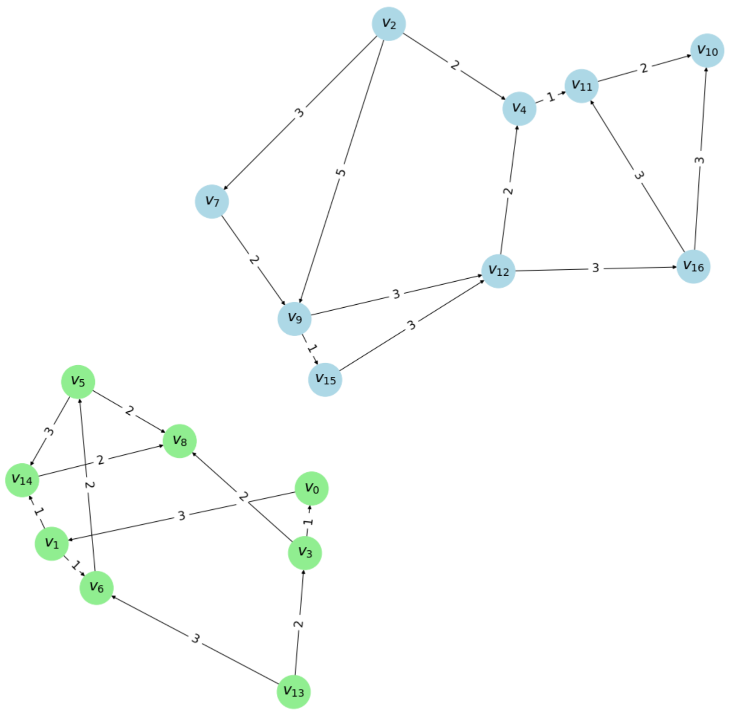

Figure 8. Example of a graph with no singular topological structure.

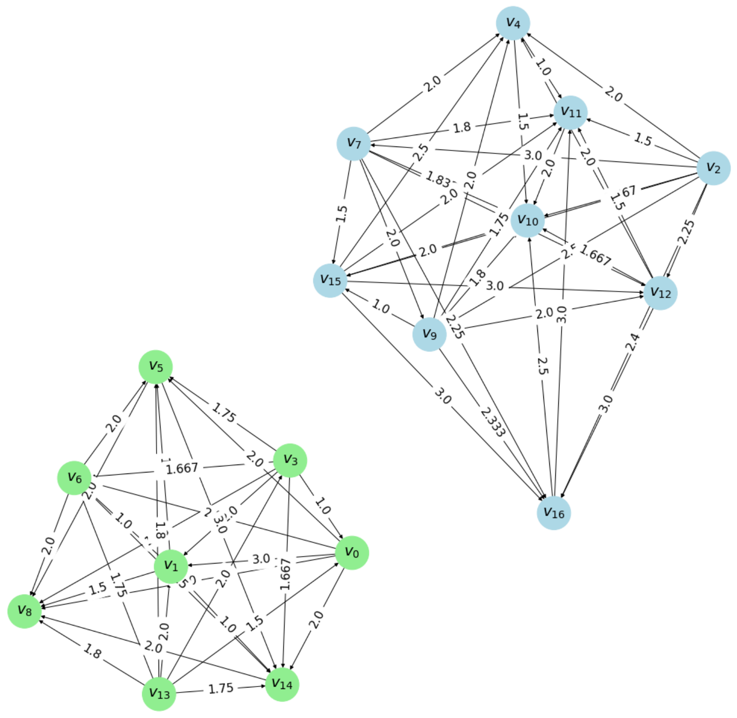

Figure 9. Distances calculated for the graph in Figure 8.

6. Special Cases: Constraints on the Weights and on the Proximity Matrix to Improve the Efficiency of the Algorithm

To reduce the complexity of the algorithm, algebraic conditions can be imposed on both the proximity matrix and the weights associated with the length of the paths. The trivial case is when these weights are all equal to a constant W. In this case, the distance coincides with the weighted path metric (see [24], p. 258) multiplied by the constant W. Therefore, the calculation of the distance can be performed using the Bellman–Ford or Dijkstra algorithms.

In this section, we will consider the weights associated with the inverse of a power of the lengths of the paths $W := \left\{\frac{1}{t^k}\right\}_{t=1}^{\infty}$, with the power k being greater than or equal to 1. The importance of these weights lies in the fact that it is the natural generalization of the case where the power k is equal to 1, i.e., the case where the distance between nodes coincides with the average of the path weights. This class of weights, which have been already used in the previous sections as general examples, is also relevant for applications (see [23]).

The results provided in this section can be implemented directly in the algorithm to make it faster. We explain the requirements that have to be met, as well as the procedures for using them.

Proposition 2. *Consider a graph $G = (V, E)$ and the weights associated with the lengths of the paths $W := \left\{\frac{1}{t^k}\right\}_{t=1}^{\infty}$, with $k \geq 1$. Fix $a, b \in V$, and let $P, P' \in \mathcal{P}(a, b)$ such that $l(P) \geq l(P')$, and $d(P) \leq d(P')$. Then, for any $c \in V$ and $R \in \mathcal{P}(b, c)$ such that $d(P') \leq d(R)$, we have that the paths $Q = P \sqcup R$ and $Q' = P' \sqcup R$ in $\mathcal{P}(a, c)$ satisfy $d(Q) \leq d(Q')$.*

Proof. Consider $P = (x_0, x_1, \ldots, x_n)$, $P' = (x'_0, x'_1, \ldots, x'_m)$, and $R = (y_0, y_1, \ldots, y_r)$ in the statement so that $l(P) = n$, $l(P') = m$, and $l(R) = r$. Then,

$$d(Q) = d(P \sqcup R) = \frac{\sum_{i=1}^{n} \phi(x_{i-1}, x_i) + \sum_{i=1}^{r} \phi(y_{i-1}, y_i)}{(n+r)^k} = \frac{n \cdot d(P) + r \cdot d(R)}{(n+r)^k},$$

$$d(Q') = \frac{m \cdot d(P') + r \cdot d(R)}{(m+r)^k}.$$

Then, by hypothesis,

$$\begin{aligned} d(P) &\leq \frac{m \cdot d(P')}{m} = \\ &= \frac{m}{n \cdot m}\big(m \cdot d(P')\big) + \frac{n-m}{n \cdot m}\big(m \cdot d(P')\big) \leq \\ &\leq \frac{m \cdot d(P')}{n} + \frac{n-m}{n} d(R). \end{aligned}$$

This implies that $n \cdot d(P) \leq m \cdot d(P') + (n-m) d(R)$. Therefore,

$$\begin{aligned} (n+r)^k (m+r) d(Q) &= (n+r)^k (m+r) \left(\frac{n \cdot d(P) + r \cdot d(R)}{(n+r)^k}\right) = \\ &= n \cdot m \cdot d(P) + n \cdot r \cdot d(P) + (m+r)(r \cdot d(R)) \leq \\ &\leq n \cdot m \cdot d(P') + r \cdot m \cdot d(P') + (n-m)(r \cdot d(R)) + (m+r)(r \cdot d(R)) = \\ &= (n+r)\big(m \cdot d(P') + r \cdot d(R)\big) = \\ &= (n+r)(m+r)^k d(Q') \leq \\ &\leq \left(\frac{n+r}{m+r}\right)^{k-1} (n+r)(m+r)^k d(Q') = \\ &= (n+r)^k (m+r) d(Q'), \end{aligned}$$

so $d(Q) \leq d(Q')$. □

Proposition 3. *Consider a graph $G = (V, E)$ and the weights associated with the lengths of the paths $W := \left\{\frac{1}{t^k}\right\}_{t=1}^{\infty}$, with $k \geq 1$. Fix $a, b \in V$, and let $P, P' \in \mathcal{P}(a, b)$ such that $l(P) \leq l(P')$, and $d(P) \leq d(P')$. Then, for any $c \in V$ and $R \in \mathcal{P}(b, c)$ such that $d(R) \leq d(P)$, we have that the paths $Q = P \sqcup R$ and $Q' = P' \sqcup R$ in $\mathcal{P}(a, c)$ satisfy $d(Q) \leq d(Q')$.*

The proof of this result is analogous to Proposition 2, so it will be omitted.

Remark 2. *It is trivial to see that Propositions 2 and 3 also work if the paths $Q = R \sqcup P$ and $Q' = R \sqcup P'$ in $\mathcal{P}(a,c)$ are used, being in this case $R \in \mathcal{P}(a,b)$ and $P, P' \in \mathcal{P}(b,c)$.*

These propositions propose to define in $\mathcal{P}(a,b)$ and for each $R \in \mathcal{P}(b,c)$ the order relation $P \preceq^1_R Q$ given by the conditions

$$l(P) \geq l(Q) \text{ and } d(P) \leq d(Q) \leq d(R)$$

(suggested by Proposition 2), and the order relation $P \preceq^2_R Q$ given by

$$l(P) \leq l(Q) \text{ and } d(R) \leq d(P) \leq d(Q)$$

(suggested by Proposition 3).

As can be seen, these order relations are more restrictive than the order relation $\preceq$ used in the previous sections (although they depend on the path R), since they are given as a function of the distance d, while $\preceq$ is a function of the sum of the weights s.

To reduce the set of paths suitable for calculating the distance with our algorithm using these order relations, we need them to not depend on the path R. This can be done by imposing some additional properties to the graph. In particular, this works if the graph meets one of the following conditions, which are different depending on the order relation we will use:

1 Requirements for the order $\preceq^1_R$: Let $G = (V, E)$ be a weighted directed graph with no cycles. The condition we impose in this case is the following: for all a, b, c in V, with $\mathcal{P}(a,b)$ being non-empty and b connected to c—that is, $(b,c) \in E$—the inequality

$$d(P) \leq \phi(b,c) \tag{5}$$

holds for all $P \in \mathcal{P}(a,b)$. Note that this requirement is independent of R, so we write $\preceq^1$ for the associate order relation. As it is not always possible to verify this condition given a graph, we will check that for all $(a,b), (b,c) \in E$, it is satisfied that

$$\phi(a,b) \leq \phi(b,c). \tag{6}$$

It is obvious that this condition implies (5). Using this, we can simplify the algorithm. Clearly, in this case, it is enough to explore the paths of the following Pareto front:

$$\mathcal{P}^*_1(a,b) = \Big\{ P \in \mathcal{P}(a,b) : \{P' \in \mathcal{P}(a,b) : P' \preceq^1 P, P \neq P'\} = \emptyset \Big\}. \tag{7}$$

2 Requirements for the order $\preceq^2_R$: As above, let $G = (V, E)$ be a weighted directed graph with no cycles. The requirement is, in this case, the following: for all a, b, c in V, with $\mathcal{P}(a,b)$ being non-empty and b connected to c—that is, $(b,c) \in E$—the inequality

$$d(P) \geq \phi(b,c) \tag{8}$$

has to be satisfied for all $P \in \mathcal{P}(a,b)$. A sufficient condition to be checked is that for all $(a,b), (b,c) \in E$, it is satisfied that $\phi(a,b) \geq \phi(b,c)$. Again, this implies (8). In this case, it is enough to consider the following paths:

$$\mathcal{P}^*_2(a,b) = \Big\{ P \in \mathcal{P}(a,b) : \{P' \in \mathcal{P}(a,b) : P' \preceq^2 P, P \neq P'\} = \emptyset \Big\}. \tag{9}$$

To show how the path filters explained above work for each of the algorithms, we will calculate the distance from node v_0 to v_6 in the graph illustrated in Figure 10 (which satisfies the requirements given by (6)) using the weights associated with the path lengths

$W := \left(\frac{1}{t}\right)_{t=1}^{\infty}$. Since v_6 is only directly connected by v_5, the computation of $d_\phi(v_0, v_6)$ is determined by the paths $Q = P \sqcup (v_5, v_6)$, with $P \in \mathcal{P}(v_0, v_5)$:

(1) $P_1 = (v_0, v_1, v_3, v_5)$, with $l(P_1) = 3$, $s(P_1) = 15$, and $d(P_1) = 5$;
(2) $P_2 = (v_0, v_2, v_3, v_5)$, with $l(P_2) = 3$, $s(P_2) = 18$, and $d(P_2) = 6$;
(3) $P_3 = (v_0, v_2, v_4, v_5)$, with $l(P_3) = 3$, $s(P_3) = 21$, and $d(P_3) = 7$;
(4) $P_4 = (v_0, v_3, v_5)$, with $l(P_4) = 2$, $s(P_4) = 13$, and $d(P_4) = 6.5$;
(5) $P_5 = (v_0, v_5)$, with $l(P_5) = 1$, $s(P_5) = 5$, and $d(P_5) = 5$.

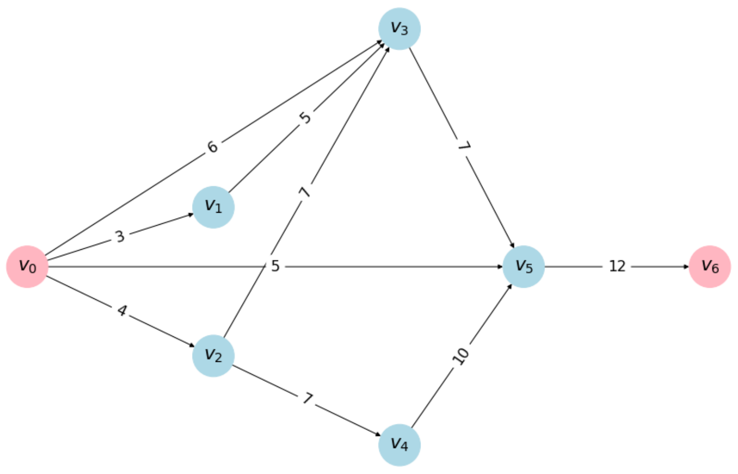

Figure 10. Example of a graph satisfying (6).

In this case, if we use the general path filtering given by (3), we would obtain the possible paths

$$\mathcal{P}^*(v_0, v_5) = \{P_1, P_4, P_5\},$$

since $P_1 \preceq P_2 \preceq P_3$.

Instead, if we use the path filtering for this particular case given by (7), we have that $P_1 \preceq^1 P_2 \preceq^1 P_3$, $P_1 \preceq^1 P_4$, and $P_1 \preceq^1 P_5$, obtaining directly

$$\mathcal{P}_1^*(v_0, v_5) = \{P_1\}.$$

Therefore, for the calculation of $d_\phi(v_0, v_6)$, we only have to compute the distance associated with the path $Q = P_1 \sqcup (v_5, v_6)$, that is, $d_\phi(v_0, v_6) = d(Q)$.

Author Contributions: Software, R.A. and S.S.; Validation, R.A., L.M.G.-R. and S.S.; Formal analysis R.A., E.A.S.P. and S.S.; Conceptualization, J.M.C. and L.M.G.-R.; Investigation, J.M.C. and E.A.S.P.; Writing—review and editing, J.M.C.; Writing—original draft, E.A.S.P. and S.S.; supervision, E.A.S.P.; Visualization, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Agencia Estatal de Investigación under grant number PID2022-138342NB-I00. The research of the first author was funded by the Universitat Politècnica de València through the Programa de Ayudas de Investigación y Desarrollo (PAID-01-21). The research of the other authors was also funded by the European Union's Horizon Europe research and innovation program under the Grant Agreement No. 101059609 (Re-Livestock).

Data Availability Statement: All the algorithms presented in this paper are available in a GitHub repository. It can be accessed through the following link: https://github.com/serjj99/path_length_weighted_distance (accessed on 12 July 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bellman, R. On a Routing Problem. *Q. Appl. Math.* **1958**, *16*, 87–90. [CrossRef]
2. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
3. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2001.
4. Entringer, R.C.; Jackson, D.E.; Snyder, D.A. Distance in graphs. *Czechoslov. Math. J.* **1976**, *26*, 283–296. [CrossRef]
5. Hakimi, S.L.; Yau, S.S. Distance matrix of a graph and its realizability. *Q. Appl. Math.* **1965**, *22*, 305–317. [CrossRef]
6. Klein, D.J. Graph geometry via metrics. In *Topology in Chemistry*; Woodhead Publishing: Sawston, UK, 2002; pp. 292–315.
7. Brandes, U. *Network Analysis: Methodological Foundations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005.
8. Chen, J.; Safro, I. Algebraic distance on graphs. *SIAM J. Sci. Comput.* **2011**, *33*, 3468–3490. [CrossRef]
9. Goddard, W.; Oellermann, O.R. Distance in graphs. In *Structural Analysis of Complex Networks*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 49–72.
10. Barnes, J.A.; Harary, F. Graph theory in network analysis. *Soc. Netw.* **1983**, *5*, 235–244. [CrossRef]
11. Harary, F.; Norman, R.Z. *Graph Theory as a Mathematical Model in Social Science*; Institute for Social Research, No. 2; University of Michigan: Ann Arbor, MI, USA, 1953.
12. Stephenson, K.; Zelen, M. Rethinking centrality: Methods and examples. *Soc. Netw.* **1989**, *11*, 1–37. [CrossRef]
13. Klein, D.J.; Randić, M. Resistance distance. *J. Math. Chem.* **1993**, *12*, 81–95. [CrossRef]
14. Chebotarev, P. A class of graph-geodetic distances generalizing the shortest-path and the resistance distances. *Discret. Appl. Math.* **2011**, *159*, 295–302. [CrossRef]
15. Yang, Y.; Klein, D.J. Two-point resistances and random walks on stellated regular graphs. *J. Phys. A Math. Theor.* **2019**, *52*, 075201. [CrossRef]
16. Bozzo, E.; Franceschet, M. Resistance distance, closeness, and betweenness. *Soc. Netw.* **2013**, *35*, 460–469. [CrossRef]
17. Bu, C.; Yan, B.; Zhou, X.; Zhou, J. Resistance distance in subdivision-vertex join and subdivision-edge join of graphs. *Linear Algebra Appl.* **2014**, *458*, 454–462. [CrossRef]
18. Yang, Y.; Klein, D.J. Comparison theorems on resistance distances and Kirchhoff indices of S, T-isomers. *Discret. Appl. Math.* **2014**, *175*, 87–93. [CrossRef]
19. Oehlers, M.; Fabian, B. Graph metrics for network robustness—A survey. *Mathematics* **2021**, *9*, 895. [CrossRef]
20. Mester, A.; Pop, A.; Mursa, B.E.M.; Greblă, H.; Diosan, L.; Chira, C. Network analysis based on important node selection and community detection. *Mathematics* **2021**, *9*, 2294. [CrossRef]
21. Buckley, F.; Harary, F. *Distance in Graphs*; Addison-Wesley: Redwood City, CA, USA, 1990.
22. Fouss, F.; Saerens, M.; Shimbo, M. *Algorithms and Models for Network Data and Link Analysis*; Cambridge University Press: Cambridge, UK, 2016.
23. Calabuig, J.M.; Falciani, H.; Sapena, A.F.; Raffi, L.G.; Sánchez Pérez, E.A. Graph distances for determining entities relationships: A topological approach to fraud detection. *Int. J. Inf. Technol. Decis. Mak.* **2023**, *22*, 1403–1438. [CrossRef]
24. Deza, M.M.; Deza, E. *Encyclopedia of Distances*; Springer: Berlin/Heidelberg, Germany, 2009.
25. Kamada, T.; Kawai, S. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.* **1989**, *31*, 7–15. [CrossRef]