



## A hybrid single-source shortest path algorithm

Hilal ARSLAN<sup>1</sup>, Murat MANGUOĞLU<sup>2\*</sup>

<sup>1</sup>Roketsan Missiles Inc., Ankara, Turkey

<sup>2</sup>Department of Computer Engineering, Faculty of Engineering, Middle East Technical University, Ankara, Turkey

Received: 03.01.2019

Accepted/Published Online: 05.05.2019

Final Version: 26.07.2019

**Abstract:** The single-source shortest path problem arises in many applications, such as roads, social applications, and computer networks. Finding the shortest path is challenging, especially for graphs that contain a large number of vertices and edges. In this work, we propose a novel hybrid method that first sparsifies a given graph by removing most edges that cannot form the shortest path tree and then applies a classical shortest path algorithm to the sparser graph. Removing all the edges that cannot form the shortest path tree would be expensive since it is equivalent to solving the original problem. Therefore, we propose an iterative bioinspired algorithm, namely the *Physarum* algorithm, as the first stage to sparsify the graph. We prove that the resulting sparser graph always contains the shortest path tree of the original graph. Next, a state-of-the-art algorithm such as Dijkstra's is applied to find the single-source shortest path on the resulting graph. The proposed method is therefore a two-stage hybrid algorithm and it computes the single-source shortest path exactly. We compare the accuracy and solution time of the proposed hybrid method against state-of-the-art implementation of Dijkstra's algorithm and the BFS algorithm on directed weighted and unweighted graphs, respectively, as a baseline. The results show that the proposed hybrid method achieves a significant speed improvement compared to the baseline.

**Key words:** Single-source shortest path, shortest path tree, Dijkstra's algorithm, *Physarum* solver

### 1. Introduction

We consider the single-source shortest path problem in a directed graph with positive edge weights. It is a common problem that arises in many applications such as large road networks [1], wireless networks [2], social networks [3], multicast routing [4], and route information protocol [5]. There are many studies to solve the single-source shortest path problem. Dijkstra's algorithm [6] is one of the most well-known algorithms to solve this problem on graphs with nonnegative edge weights. Let  $G = (V, E)$  be a graph in which  $V$  and  $E$  are the sets of vertices and edges, respectively. Dijkstra's algorithm has  $O(|V|\log|V| + |E|)$  time complexity when a priority queue is utilized. We note that if the graph is unweighted (i.e. the edges have the unit weight), breadth-first search (BFS) is more efficient with a time complexity of  $O(|V| + |E|)$ . The Bellman-Ford algorithm [7] is another well-known single-source shortest path algorithm, which allows negative edge weights and can detect negative cycles in a graph as well. However, it fails to find the shortest path for graphs containing negative cycles. It has  $O(|E||V|)$  time complexity and is costlier than Dijkstra's algorithm, especially for large graphs. Although those algorithms can accurately find the shortest path, they are computationally expensive. Therefore, many bioinspired algorithms have emerged, such as genetic [8], ant colony [9], and *Physarum* solver [10] algorithms, to find the shortest path approximately.

\*Correspondence: [manguoglu@ceng.metu.edu.tr](mailto:manguoglu@ceng.metu.edu.tr)

*Physarum* solver is a bioinspired method for finding the shortest path from a source to a target vertices. The method is developed based on the behavior of an amoeba-like organism, namely the true slime mold *Physarum polycephalum*. Placing food resources at two different locations of a maze, *Physarum polycephalum* absorbs nutrients and creates a path that approximates the minimum length between two resources via its biological structures [11]. *Physarum* solver is a flexible iterative algorithm in the sense that the execution of the algorithm can be terminated whenever the desired accuracy is reached. Furthermore, the paths that cannot connect the source and target vertices are eliminated in a few iterations. Moreover, it gradually eliminates the longer path and approximates the shortest path. Thus, it can detect shorter paths in addition to the shortest one. The mathematical model of *Physarum* was defined in [12, 13]. The *Physarum* model was extended in [14] for solving the shortest path tree problem in directed graphs. Recently, Zhang et al. [15] and Gao et al. [16] accelerated *Physarum* solver in order to solve single-source, single-target shortest path problems efficiently.

*Physarum* solver is used in various other applications. In [17], it is used to solve the Steiner tree problem, which is an NP-hard problem, by using its path-finding ability. *Physarum* solver is also used for identification of critical components in transportation networks [18]. Moreover, it is applied for solving the minimal exposure path problem in wireless sensor networks by converting the problem into the shortest path problem [19] and for routing protocols [20]. In another study [21], it was used for improving the ant colony algorithm. Those authors reported that their proposed method was more efficient and robust than the original ant colony algorithm. For solving the 0-1 knapsack problem, another *Physarum*-related algorithm was proposed in [22]. Furthermore, its adaptivity to solve the shortest path problems on dynamically changing graphs was used in network design in [23]. Another interesting application of *Physarum* solver is the traffic assignment problem [24]. In addition to its applications in various problems, recently we proposed a parallel implementation of *Physarum* solver for solving large-scale shortest path problems in multicore cluster environments [25] since it is inherently distributed and scalable.

In this paper, we use a variant of the *Physarum* solver algorithm in order to detect the edges that cannot form the shortest path tree and we propose a hybrid method, which comprises two stages. In the first stage, we sparsify the given graph by removing the edges that cannot be a part of the shortest path tree using *Physarum* solver. After obtaining a simpler graph, in the second stage, any classical algorithm such as Dijkstra's can be used to find the shortest path (or paths). In this stage, we use the Dijkstra and BFS algorithms for directed weighted and unweighted graphs, respectively. Thus, the search space of the algorithm in the second stage is significantly reduced when compared to the original graph; as a result, the execution time is also significantly reduced. Moreover, the proposed hybrid method finds the shortest path exactly, even though, *Physarum* solver by itself finds the shortest path approximately. This underlines the accuracy of the hybrid method to find the shortest path exactly.

The rest of the paper is organized as follows. In Section 2, we give a description of the *Physarum* model, which leads to the *Physarum* solver algorithm and variants of the *Physarum* solver. In Section 3, we describe the proposed hybrid scheme and prove that it is guaranteed to find the exact shortest path. In Section 4, we present the performance and accuracy using a variety of graphs. Finally, we conclude with some remarks in Section 5.

## 2. *Physarum* model

*Physarum polycephalum* is an amoeboid organism that comprises a dendritic network of tube structures, which are analogous to neurons [26]. An important feature of *Physarum polycephalum* is the adaptation of its shape

to the environment [27] via a self-organizing collective decision-making process [28]. If food resources are placed at two different locations, *Physarum* is able to connect them. In experiments it is shown that the plasmodium of *Physarum polycephalum* approximates the shortest path between two locations in a maze [11]. Next, we give the mathematical model [13] of *Physarum*, which is called the *Physarum* solver. In this model, a maze can be considered as a graph.

Let  $G = (V, E)$  be a connected graph where  $V = \{V_i\}$  and  $E = \{E_{ij}\}$  are the set of vertices and set of edges, respectively. The length of edge  $E_{ij}$ , which is between  $V_i$  and  $V_j$ , is represented by  $L_{ij}$ .  $G$  may be considered as a flow network in which the flow is coming from a source vertex into a target vertex. The variable  $Q_{ij}$  presents the flux through the edge  $E_{ij}$  and is computed by

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}(p_i - p_j), \quad (1)$$

where  $p_i$  is the pressure,  $D_{ij}$  is the conductivity of edge  $E_{ij}$ , and  $L_{ij}$  is the length of the edge. By applying Kirchhoff's law at each node, we have

$$\sum_{(i,j) \in M} Q_{ij} = 0, \quad (j \neq 1, n), \quad (2)$$

where the source node is the first node and the sink node is the last node. For the source and target nodes, the following equations are obtained:

$$\sum_i Q_{i1} + I_0 = 0, \quad (3)$$

$$\sum_i Q_{in} - I_0 = 0, \quad (4)$$

where  $I_0$  is the flux from the source node. The conductivity  $D_{ij}$  changes with time with respect to the flux  $Q_{ij}$ . Eq. (5) is called the adaptation equation and describes the evolution of  $D_{ij}(t)$ :

$$\frac{d}{d\tau} D_{ij} = f(|Q_{ij}|) - D_{ij}, \quad (5)$$

where  $f(|Q_{ij}|) = |Q_{ij}|$  since it guarantees that the method converges to the shortest path irrespective of the initial distribution of conductivities [13].

Poisson's equation, which is the second rule for the flux, is obtained by using Eqs. (1) and (5) to compute the pressure of each vertex in Eq. (6):

$$\sum_i \frac{D_{ij}}{L_{ij}}(p_i - p_j) = \begin{cases} +1 & \text{if } j = 1 \\ -1 & \text{if } j = n \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Eq. (6) can also be written in matrix form:

$$Ap = b, \quad (7)$$

where  $p = (p_0, p_1, p_2, \dots, p_n)^T$  and  $b$  are the unknown and right-hand side vectors, respectively.  $A = (A_{ij})$  given in Eq. (8) is a symmetric M-matrix, which was proved in [12]:

$$A_{ij} = \begin{cases} \sum_{l \neq i} T_{il} & \text{if } i = j \\ -T_{ij} & \text{otherwise,} \end{cases} \quad (8)$$

where

$$T_{ij} = \frac{D_{ij}}{L_{ij}}. \quad (9)$$

Note that  $p_i$  can be computed by just solving Eq. (7). Then each  $Q_{ij}$  is obtained via Eq. (1).

### 2.1. Variants of the *Physarum* model

In this part, we describe variants of the *Physarum* model. The original *Physarum* solver approximates the shortest path from a single source to a single target vertex. The shortest path from a source vertex to a set of target vertices instead of a single target vertex can be given simply by modifying the original model as in [29]. In this generalization,  $DE$  represents the set of target nodes and Eq. (10) describes the flux from a source vertex to the target vertices:

$$\sum_i \frac{D_{ij}}{L_{ij}} (p_i - p_j) = \begin{cases} (n-1) & \text{if } j = s \\ -1 & \text{if } j \in DE \\ 0 & \text{otherwise,} \end{cases} \quad (10)$$

where  $n$  is the cardinality of set  $DE$ .

We recall that the original *Physarum* solver can be applied only on undirected graphs. A generalization to directed graphs was given in [14], which essentially replaces Eq. (6) by the following:

$$\sum_i \left( \frac{D_{ij}}{L_{ij}} + \frac{D_{ji}}{L_{ji}} \right) (p_i - p_j) = \begin{cases} +1 & \text{if } j = s \\ -1 & \text{if } j = t \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

In the proposed hybrid algorithm, the first stage combines the generalization in Eq. (10) into Eq. (11) and the system of equations in Eq. (12) is obtained since we aim to find the shortest path from a source vertex to all other vertices:

$$\sum_i \left( \frac{D_{ij}}{L_{ij}} + \frac{D_{ji}}{L_{ji}} \right) (p_i - p_j) = \begin{cases} (n-1) & \text{if } j = s \\ -1 & \text{if } j \in DE \\ 0 & \text{otherwise.} \end{cases} \quad (12)$$

We note that

$$T_{ij} = \frac{D_{ij}}{L_{ij}} + \frac{D_{ji}}{L_{ji}}, \quad (13)$$

and the matrix in Eq. (8) is still a symmetric M-matrix. Next, the pressure values are obtained by solving the linear system in Eq. (12). This essentially removes the directions of the edges when computing the pressure values, which is an approximation. In the next step, the flux and conductivity of the edges are updated using Eqs. (1) and (5), respectively, which still take directions into account.

### 3. Proposed hybrid method

In this section, a new hybrid algorithm is proposed in order to solve the single-source shortest path problem in directed graphs with positive edge weights based on the *Physarum* solver and a single-source shortest path algorithm, namely Dijkstra's algorithm for weighted graphs and the BFS for unweighted graphs.

Dijkstra's algorithm computes the exact single-source shortest path on graphs that have nonnegative edge weights. However, when the number of the vertices and edges is larger, it requires excessive computational time to find the shortest path even if those edges may not be a part of the shortest path. For instance, a graph may include many cycles and at least one edge on a cycle cannot be a part of the shortest path. Additionally, if the graph is complete, at least half of the edges cannot be a part of the single-source shortest path tree. If those edges are removed from the graph, the computational time of Dijkstra's algorithm could be improved significantly without compromising its accuracy.

In our proposed algorithm, after the edges that cannot be a part of the single-source shortest path are determined by using the *Physarum* algorithm, a sparser graph is formed. Next, Dijkstra's algorithm and the BFS are applied for weighted graphs and unweighted graphs, respectively, in order to find the single-source shortest path on the sparser graph. This leads to a significant reduction in the computational time of Dijkstra's algorithm (or the BFS).

Now we prove a crucial theorem showing that *Physarum* solver eliminating edges whose fluxes are less than zero does not disturb the shortest path tree. First we need to prove a few lemmas, leading to the proof of our theorem.

**Lemma 1** *Any edge  $L_{ij}$  with negative flux  $Q_{ij}$  cannot be one of the edges forming the shortest path.*

**Proof** The mathematical model of *Physarum* [10] based on Poiseuille flow and Krichooff's laws is represented by an electrical circuit system [30]. In this representation, the electrical current passing from node  $i$  to node  $j$  is equal to the flux on edge  $L_{ij}$ . The potential difference between node  $i$  and node  $j$  corresponds to the difference of the pressures ( $p_i - p_j$ ). Therefore, a mapping from the *Physarum* network to an electrical circuit is constructed by Eqs. (14), (15), and (16):

$$I_{ij} = Q_{ij}, \quad (14)$$

$$V_{ij} = p_i - p_j, \quad (15)$$

$$R_{ij} = \frac{L_{ij}}{D_{ij}}, \quad (16)$$

where  $I_{ij}$  is the electrical current,  $V_{ij}$  is the potential, and  $R_{ij}$  is the resistance between vertex  $i$  and vertex  $j$ . In other words,

$$I_{ij} = \frac{V_{ij}}{R_{ij}} = \frac{D_{ij}}{L_{ij}}(p_i - p_j) = Q_{ij}. \quad (17)$$

Therefore, Eq. (17) is a model of the flow thorough the tubes in the *Physarum* network (see Eq. (1)). We note that the flow direction is consistent with the edge orientation. If the potential difference between node  $i$  and node  $j$  is positive, the electrical current flows from node  $i$  to node  $j$  by Ohm's law. Now we apply this information in the *Physarum* network. If the flux  $Q_{ij}$  of an edge  $L_{ij}$  is negative, it means that the flow moves from node  $j$  to node  $i$ , which is the opposite of edge orientation. Thus, this edge cannot be on the shortest

path. In conclusion, the edge with negative flux cannot be one of the edges forming the shortest path from  $s$  to  $t$  since the orientation of the edge is not consistent with the flow direction.  $\square$

**Lemma 2** *Let  $G$  be a graph with positive edges. There is at least one edge on a cycle in  $G$  whose flux value is negative.*

**Proof**

Let  $C$  be a cycle in  $G$  such that

$$C = e_{ab}e_{bc}e_{cd}\dots e_{za},$$

where  $e_{ij}$  represents the edge connecting  $v_i$  and  $v_j$ . We assume that the fluxes with respect to these edges on  $C$  are positive. By using Eq. (1) for edge  $e_{ab}$ ,

$$Q_{ab} = \frac{D_{ab}}{L_{ab}}(p_a - p_b) > 0,$$

and hence  $p_a > p_b$ . If all fluxes on  $C$  are positive, then we obtain that

$$p_a > p_b > p_c > p_d > \dots > p_z > p_a.$$

This is a contradiction. As a result, there has to be at least one edge on the cycle whose flux value is negative.  $\square$

**Theorem 1** *Let  $G$  be a connected graph. If we remove the edges whose flux values are less than zero and call the resulting graph  $G_s$ , then  $G_s$  is a tree and the shortest path tree of the original graph  $G$  is a subtree of  $G_s$ .*

**Proof** First, we prove that  $G_s$  is a tree. In order to prove this, we will show that  $G_s$  does not contain any cycle and is connected. First, we will show that  $G_s$  does not contain any cycle. If all edges whose flux values are negative are removed from  $G$  and in a cycle there is at least one edge with negative flux by Lemma 2, at least one edge on a cycle is removed. Thus, the cycle is removed. Second, by Lemma 1, edges with negative fluxes cannot be on the shortest path. Therefore, removing such edges does not affect the connectivity of the graph since we only remove the edges that cannot be on the shortest path. As a result,  $G_s$  is a tree. Finally, since  $G_s$  includes all edges that may form the shortest path tree, the shortest path tree is a subtree of  $G_s$ .  $\square$

Now we give the pseudocode of the hybrid algorithm in Algorithm 1. The input of the algorithm is a graph  $G = (V, E)$ , where  $V$  and  $E$  are the sets of vertices and edges, respectively.  $L$  is the corresponding adjacency matrix. In Line 3, the diagonal entries of  $L$  (i.e. self loops) are removed directly since the shortest path cannot include the same node. Then we apply the *Physarum* algorithm (Lines 8–13) in order to find the flux values ( $Q$  matrix). This is *Stage1* of hybrid algorithm. In the *Physarum* model, Eq. (18), which is derived from Eq. (10) and Eq. (11), is used since our goal is to compute the shortest path from a source vertex to all other vertices. In our algorithm, the *while* loop is iterated only once in order to determine the direction of the flux. For determining more edges not forming the single-source shortest path, the number of the *while* loop iterations may be increased. The algorithm requires the solution of a sparse linear system of equations (Line 10). We use the same iterative linear system solver proposed in [25]. It is sufficient to solve the linear system with moderate accuracy and hence we stop the iterations when the residual norm relative to the norm of the

---

**Algorithm 1** Hybrid shortest path algorithm.

---

1: **Data:**  $G = (V, E)$  where  $V$  and  $E$  are the sets of vertices and edges, respectively, of the graph (which is represented as an adjacency matrix  $L$ ),  $s$  is the source vertex, and  $n$  is the number of vertices.  
2: **Result:** the single-source shortest path  
3:  $L^s = L - \text{diag}(L)$   
4: //Physarum Algorithm  
5:  $D \leftarrow (0, 1]$   
6:  $Q \leftarrow 0$   
7:  $p \leftarrow 0$   
8: **while** a termination condition is not met **do**  
9:      $p_t \leftarrow 0$   
10:     Computing the pressures by Eq. (18)

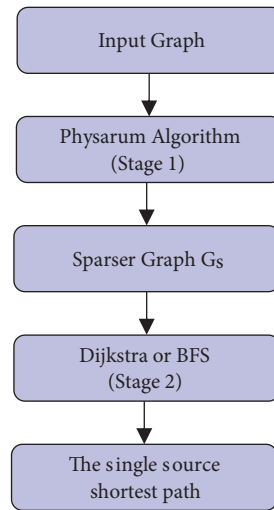
$$\sum_i \left( \frac{D_{ij}}{L_{ij}^s} + \frac{D_{ji}}{L_{ji}^s} \right) (p_i - p_j) = \begin{cases} n-1 & \text{if } j = s \\ -1 & \text{otherwise} \end{cases} \quad (18)$$

11:      $Q_{ij} \leftarrow \frac{D_{ij}}{L_{ij}^s} \times (p_i - p_j)$   
12:      $D \leftarrow (Q + D)/2$   
13: **end while**  
14: //Removing edges not forming the shortest path  
15: **if**  $Q_{ij} < \epsilon$  **then**  
16:      $L_{ij}^s = 0$   
17: **end if**  
18: //Dijkstra's algorithm (or BFS) using  $G_s$   
19: [dist,path]=**graphshortestpath**( $G_s, s$ ) where  $G_s = (V, E_s)$  represented by  $L^s$

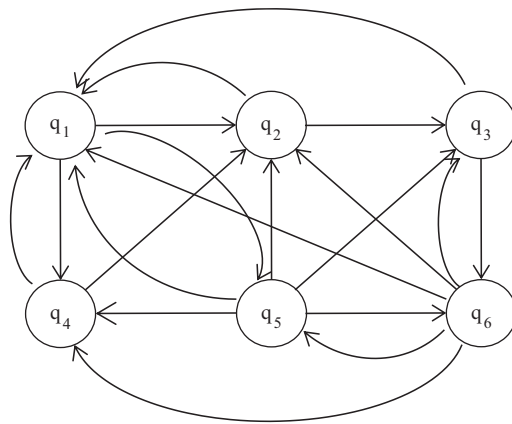
---

right-hand-side vector is less than or equal to  $10^{-3}$ . After the flux values are computed, the edges whose flux values are less than  $\epsilon$  (machine precision) are excluded (Lines 15–17) since they are practically zero. Therefore, the resulting graph  $G_s$  is sparser, and by Theorem 1,  $G_s$  includes all edges that form the shortest path tree. Therefore, finding the single-source shortest path on  $G$  is equivalent to finding the single-source shortest path on  $G_s$ . Finally, Dijkstra's algorithm (or the BFS) efficiently computes the shortest path using the simple and sparse input graph. We call the final stage *Stage2* of the hybrid algorithm. To summarize, we present the flow of the algorithm in Figure 1.

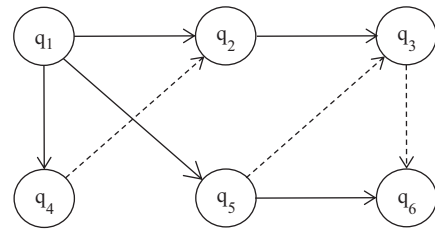
To illustrate how the proposed method works, we give a small example. The example graph  $G$  is given in Figure 2a. There are 6 vertices ( $q_1, q_2, q_3, q_4, q_5$ , and  $q_6$ ) and 19 edges in this graph. We assume that  $q_1$  is the source vertex and all edge weights are 1. In order to remove unnecessary edges, Stage 1 of the proposed method is applied. Therefore, after the edges that cannot form the the shortest path tree are removed by following the steps in Algorithm 1, the resulting sparse graph,  $G_s$ , is obtained. In Figure 2b,  $G_s$  is shown by all edges with solid and dashed lines and the shortest path tree is shown by only edges with solid lines. That is, the dashed edges belong to  $G_s$ , but are not in the shortest path tree. To sum up, there are eight edges in  $G_s$  and only three of them are not included in the shortest path tree. Finally, the single-source shortest path is computed by BFS on the  $G_s$  graph. If we choose the vertex  $q_6$  as a target vertex, the shortest path from  $q_1$  to  $q_6$  is computed as  $s = q_1 - q_5 - q_6 = t$ . In the next section, we show the performance of the proposed method on larger graphs.



**Figure 1.** Flow of the hybrid algorithm.



(a) An example directed graph  $G$ .



(b) The resulting graph  $G_s$  (solid and dashed edges) where the solid edges indicate the shortest path tree.

**Figure 2.** Illustration of the proposed hybrid method on an example graph  $G$ .

#### 4. Results

In this section, we present the results of numerical experiments. In order to demonstrate the efficiency of the proposed algorithm, as a baseline method, we use the Dijkstra and BFS implementations of the *graphshortestpath* function available in MATLAB. The proposed hybrid algorithm is also implemented in MATLAB. All runs are performed on a computer with 2 x Intel Xeon E5-2650v3 (2.30 GHz) processors and 64 GB RAM on a CentOS operating system. The number of vertices, edges, and types of the graphs are given in Table 1. The first four graphs are generated by using the Erdos–Renyi model (*erdos.renyi.game* function [31] of igraph library in R language). The last four graphs, which are R-MAT graph models, are generated by using the PaRMAT library [32]. Diverse real graphs can be well approximated by these models [33]. In the dataset, the number of vertices varies from 3500 to 10000 and the number of edges varies from 6M to 50M. We use random and unit weights for the Erdos–Renyi and R-MAT graphs, respectively.



**Table 1.** Graph properties.

Graph name	Vertices	Edges	Kind
Rmat1	3500	6M	R-MAT
Rmat2	4000	7M	R-MAT
Rmat3	5000	12M	R-MAT
Rmat4	6000	18M	R-MAT
Erdos1	7000	24M	Erdos–Renyi
Erdos2	8000	32M	Erdos–Renyi
Erdos3	9000	40M	Erdos–Renyi
Erdos4	10000	50M	Erdos–Renyi

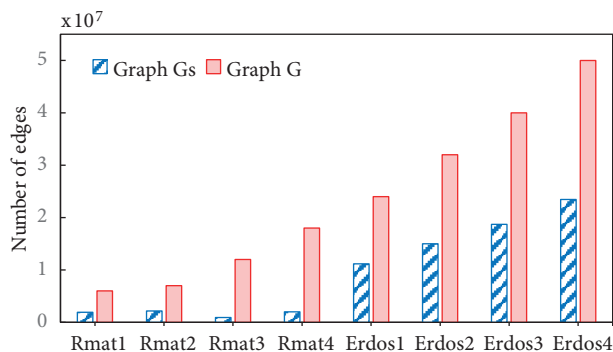
We evaluate the algorithms based on required time to solution as well as accuracy. In the hybrid algorithm, first, a sparser graph that does not contain the edges that cannot form the shortest path tree is formed. Figure 3 presents the number of edges of  $G$  (i.e. the original graph) and  $G_s$  (i.e. excluding the edges that cannot form the shortest path) for each graph. As shown in Figure 3, there are a large number of edges that cannot form the shortest path tree and the proposed algorithm identifies and removes such edges efficiently. Second, after removing a large number of unused edges in the shortest path tree, Dijkstra’s algorithm or the BFS is used to compute the single-source shortest path by the *graphshortestpath()* function in MATLAB. Therefore, Dijkstra’s algorithm or the BFS efficiently computes the single-source shortest path.

Next, we look into the required time for the proposed and baseline algorithms. The proposed hybrid algorithm consists of two stages. The required solution time of each stage as well as the required time for the baseline (Dijkstra or BFS algorithms) are shown in Table 2, in seconds. The algorithm detects the edges that are not a part of the single-source shortest path relatively quickly in *Stage1* and the required time for *Stage1* is given in Table 2. *Stage2* employs Dijkstra’s algorithm for weighted graphs (or the BFS for unweighted graphs) on a sparser graph. We note that the baseline is applied on the original graph while the same baseline is also applied on the sparser graph obtained after *Stage1*. We observe that on average *Stage2* is 2.6 times faster than the baseline algorithm. The reason for this is that the search space in *Stage2* is significantly reduced when compared to the search space in the baseline algorithm. We also note that for graph Rmat3, the number of the edges removed from the graph in Figure 3 is larger and therefore the required time for *Stage2* is significantly reduced for this graph. Now we look into the results in terms of the required total solution time. Total solution time of the hybrid algorithm is the sum of two stages, namely *Stage1* and *Stage2*. Figure 4 presents the total solution time of the Dijkstra/BFS and the hybrid algorithm for each test graph separately. As shown in Figure 4, the required time for the hybrid algorithm is significantly less than that for Dijkstra’s algorithm/BFS for all test graphs. In fact, the proposed hybrid algorithm is about 1.9 times faster than Dijkstra’s algorithm on average. Removing the edges that cannot form the shortest path tree saves a significant amount of time. When the size of the graph increases, the required time also increases, as expected.

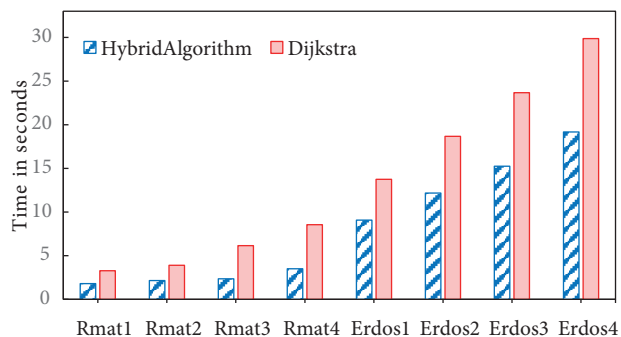
Now we look into the results in terms of accuracy. We compute the shortest path from the source vertex to the other vertices. The hybrid algorithm and the Dijkstra/BFS algorithm compute exactly the same shortest path with the same distance. On the other hand, if one had used the *Physarum* algorithm alone to find the shortest path, it would be only an approximation [34].

**Table 2.** Time (in seconds, rounded to two decimal places) for hybrid and Dijkstra (or BFS) algorithms. Hybrid algorithm consists of two stages and the required time for each stage is also shown.

Graph name	Hybrid algorithm			Baseline algorithm
	Stage1	Stage2	Total	Dijkstra/BFS
Rmat1	0.94	0.87	1.81	3.27
Rmat2	1.10	1.04	2.14	3.90
Rmat3	1.97	0.39	2.36	6.16
Rmat4	2.61	0.88	3.49	8.55
Erdos1	2.88	6.19	9.07	13.74
Erdos2	3.83	8.35	12.18	18.67
Erdos3	4.77	10.48	15.25	23.68
Erdos4	5.99	13.18	19.17	29.87



**Figure 3.** Total number of edges for the adjacency matrices corresponding to the original ( $G$ ) and sparsified ( $G_s$ ) graphs.



**Figure 4.** Time in seconds of Dijkstra (or BFS) and hybrid algorithms for each test graph.

## 5. Conclusions

Finding the shortest path from a single-source vertex to all other vertices is an important problem with many applications requiring significant computation power, especially for large problems. However, there could be many edges in the graph that cannot possibly be on the shortest path. Such edges drastically increase the solution time of the single-source shortest path algorithms. Motivated by this, we propose a two-stage hybrid method. In the first stage, a large number of edges that are not a part of the single-source shortest path tree are removed by using the *Physarum* algorithm, which is inherently distributed and scalable. We claim and prove that the graph after removing all of those edges is a tree and it contains the shortest path tree of the original graph. Next, in the second stage, either Dijkstra's algorithm or the BFS is applied to compute the single-source shortest path using the simple and sparser tree obtained in the earlier stage. Thus, Dijkstra's algorithm (or the BFS) efficiently computes the single-source shortest path since the search space of the algorithm is significantly reduced by *Physarum*. We compare our results against the state-of-the-art implementation of Dijkstra's algorithm for weighted and the BFS for unweighted graphs as a baseline using a number of test problems generated based on Erdos-Renyi and RMAT models. Experimental results show that the hybrid method is about two times faster than the baseline on average. As future work, we also note that the Dijkstra and BFS algorithms in the second stage of the proposed hybrid method can be replaced with any other available shortest path algorithm.

## References

- [1] Ertl G. Shortest path calculation in large road networks. *Operations Research Spektrum* 1998; 20 (1): 15-20.
- [2] Nguyen UT, Xu J. Multicast routing in wireless mesh networks: minimum cost trees or shortest path trees? *IEEE Communications Magazine* 2007; 45 (11): 72-77.
- [3] Gong M, Li G, Wang Z, Ma L, Tian D. An efficient shortest path approach for social networks based on community structure. *CAAI Transactions on Intelligence Technology* 2016; 1 (1): 114-123.
- [4] Bauer F, Varma A. Distributed algorithms for multicast path setup in data networks. In: *Proceedings of GLOBECOM '95*; Singapore; 1995. pp. 1374-1378.
- [5] Rescigno AA. Optimally balanced spanning tree of the star network. *IEEE Transactions on Computers* 2001; 50: 88-91.
- [6] Dijkstra EW. A note on two problems in connexion with graphs. *Numerische Mathematics* 1959; 1: 269-271.
- [7] Bellman R. On a routing problem. *Quarterly of Applied Mathematics* 1958; 16: 87-90.
- [8] Ahn CW, Ramakrishna RS. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation* 2002; 6: 566-579.
- [9] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1997; 1 (1): 53-66.
- [10] Tero A, Kobayashi R, Nakagaki T. Physarum solver: a biologically inspired method of road-network navigation. *Physica A Statistical Mechanics and Its Applications* 2006; 363: 115-119.
- [11] Nakagaki T, Yamada T, Toth A. Maze-solving by an amoeboid organism. *Nature* 2000; 407: 470.
- [12] Miyaji T, Ohnishi I. Physarum can solve the shortest path problem on Riemann surface mathematically rigorously. *International Journal of Pure and Applied Mathematics* 2008; 47: 353-369.
- [13] Tero A, Kobayashi R, Nakagaki T. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of Theoretical Biology* 2007; 244(4): 553-564.
- [14] Wang Q, Lu X, Zhang X, Deng Y, Xiao C. An anticipation mechanism for the shortest path problem based on Physarum polycephalum. *International Journal of General Systems* 2015; 44(3): 326-340.
- [15] Zhang X, Zhang Y, Zhang Z, Mahadevan S. Rapid Physarum algorithm for shortest path problem. *Applied Soft Computing* 2014; 23: 19-26.
- [16] Gao C, Zhang X, Yue Z, Wei D. An accelerated Physarum solver for network optimization. *IEEE Transactions on Cybernetics* (in press).
- [17] Liu L, Song Y, Zhang H, Ma H. Physarum optimization: a biology-inspired algorithm for the Steiner tree problem in networks. *IEEE Transactions on Computers* 2015; 64: 818-831.
- [18] Zhang X, Adamatzky A, Yang H, Mahadaven S, Yang XS et al. A bio-inspired algorithm for identification of critical components in the transportation networks. *Applied Mathematics and Computation* 2014; 248: 18-27.
- [19] Liu L, Song Y, Ma H, Zhang X. Physarum optimization: a biology-inspired algorithm for minimal exposure path problem in wireless sensor networks In: *2012 Proceedings In: IEEE INFOCOM*; Orlando, FL, USA; 2012. pp. 1296-1304.
- [20] Li K, Torres CE, Thomas K, Rossi LF, Shen CC. Slime mold inspired routing protocols for wireless sensor networks. *Swarm Intelligence* 2011; 5: 183-223.
- [21] Zhang Z, Gao C, Liu Y, Qian T. A universal optimization strategy for ant colony optimization algorithms based on the Physarum-inspired mathematical model. *Bioinspiration and Biomimetics* 2014; 9 (3): 036006.
- [22] Zhang X, Huang S, Hu Y, Zhang Y, Mahadevan S et al. Solving 0-1 knapsack problems based on amoeboid organism algorithm. *Applied Mathematics and Computation* 2013; 219: 9959-9970.

- [23] Zhang X, Chan FT, Yang H, Deng Y. An adaptive amoeba algorithm for shortest path tree computation in dynamic graphs. *Information Sciences* 2017; 405: 123-140.
- [24] Xu S, Jiang W, Deng X, Shou Y. A modified Physarum-inspired model for the user equilibrium traffic assignment problem. *Applied Mathematical Modelling* 2018; 55: 340-353.
- [25] Arslan H, Manguoglu M. A parallel bio-inspired shortest path algorithm; *Computing* 2019; 101: 969-988. doi: 10.1007/s00607-018-0621-x
- [26] Adamatzky A. A would-be nervous system made from a slime mold. *Artificial Life* 2015; 21 (1): 73-91.
- [27] Jones J. Characteristics of pattern formation and evolution in approximations of Physarum transport networks. *Artificial Life* 2010; 16 (2): 127-153.
- [28] Meyer B. Optimal information transfer and stochastic resonance in collective decision making. *Swarm Intelligence* 2017; 11 (2): 131-154.
- [29] Liang M, Gao C, Zhang Z. A new genetic algorithm based on modified Physarum network model for bandwidth-delay constrained least-cost multicast routing. *Natural Computing* 2017; 16 (1): 85-98.
- [30] Zhang X, Zhang Y, Deng Y. An improved bio-inspired algorithm for the directed shortest path problem. *Bioinspiration and Biomimetics* 2014; 9 (4): 046016.
- [31] Erdos P, Renyi A. On the evaluation of random graphs. *Mathematical Institute of the Hungarian Academy of Sciences* 1960; 5 (7): 17-61.
- [32] Khorasani F, Gupta R, Bhuyan LN. Scalable SIMD-efficient graph processing on GPUs. In: *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques*; Limassol, Cyprus; 2018. p 39-50.
- [33] Chakrabarti D, Zhan Y, Faloutsos C. R-MAT: A recursive model for graph mining. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*; Florida, USA; 2004. p. 442-446.
- [34] Becchetti L, Bonifaci V, Dirnberger M, Karrenbauer A, Mehlhorn K. Physarum can compute shortest paths: convergence proofs and complexity bounds. In: *Automata Languages and Programming: 40th International Colloquium*; Riga, Latvia; 2013. p. 472-483.