CHALMERS | GÖTEBORG UNIVERSITY

Technical Report No. 2012-08

Physarum-inspired self-biased walkers for distributed clustering

DEVAN SOHIER
GIORGOS GEORGIADIS
SIMON CLAVIERE
MARINA PAPATRIANTAFILOU
ALAIN BUI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY/
GÖTEBORG UNIVERSITY
Göteborg, Sweden, 2012

Physarum-inspired self-biased walkers for distributed clustering

Devan Sohier[2], Giorgos Georgiadis[1], Simon Clavière[2], Marina Papatriantafilou[1] and Alain Bui[2]

[1]Department of Computer Science and Engineering, Chalmers University of Technology, S-412 96 Göteborg, Sweden, Email: {georgiog,ptrianta}@chalmers.se, Fax: +46-31-7723663
[2]Laboratoire PRiSM (UMR CNRS 8144), Université de Versailles St-Quentin-en-Yvelines, 78035 Versailles, France, Email: {devan.sohier,simon.claviere,alain.bui}@prism.uvsq.fr, Fax: +33-1-39254057

Abstract

In this article, we propose a distributed scheme to compute distance-based clusters. We first present a mechanism based on the flow of distributed tokens called walkers, circulating randomly between a source and a sink to compute a shortest path. This mechanism is a discrete emulation of the slime mould (*Physarum polycephalum*) dynamics presented in [16]: each node observes the flow of walkers going through each adjacent edge and uses this flow to compute the probabilities with which it sends the walkers through each edge. Then, based on this mechanism, we show how several sources compute a shortest path DAG to a given sink. Finally, given some clusterheads acting like sinks, we prove that this process converges to distance-based clusters (i.e. nodes join the clusterhead to which they are closest) with shortest-path DAGs. The algorithm is designed with a special focus on dynamic networks: the flow locally adapts to the appearance and disappearance of links and nodes, including clusterheads.

1 Introduction

We focus on overlay construction over networks, in a way that nodes are clustered around specified clusterheads. Clustering can facilitate locality and scalability properties in a variety of networks – wireless, ad-hoc, sensor networks – for services such as routing, data aggregation, resource finding and sharing in neighborhoods. The latter is instrumental for realizing interest-based groups and facilitating grouping based on gradient proximity measures [27]. When it comes to resource sharing, load balancing is aligned with clustering; it forms an incentive to achieve and a way to enforce fairness and group participation in collaboration.

In particular the overlay we are aiming for here should define for each node efficient ways of reaching the closest clusterhead through shortest paths (using an application related distance metric); moreover, nodes at equal distances from several clusterheads should connect to all of them, thus also forming bridges among clusters. We are interested in methods that do not need global knowledge of the network and have strong locality and self-organizing properties and small, lightweight messages [21].

Towards this goal one of the basic instruments will be based on an adaptation of *random walks* we call self-biased walkers. Random walks have been shown to be effective in the searching, dissemination and construction of overlay networks as a statistical process that associates with important parameters and properties of the corresponding network [18]. We will show that the proposed walkers adaptively develop bias towards corresponding shortest paths and can effectively simulate a process of natural computation, in particular the slime mould *Physarum polycephalum*. We will further show how to use this basic form of computation towards achieving the aforementioned goal of this work.

In their seminal work, Nakagaki et al [25] found that they could use the slime mould *Physarum polycephalum* in order to solve simple mazes, by placing food at the exits, introducing the mould into the maze and letting it reach the food using extensible tube-like protuberances (pseudopodia). The mould

not only solved the maze but did so efficiently, by forming shortest paths between the entry and the exit points of the maze. In doing so, the mould changes the diameter of its pseudopodia, depending on whether the particular protuberance is vital in the nutrient circulation. Based on these works and with further experimentations, researchers succeeded in reproducing the transportation networks of the Tokyo metropolitan area [26] and countries such as Brazil [15] by placing food in population centers and having the mould connect them.

The mechanism through which the mould achieves the quality of its solutions remains an open research question. Several authors [25, 23, 24, 26] suggested an electric circuit model, where nodes are connected through resistors with a diameter property that can affect the amount of current that passes through them. This diameter corresponds roughly to the pseudopodia diameter of the mould and tends to follow the electric current flow: if the flow becomes greater the diameter increases, otherwise it decreases. For the exact relation between diameter D_{ij} and current Q_{ij} passing through a resistor (i, j) though, several mechanisms have been proposed, for example

$$[23] \quad \frac{dD_{ij}}{dt} = f\left(|Q_{ij}|\right) - D_{ij} \quad (1) \qquad [24] \quad \frac{dD_{ij}}{dt} = |Q_{ij}| - D_{ij} \quad (3)$$

$$[25] \quad \frac{dD_{ij}}{dt} = |Q_{ij}|^{\mu} - aD_{ij} \quad (2) \qquad [26] \quad \frac{dD_{ij}}{dt} = \frac{|Q_{ij}|^{\gamma}}{1 + |Q_{ij}|^{\gamma}} - D_{ij} \quad (4)$$

which lead to different dynamics. It has been conjectured that the usage of these dynamics can lead to efficient algorithms in applications that involve shortest paths and similar transportation problems. To our knowledge, the distributed algorithm presented here is the first such algorithm.

Related work on Physarum In the Physarum-related literature, a number of different dynamics have been explored to explain the mould's behavior, focusing on the different forms of function $f(\cdot)$ of equation 1 [26, 19, 23, 16]. Specifically Ito et al [19] use the equation 3 for the diameter dynamics and prove that their model is able to solve transportation problems, while the diameters converge exponentially to their final values. Miyaji and Onishi [24] use equation 3 and prove analytically that the dynamics converge for planar graphs with a unique shortest path between source and sink nodes, to a flow that uses only the shortest path. Bonifaci et al [16] move one step further by proving that dynamics based on equation 3 similarly converge to the unique shortest path for all graphs, provided a unique short path between source and sink exists. Furthermore, they show that in the generic case of multiple sources and sinks, the dynamics are attracted to a set of equilibria that are candidate solutions for the transportation problem. Recently the Physarum dynamics have been proposed to address common computer science problems such as the work of Johannson and Zou [20] on Linear Programming problems and Li et al [22] on routing protocols for sensor networks: the first proposes a method to encode any linear program into the physarum dynamics and solve it efficiently, while the second adapts two Physarum mechanisms, path growth and path evolution, to efficiently route messages in a wireless sensor network.

Related work on clustering A number of methods have been proposed in the literature to construct clusters in unstructured networks. For example, the algorithms of [2, 4, 8, 9] produce clusters with radius one. Each cluster has a node called a *clusterhead* and all other nodes in that cluster are neighbors of the clusterhead. Clusters can be built using a hierarchical method: the clustering algorithm is iterated on the overlay network obtained by considering clusters as nodes, until a single cluster is obtained [7, 10, 12]. On the other hand, the solution given in [1] builds k-hop clusters (clusters of radius k) and in [6], a self-stabilizing[1] $O(n)$-time algorithm is given for computing a minimal k-dominating set; this set can then be used as the set of clusterheads for a k-clustering. Other clustering algorithms are based on random walks [3, 5]. In [5] the clusters are built around bounded-size connected dominating sets, while

[1] A self-stabilizing system is a system that eventually recovers a normal behavior after a transient fault.

in [3] the algorithm recursively breaks the network into two clusters as long as every cluster size satisfies a lower bound.

These solutions are bottom-up processes while we propose a top-down approach [11] by relying on predefined clusterheads. We then compute clusters consisting of the nodes that are closer to a given clusterhead than to any other. Our algorithm has a focus on topological changes: the use of a flow-based process allows the system to reconfigure after a topological change without affecting nodes of other clusters. In particular, a topological change can affect the cluster in which it occurs and possibly neighboring clusters, but not further nodes.

To our knowledge, the present paper is the first work that uses the Physarum dynamics together with a discrete probabilistic tool such as self-biased walks in order to solve a distributed problem, namely node clustering.

Outline In the algorithm presented here we show how adaptive self-biased walkers can emulate the dynamics of the Physarum. In section 2 we introduce the necessary notation and definitions. We then proceed by showing the algorithm for the walkers and furthermore how to use the method in order to enable each node in a network, in a distributed way, to determine the direction towards its closest clusterhead (using a distance metric noted by L in the following) (section 3). In section 4 we study experimentally the behavior of the algorithm in terms of resulting cluster computation, reaction to changes in the graph and cost. We complete our presentation with our conclusions and future work in section 5.

2 Definitions and notation

We consider a distributed system, consisting of nodes with computation capabilities and communication edges between them, in the form of an undirected graph $G = (V, E)$, $|V| = n, |E| = m$. Nodes have no id and the system is anonymous. We treat synchronous systems and c-asynchronous, i.e. i: in a round, all processors receive all pending messages and treat them and ii: in a round for a given node, no node has executed more than c rounds. Each edge (i, j) has also a length $L_{ij} > 0$ that we assume is known to both endnodes and a time-varying diameter $D_{ij}(t) > 0$.

A discrete time biased walk on a graph with a bias function C (here, the conductance) is a process by which a node is chosen at each step according to the following rule: if node i holds the walker at step t, then any neighbor j of i will hold it at step $t + 1$ with probability $P_{ij} = \frac{C_{ij}}{\sum_{k \in \Gamma_i} C_{ik}}$, where Γ_i is the set of immediate neighbors of node i. Such a process can easily be implemented in a distributed system.

In their seminal work, Doyle and Snell [17] connected random walks on weighted undirected graphs and electric circuits, by showing that a random walker behaves in a way similar to the simplistic view of electrical current as movement of electrons, provided every edge (i, j) is treated as a resistor with *conductance* $C_{ij}(t)$[2] equal to the edge's weight. In fact they showed that, when a unit current is injected into the graph, the *electrical current* $Q_{ij}(t)$ flowing over a resistor (i, j) corresponds to the mean number of walker crossings over the respective edge. Recent work showed that it is possible to use this paradigm in order to form and study overlay networks in an efficient way [28, 29]. These results imply that the electrical current flow can be emulated by counting the mean number of walker crossings of an edge and be used to form a clustering overlay.

The connection between the physarum dynamics and electrical networks, as well as between electrical networks and random walks, is the basis for the implementation presented in this paper.

In the rest of the paper we define the conductance $C_{ij}(t)$ of an edge (i, j) to be equal to $C_{ij}(t) = \frac{D_{ij}(t)}{L_{ij}}$. The transition probability of the biased walker from node i to node j now becomes $P_{ij}(t) = \frac{C_{ij}(t)}{\sum_{k \in \Gamma_i} C_{ik}(t)}$.

[2]Note that all electrical measures are necessarily time-varying in order to simulate the Physarum dynamics.

The mould dynamics are defined by the evolution of the edge diameters of the network and here we follow the definition of Bonifaci et al [16] in its discrete form:

$$\begin{aligned} D_{ij}(t+1) = &\quad D_{ij}(t) + \varepsilon(|Q_{ij}(t)| - D_{ij}(t)) \\ = &\quad (1-\varepsilon)D_{ij}(t) + \varepsilon|Q_{ij}(t)| \end{aligned} \tag{5}$$

3 Distributed emulation of Physarum dynamics: PECAN algorithm

The PECAN (*Physarum walkErs Clustering*) algorithm we present here is based on monitoring the flow of walkers on each edge and then adjusting probabilities accordingly. This flow is used as an estimation of the electrical current flowing on edges, and the diameter D is updated accordingly.

The algorithm assumes there is a unit flow of walkers that are created at nodes called *sources* and destroyed at nodes called *sinks*. At each round, every node counts the number of walkers going in and out each of its adjacent edges and updates its flow information. Instant flow, meaning the flow measured during a round, may vary abruptly from round to round, due to the discrete stochastic nature of the system. To avoid sharp changes, we take into account past observations. However, to allow the system to evolve with newer observations, we measure the flow with a discount factor applied to past observations. In practice the flow estimation Q is computed as

$$Q_{ij}\,(t+1) = \alpha Q_{ij}\,(t) + (1-\alpha)\#w_{ij}(t) \tag{6}$$

where $\#w_{ij}(t)$ is the observed flow, i.e. the number of walkers that have crossed (i,j) in this direction at round t minus the number that have crossed in the opposite direction, and $\alpha \in [0,1]$ a parameter. This formula is such that if $\#w_{ij}$ eventually reaches a probability distribution with an average, Q_{ij} stabilizes to this average.

Equation 5 expresses the fact that $D_{ij}(t+1)$ is a weighted average of $D_{ij}(t)$ and $|Q_{ij}(t)|$, and equation 6 that $Q_{ij}(t+1)$ itself is a weighted average of $Q_{ij}(t)$ and $\#w_{ij}(t)$. Asymptotically, the sign of Q should stabilize or Q tends to 0. Thus, asymptotically, $D_{ij}(t+1)$ will be a weighted average of $D_{ij}(t)$ and $\#w_{ij}(t)$. By taking $\alpha' = (1-\varepsilon)\alpha$, the diameter D_{ij} then follows asymptotically equation 6. In the following we take $D_{ij}(t+1) = |Q_{ij}(t)|$ and the transition probabilities are then:

$$P_{ij}\,(t+1) = \frac{|Q_{ij}\,(t)|\,/L_{ij}}{\sum\limits_{k\in\Gamma_i} |Q_{ik}\,(t)|\,/L_{ik}} \tag{7}$$

As we show in the following lemma, the dynamics converge to the same point as the one of Bonifaci et al [16] in the case of single source and single sink, namely by setting the diameter of edges on shortest paths equal to 1 (as well as 0 to all the others).

Lemma 1 *If there is a unique shortest path from source to sink and the dynamics stabilize, the diameter of edges on the shortest path converge to 1 while those not on the shortest path to 0.* [3]

The α parameter has a strong influence on the behavior of the system: a conservative α (close to 1) slows down the convergence, while a small α may make the system converge to a suboptimal solution (which a slight modification of the system dynamics can solve), or entail sharp changes in the values of Q_{ij} even after the system seems to have converged, thus making the solution *"unstable"*.

Any path between the source and the sink is a fixed point of these dynamics (even if only shortest paths are attractive): if Q_{ij} is set to 1 on all edges of a non-shortest path from the source to the sink and to 0 everywhere else, then the system does not evolve anymore. However, such a solution is an *"unstable equilibrium"*, as shown by [16] in the continuous case. To solve these situations, we add a β parameter,

[3] Please find the missing lemma proof in the Appendix.

the role of which is to prevent the system from converging fully. The β parameter is a lower bound on the probability that a node sends the walker across an adjacent edge. Thus, if close to a non-optimal (repulsive) solution, the system will be driven away from it. If close to an optimal (attractive) solution, the system will tend to get closer to it, even if β hinders this progression.

The β parameter also allows to initialize the system with a null flow, by setting Q_{ij} to 0 for all edges on all nodes. The algorithm presented below then ensures that Q_{ij} is really a flow:

- for any nodes i and j, $Q_{ij} = -Q_{ji}$
- for any node i that is not a source or a sink, if it does not hold a token in this round, $\sum_j Q_{ij} = 0$
- for a source (*resp.* a sink), $\sum_j Q_{ij}$ tends to 1 (*resp.* minus the number of sources).

To build clusters, all clusterheads will be sinks, and all nodes willing to join a cluster will be sources.

3.1 Monitoring the flow and routing walkers

A node i has one variable Q_{ij} for each adjacent edge (i, j). This variable is used to monitor the flow. Each time a walker comes in through a given edge (i, j), Q_{ij} is increased and each time a walker comes out through it, it is decreased. At each round, the value of Q_{ij} is scaled down.

The only type of messages we use is a $Walker$ message, with no content, and there are three types of nodes:

- *sinks*, that at each round update their flow information with equation 6 according to the number of received walkers and subsequently delete them;
- *ordinary nodes*, that at each round update their flow information and forward received walkers;
- *sources*, which act as ordinary nodes but send an extra token per round.

Depending on the choice of sources and sinks, this algorithm may be used to compute various related distributed structures: shortest paths, shortest path DAGs and distance-based clusters. We detail those properties in the next section.

Procedure 1 Upon a $Walker$ reception from j

$w_{in}[j] \leftarrow w_{in}[j] + 1$

Procedure 2 At each round, on an ordinary node (*resp.* source node)

$w_{in} \leftarrow 0$ (*resp.* 1)
for *any adjacent node* j **do**
  $w_{out}[j] \leftarrow 0$
  $w_{in} \leftarrow w_{in} + w_{in}[j]$
  $Q_{ij} \leftarrow Q_{ij} - \alpha(1-\alpha)w_{in}[j]$
  $Q_{ij} \leftarrow \alpha Q_{ij}$
for $i = 0$ *to* w_{in} **do**
  `Choose a neighbor` k `at random according to` Q/L
  `Send` $Walker$ `to` k
  $w_{out}[k] \leftarrow w_{out}[k] + 1$
for *any adjacent node* j **do**
  $Q_{ij} \leftarrow Q_{ij} + (1-\alpha)w_{out}[j]$

Procedure 3 At each round, on a sink node

for *any adjacent node* j **do**
  $Q_{ij} \leftarrow Q_{ij} - \alpha(1-\alpha)w_{in}[j]$
  $Q_{ij} \leftarrow \alpha Q_{ij}$

Procedure 4 Choose a neighbor at random according to Q/L

$sum \leftarrow 0$
for *each adjacent node* j **do**
  if $Q_{ij}/L_{ij} > \beta$ **then**
    $sum \leftarrow sum + Q_{ij}/L_{ij}$
  else
    $sum \leftarrow sum + \beta$
$v \leftarrow$ `random value in` $[0, sum]$
$j \leftarrow$ `first neighbor of` i; $sum \leftarrow Q_{ij}/L_{ij}$
while $sum < v$ **do**
  $j \leftarrow$ `next neighbor of` i
  if $Q_{ij}/L_{ij} > \beta$ **then**
    $sum \leftarrow sum + Q_{ij}/L_{ij}$
  else
    $sum \leftarrow sum + \beta$
$return(j)$

When receiving a walker, a node updates the weight of the edge through which it has come. Then,

if it is not a sink, it chooses a neighbor at random according to Q/L, sends the walker to it and updates the out-edge value (procedures 2, 4 and 1).

Note that choosing a neighbor according to Q/L consists of i choosing a random neighbor j with a probability proportional to $\frac{Q_{ij}}{L_{ij}}$. If this probability $\frac{Q_{ij}}{L_{ij}}$ is smaller than a parameter β we replace it by β, thus a neighbor has always a positive probability of being chosen.

Ordinary nodes, at each round, forward the received walkers and update their variables. The array w_{in} (resp. w_{out}) stores the number of walkers received (resp. sent) on each adjacent edge, so as to update the outgoing flow after the walkers have been sent. Additionally, sources create a new walker at each round and send it: they act as ordinary nodes, except for the computation of w_{in}, to which 1 is added to account for the extra token that the source creates during the round (procedure 2). On the other hand, sinks delete all received tokens but they keep track of the incoming flow (procedure 3).

Note that messages are received one step after they have been sent, so that Q_{ji} has been multiplied by α before i receives the walker sent by j. Thus, to maintain the symmetry $Q_{ij} = -Q_{ji}, \forall (i,j)$ and for the flow to be valid, we are led to subtract $\alpha(1-\alpha)$. On the other hand, note that when a node sends several messages to the same neighbor in a given round, these walkers can be replaced by a single weighted walker, at the expense of introducing a content in the walkers. We ran the algorithm in both settings and called the messages in the latter setting *aggregated messages*.

The above algorithm has two parameters, α and β. The α parameter is used to *"smoothen"* the flow evaluation. The β parameter ensures that all edges will be used, which is necessary for the system not to get *"trapped"* in a suboptimal solution (non shortest paths to the sink with weight 1, which are fixed points of the system dynamics, but repulsive ones). In particular, if topological changes occur, the β parameter allows walkers to visit new and possibly better paths.

3.2 Using flows to solve clustering

When several nodes produce walkers, all of them compute shortest paths to the single sink. The optimal substructure property of the shortest paths problem [13] ensures that the process actually converges to a shortest path DAG. Indeed, for a given sink, starting from the point where they meet, two shortest paths from two different sources to the same sink are the same (or, if shortest paths are not unique, can be the same).

When several sinks are present, the flows do not distinguish between them. The system acts as if they were a unique sink, consisting in the merging of all sinks, and the shortest paths computation leads to flow running along the shortest path to the closest sink.

Consider now a system in which some clusterheads behave as sinks, and nodes seeking clusters behave like sources. After some time, the algorithm will converge to flows circulating along shortest paths from sources to sinks. Thus, nodes sending flow to a given sink are closer to this sink than to any other, and have to join this cluster. In this way, any node will be able to route a message to its sink, by sending the message along its edge with the strongest outgoing flow.

Moreover, the flow brings some extra information: since all sources generate a unit flow, the flow that a sink receives is the number of nodes that are members of its cluster. Similarly, suppose that the shortest paths are unique. Then the flow coming from an edge on a node indicates to this node the size of the corresponding subtree, and all nodes have a local knowledge of the shortest path tree of their cluster. If the shortest paths are not unique they define a DAG, and the flow indicates the number of descendants on this side of the DAG, possibly counted partially as descendants of several nodes.

The clusterhead can then launch a new phase of the algorithm and use the DAG to inform all nodes of its presence if required, allowing to meet the classical clustering specification.

3.3 c-asynchronous case

Synchronicity and round-based distributed processing of messages is a strong assumption. We can relax it to c-asynchronicity, meaning that in a round for a given node, no node has executed more than c rounds.

With respect to slower nodes, a fast source may appear as generating up to c walkers per round. This output cannot be distinguished from a situation where the faster node is replaced with a central node connected to c other nodes in a star formation. The computations of other nodes will run alike and lead to shortest paths to these nodes. Since the shortest paths to these virtual nodes all go through the central node, the computed shortest paths will be consistent.

A fast ordinary node, or a fast sink, will output the right number of walkers (all received walkers are sent forth and received in the next round by slower neighbors), and thus will not interfere with the dynamics. A slow ordinary node (or a slow sink) will also output the right number of walkers on average, but its output will burst periodically while being null the rest of the time. If α is too low (putting the focus on short-term monitoring of the flow) and/or the processor too slow, this may hinder the convergence of the system. Nevertheless, with a bound on c depending on α and β, the system and dynamics will converge to shortest paths.

Note that a slow node generates a weaker flow, if seen by the perspective of a faster one. This flow is taken into account if it is not lower than β. Indeed, if it is lower than β it will be replaced by β in the routing of walkers and the slow node will not contribute to the dynamics.

4 Simulation results on clustering

The scheme presented in the previous section computes shortest paths between sources and sinks.

We ran simulations with different settings:

- one source, one sink (shortest path);
- all nodes are sources, except for one node which is a sink (shortest paths DAG);
- all nodes are sources, except for two nodes which are sinks (distance-based clustering with shortest path DAGs);
- all nodes are sources and sinks have dynamically changed.

We simulated the algorithm on unweighted 10×10 grids, on unweighted random graphs (100 nodes, $p = 0.2$) and on randomly weighted grids (weights were integer drawn uniformly at random between 1 and 10). Grids were chosen because the structure of the shortest paths in those graphs is easy to describe, and because they are far from being unique. Random graphs are intended to be closer to real-world distributed systems.

The simulations were run using DASOR [14]. All graphical results in this section show arrows proportional to the flow. The simulation campaign is at its beginning and we do not claim statistical accuracy of these data. Each presented figure is the average of 100 simulation results, together with its standard deviation: the number of messages, of aggregated messages (i.e. weighted messages representing several walkers, as explained in the previous section), the convergence time or measures of the distance to the solution (detailed for each measure in the relevant subsection) were recorded for each simulation, and then averaged for each round of simulations in a given setting.

4.1 Preliminary: Shortest path (1 source 1 sink setting)

When there is only one source and only one sink in the system, the system dynamics allow to compute a shortest path (or several of them) between the source and the sink.

We considered that the system had achieved convergence when the strongest flow ran from the sink to the source on a shortest path, and remained so until the end of the simulation (2000 rounds).

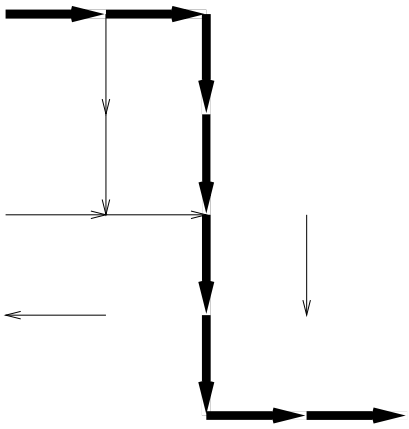

Figure 1: Flow of walkers on shortest paths (with some remaining extra flow)

In an unweighted 10×10 grid with a source and a sink located at opposite corners, with $\alpha = 0.995$ and $\beta = 0.01$, on average, the system converges in 560 rounds (standard deviation: 142), 147,808 messages (standard deviation: 68,407) and 52,095 messages (standard deviation: 19,306) if we aggregate them.

In a randomly weighted grid (with weights uniformly distributed between 1 and 10) with a source and a sink located at opposite corners, with $\alpha = 0.995$ and $\beta = 0.01$, convergence took 691 round on average (standard deviation: 576), 9,542,305 messages (standard deviation: 9,684,811) and 205,930 aggregated messages (standard deviation: 163,589).

In a random graph with 100 nodes and $p = 0.2$ with random source and sink, with $\alpha = 0.999$ and $\beta = 0.01$), convergence was realized after 31 rounds (standard deviation: 12), 38,905 messages (standard deviation: 22,515), or 18,930 aggregated messages (standard deviation: 9219).

Those simulation results confirm the convergence of the discrete dynamics to shortest paths (see figure 1). Time and messages complexity are of the order of the ones of existing distributed shortest path algorithm ($O(n)$ and $O(mn)$ respectively) on the two unweighted settings. The results on weighted grids need to be further investigated, with different choices of α and β.

4.2 Shortest paths DAG of a cluster: n sources 1 sink setting

In these simulations, we measured convergence by counting the average number of nodes the main outgoing flow of which was not directed to a node closer to the sink than the current node (between parentheses are the standard deviations). All nodes are sources, but for one node (a corner node in the case of the grid) that is a sink.

	25 rounds	50 rounds	100 rounds	200 rounds
convergence	44.5 (5.19)	37.98 (3.19)	1.51 (0.15)	0.48 (0.07)
#msg	26,004 (175)	104,772 (756)	385,795 (3200)	1,238,815 (11,024)
#ag msg	6,339 (43)	14,306 (107)	29,295 (244)	58,384 (606)

Figure 2: Reconfiguration results on a 10×10 grid with two sinks and $\alpha = 0.975$ and $\beta = 0.02$

	50 rounds	100 rounds	200 rounds	400 rounds
convergence	38.77 (5.77)	12.86 (3.12)	4.28 (2.01)	1.28 (1.25)
#msg	85,408 (4,292)	140,042 (12,289)	191,531 (16,858)	269,050 (23,304)
#ag msg	51,912 (1,529)	88,148 (5,122)	125,615 (7,518)	179,860 (10,515)

Figure 3: Convergence results on a random graph with 100 nodes and $p = 0.2$, with one sink $\alpha = 0.999$ and $\beta = 0.01$

These results (figures 2 and 3) show that the algorithm converges to a shortest path DAG quite

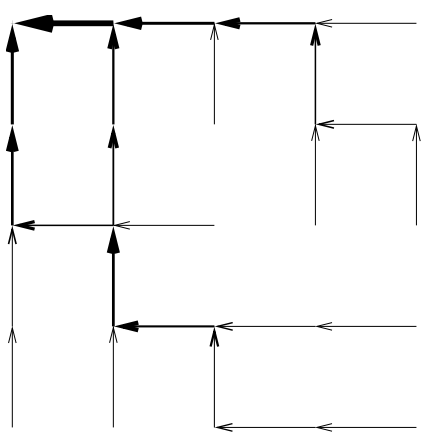

Figure 4: Flow of the walkers on a shortest paths DAG

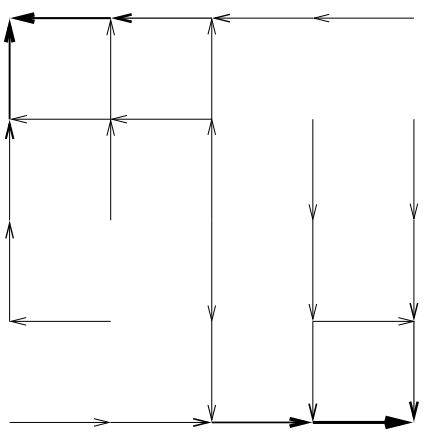

Figure 5: Computation of two clusters with their shortest paths DAGs

fast for most nodes (see 4), and then takes time to reach full convergence. In a grid, aggregation of walkers reduces highly the number of exchanged messages. The very conservative choice of α explains why convergence is slow in the second round of simulations. The high number of messages in the first round of simulations make the use of aggregated walkers interesting.

4.3 Clustering (n sources, several sinks setting)

The simulations were ran on a 10×10 grid, starting from an initial configuration with a unique source at a corner.

In these simulations, we measured convergence by counting the average number of nodes the main outgoing flow of which was not directed to a node closer to the sink than the current node (between parentheses are the standard deviations). All nodes are sources, but for two nodes (opposite corner nodes in the case of the grid) that are sinks.

	25 rounds	50 rounds	100 rounds	200 rounds
convergence	19.4 (4.37)	3.89 (2.28)	1.33 (1.24)	0.55 (0.79)
#msg	24,394 (231)	92,060 (1,082)	290,887 (4,414)	710,330 (11,769)
#ag msg	6,192 (51)	13,666 (109)	27,607 (283)	54,230 (687)

Figure 6: Convergence results on a 10×10 grid with two sinks and $\alpha = 0.975$ and $\beta = 0.02$

	50 rounds	100 rounds	200 rounds	400 rounds
convergence	39.3 (6.5)	12.41 (3.43)	4 (2.05)	1.08 (1.13)
#msg	86,601 (5,148)	147,325 (36,828)	205,162 (105,107)	303,000 (298,876)
#ag msg	51,427 (5,230)	87,955 (9,376)	126,313 (9,942)	180,939 (12,371)

Figure 7: Convergence results on a random graph with 100 nodes and $p = 0.2$, with two sinks and $\alpha = 0.999$ and $\beta = 0.01$

The results with several sinks (figures 6 and 7) illustrate the reduction in the complexity of the algorithm when several sinks are present. The use of these dynamics allows to compute a clustering, as illustrated by figure 5: nodes can use their incoming flows to reach the closest clusterhead.

4.4 Reconfiguration of clusters after a topological change

The process presented is tolerant to topological changes: starting from a configuration with flows running to a given sink, thanks to the β parameter, the apparition of a new sink, or a change in a sink make the

flow so as to adapt to the new situation. The nodes concerned by the change in the situation see their flow change; nodes to which they were related are also subject to a change in the flow circulation. But the dynamics of further nodes is not affected by this topological change.

The simulations were ran on a 10×10 grid, starting from an initial configuration with a unique sink at the bottom right corner, and a consistent stable flow (all nodes send all incoming flow to their bottom neighbor if they have one, or to their right neighbor). Then, two rounds of simulation were ran: one with a new sink appearing at the opposite corner, and another with the original sink becoming an ordinary node, and a new sink appearing at the opposite corner.

In these simulations, we measured convergence by counting the average number of nodes the main outgoing flow of which was not directed to the a node closer to the sink than the current node (between parentheses are the standard deviations). All nodes are sources, but for two nodes (opposite corner nodes in the case of the grid) that are sinks.

In these simulations, we measured the average time before two nodes or less remain with a flow directed to a node not closer to the sink than the current one. The parameters were $\alpha = 0.92$ and $\beta = 0.02$.

	25 rounds	50 rounds	100 rounds	200 rounds
convergence	27.11 (3.2)	6.74 (2.52)	1.27 (1.08)	1.21 (1.05)
#msg	24,703 (205)	94,083 (842)	303,169 (3,159)	739,326 (7810)
#ag msg	5,001 (48)	12,329 (87)	26,618 (205)	54,212 (477)

Figure 8: Convergence results on a 10×10 grid after the apparition of a second sink, with $\alpha = 0.92$ and $\beta = 0.02$

The reconfiguration of the algorithm after a topological change is carried out rapidly for most nodes. A closer examination of the simulation shows that the nodes closer to the clusterhead that was already in place do not modify much their flow structure (see figure 9).

5 Conclusions

Based on the analogy between the growth of the mould Physarum and the behavior of electrical flow, we proposed a randomized distributed algorithm in which walkers emulate electrical current (as studied by [17]), while the resistance of the edges are modified by the flow according to a discrete-time version of the equations in [16].

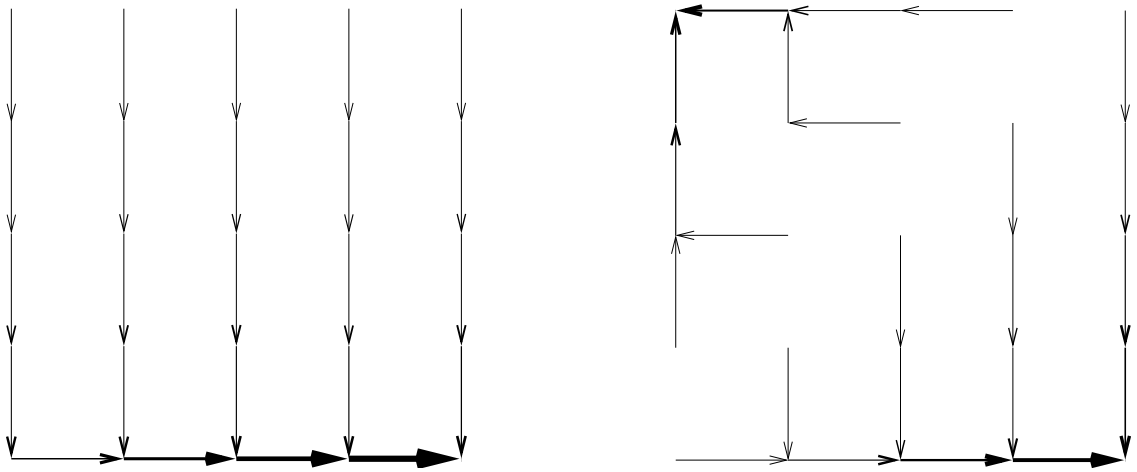

Figure 9: Convergence after a new sink has appeared (on the left, initial configuration; on the right, configuration after convergence)

As expected, the simulations show that the dynamics of this algorithm converge to flows running along shortest paths from sources to sinks. This algorithm is local, and uses only one type of message, with no content. This may compensate partially for the high number of messages needed to reach full convergence. Also, it may be noted that most nodes find a path to their clusterhead in the very first rounds. After a topological change, nodes soon find their new cluster.

This work provides an insight on flows of walkers, and on the interest of biasing the walkers in a non-markovian fashion (the future moves of walkers depend on their past moves). Interesting questions following these results are the choice of the algorithm parameters and a more comprehensive study of the theoretical foundations of this discrete process.

References

[1] A.D. Amis, R. Prakash, T.H.P. Vuong and D.T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 32-41, 2000.

[2] S. Basagni. Distributed clustering for ad hoc networks. In *International Symposium on parallel Architectures, Algorithms and Networks (ISPAN)*, pages 310–315, 1999.

[3] T. Bernard, A. Bui, L. Pilard and D. Sohier. Distributed Clustering Algorithm for Large-Scale Dynamic Networks. *International Journal of Cluster Computing*, Springer eds. DOI: 10.1007/s10586-011-0153-z. 2010.

[4] A. Bui, S. Clavière, A. K. Datta, L. L. Larmore and D. Sohier. Self-Stabilizing Construction of Bounded Size Clusters. *8th International Colloquium on Structural Information and Communication Complexity SIROCCO 2011*, Lecture Notes in Computer Science, Springer, 2011.

[5] A. Bui, A. Kudireti and D. Sohier. An adaptive random walks based distributed clustering algorithm. In *International Journal of Foundations of Computer Science*, vol. 23(4), pages 802–830, 2012 (*to appear*).

[6] A. K. Datta, L. L. Larmore and P. Vemula. A self-stabilizing $O(k)$-time k-clustering algorithm. *The Computer Journal*, 53(3): pages 342–350, 2010.

[7] S. Dolev and N. Tzachar. Empire of colonies : Self-stabilizing and self-organizing distributing algorithm. *Theoretical Computer Science*, 410: pages 514–532, 2009.

[8] A. Ephremides, J.E. Wieselthier and D.J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, pages 56-73, 1987.

[9] C. Johnen and L. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, 410(6-7): pages 581–594, 2009.

[10] J. Sucec and I. Marsic. Location management handoff overhead in hierarchically organized mobile ad hoc networks. In *International Parallel and Distributed Processing Symposium*, (IPDPS), page 198, 2002 2:0194, 2002.

[11] D.G. Thaler and C.V. Ravishankar. Distributed top-down hierarchy construction. In *IEEE INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 693-701 vol.2, 1998.

[12] S-J. Yang and H-C. Chou. Design Issues and Performance Analysis of Location-Aided Hierarchical Cluster Routing on the MANET. In *Communications and Mobile Computing*, (CMC), pages 26–31, 2009.

[13] R. Bellman (1957), *Dynamic Programming*, Princeton University Press. Dover, 1957.

[14] C. Rabat. Dasor, a Discret Events Simulation Library for Grid and Peer-to-peer Simulators *Studia Informatica Universalis*, Volume 7, 2009.

[15] A. Adamatzky and P.P.B. de Oliveira. Brazilian highways from slime mold's point of view. *Kybernetes*, 40(9):1373–1394, 2011.

[16] V. Bonifaci, K. Mehlhorn and G. Varma. Physarum can compute shortest paths. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–240, 2012.

[17] P. G. Doyle and L. J. Snell. *Random Walks and Electrical Networks.* Mathematical Association of America, Dec 1984.

[18] C. Gkantsidis, M. Mihail and A. Saberi. Random walks in peer-to-peer networks: Algorithms and evaluation. *Performance Evaluation*, 63(3):241–263, 2006.

[19] K. Ito, A. Johansson, T. Nakagaki, and A. Tero. Convergence properties for the physarum solver. *arXiv:1101.5249*, January 2011.

[20] A. Johannson and J. Zou. A slime mold solver for linear programming problems. In S. Cooper, A. Dawar and B. Lowe, editors, *How the World Computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 344–354. Springer Berlin / Heidelberg, 2012.

[21] C. Lenzen, J. Suomela and R. Wattenhofer. Local algorithms: Self-stabilization on speed. *Stabilization, Safety and Security of Distributed Systems*, page 1734, 2009.

[22] K. Li, C. Torres, K. Thomas, L. Rossi and C.-C. Shen. Slime mold inspired routing protocols for wireless sensor networks. *Swarm Intelligence*, 5(3):183–223, 2011.

[23] T. Miyaji. Mathematical analysis to an adaptive network of the plasmodium system. *Hokkaido Mathematical Journal*, 36(2):445–465, May 2007. Mathematical Reviews number (MathSciNet): MR2347434.

[24] T. Miyaji and I. Onishi. Physarum can solve the shortest path problem on riemannian surface mathematically rigourously. *International Journal of Pure and Applied Mathematics*, 47(3), 2008.

[25] T. Nakagaki, A. Tero, R. Kobayashi, I. Onishi and T. Miyaji. Computational ability of cells based on cell dynamics and adaptability. *New Generation Computing*, 27(1):57–81, 2008.

[26] A. Tero, S. Takagi, T. Saigusa, K. Ito, D.P. Bebber, M.D. Fricker, K. Yumiki, R. Kobayashi and T. Nakagaki. Rules for biologically inspired adaptive network design. *Science*, 327(5964):439–442, January 2010.

[27] D. Wagner and R. Wattenhofer, editors. *Algorithms for Sensor and Ad Hoc Networks, Advanced Lectures [result from a Dagstuhl seminar]*, volume 4621 of *Lecture Notes in Computer Science.* Springer, 2007.

[28] G. Georgiadis and M. Papatriantafilou. A Least-Resistance Path in Reasoning about Unstructured Overlay Networks, In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing, Euro-Par '09*, pages 483-497, 2009

[29] A. Bui and D. Sohier. How to compute times of random walks based distributed algorithms, *Fundamenta Informaticae*, 80(4), pages 363-378, 2007

6 Appendix

6.1 Analysis of algorithm's dynamics

The Physarum dynamics followed by the PECan algorithm, as well as their specific implementation, are defined by equations 5 and 6. Combining these two equations we get the following expression for the edge diameters:

$$
\begin{aligned}
D_{ij}(t+1) &= (1-\varepsilon) D_{ij}(t) + \varepsilon |Q_{ij}(t)| = \\
&= \begin{cases}
(1-\varepsilon) D_{ij}(t) + \varepsilon |\alpha Q_{ij}(t-1) + (1-\alpha)\cdot 1|, (X_t, X_{t+1}) = (i,j) \\
(1-\varepsilon) D_{ij}(t) + \varepsilon |\alpha Q_{ij}(t-1) - (1-\alpha)\cdot 1|, (X_t, X_{t+1}) = (j,i) \\
(1-\varepsilon) D_{ij}(t) + \varepsilon |\alpha Q_{ij}(t-1)|, (X_t, X_{t+1}) \neq (i,j), (j,i)
\end{cases} \\
&= \begin{cases}
(1-\alpha') D_{ij}(t) + \alpha', & \forall (i \to j, Q_{ij}(t-1) \geq 0) \\
& \quad \vee (j \to i, Q_{ij}(t-1) < 0) \\
(1-(2\varepsilon-\alpha')) D_{ij}(t) + \alpha', & \forall \left(i \to j, -\frac{(1-a)}{a} \leq Q_{ij}(t-1) < 0\right) \\
& \quad \vee \left(j \to i, 0 \leq Q_{ij}(t-1) < \frac{(1-a)}{a}\right) \\
(1-\alpha') D_{ij}(t) - \alpha', & \forall \left(i \to j, Q_{ij}(t-1) < -\frac{(1-a)}{a}\right) \\
& \quad \vee \left(j \to i, Q_{ij}(t-1) \geq \frac{(1-a)}{a}\right) \\
(1-\alpha') D_{ij}(t), & otherwise
\end{cases}
\end{aligned}
\tag{8}
$$

where $\alpha' = \epsilon(1-\alpha)$.

In the following lemma we use lemma 14 of [16] which states that, if a unique shortest path between source and sink exists, the flow over any edge not on the shortest path converges to zero. The lemma is applicable here since it uses the same definitions of electrical metrics (i.e. $D_{ij}(t), C_{ij}(t)$), the same dynamics (equation 5) and Ohm's law of electric currents which is also valid in our case. The following lemma verifies that the diameter takes the value 1 on edges belonging to the shortest path and 0 otherwise, and shows that the PECan algorithm's implementation of the specific dynamics follows closely the analysis of Bonifaci et al [16].

Lemma 1 *If there is a unique shortest path from source to sink and the dynamics stabilize, the diameter of edges on the shortest path converge to 1 while those not on the shortest path to 0.*

Proof: Let τ be the stabilization time after which there is no flow over edges not belonging on the shortest path (lemma 14 of [16]). Since the flow is defined by equation **??**, it implies that the walker moves only over edges on the unique shortest path between source and sink. For each edge (i,j) on the shortest path, the walker will move always in the direction of the flow, after which point we know from equation 8 that $D_{ij}(t+1) = (1-\alpha') D_{ij}(t) + \alpha'$. Solving the recursion we get

$$
\begin{aligned}
D_{ij}(t+1) &= \left(1-\alpha'\right) D_{ij}(t) + \alpha' = \\
&= \left(1-\alpha'\right)^{t+1-\tau} D_{ij}(\tau) + \alpha' \frac{1-(1-\alpha')^{t+1-\tau}}{1-(1-\alpha')} = \\
&= \left(1-\alpha'\right)^{t+1-\tau} (D_{ij}(\tau) - 1) + 1
\end{aligned}
$$

which means that for $t \to +\infty$ we get $D_{ij} \to 1$, since $\alpha' \leq 1$.

Similarly, for each edge (i,j) not on the shortest path the walker will not move through it. Solving the recursion gives

$$
\begin{aligned}
D_{ij}(t+1) &= \left(1-\alpha'\right) D_{ij}(t) = \\
&= \left(1-\alpha'\right)^{t+1-\tau} D_{ij}(\tau)
\end{aligned}
$$

which means that for $t \to +\infty$ we get $D_{ij} \to 0$. □