NATURAL COMPUTING

Exploring the properties of simulated Physarum polycephalum transport networks

VICTOR-GABRIEL PETRE *s1097736*
BRAM PULLES *s1015194*
Radboud University *June 14, 2024*

1 Introduction

1.1 Transport networks, and *Physarum polycephalum*

Transport networks are networks describing possibilities for movement across space, and their careful construction is crucial for much of the infrastructure on which society relies (e.g. public transport systems, the power grid, etc.). While transport networks must be performant (fast, spanning the smallest possible distance between destinations), they must also be fault-tolerant (alternative routes should be available in case of localized network damage). This resilience requires some degree of redundancy, but is expected to increase implementational cost. As such, a primary challenge in transport network design is balancing performance, fault-tolerance, and cost-effectiveness. Historically, such networks were created without following universally-agreed-upon principles, generally prioritizing performance and low cost at the expense of fault-tolerance.

In response, some are turning to organisms known to create transport networks in nature for inspiration, on the assumption that natural selection will have pressured such organisms into maintaining performant, fault-tolerant networks while expending minimal energy to create them. One such organism is the unicellular slime mold *Physarum polycephalum*, which forages by creating tubular networks connecting patch-like food sources together. These connections can either directly bridge between food sources (akin to a graph defined using vertices and edges connecting them), or create additional junctions between food sources (Steiner points). In general, the resulting networks are both performant and resilient (Nakagaki et al., 2004a,b), raising questions about whether simulations of *Physarum* transport network creation might be useful for designing transport networks in the real world. To this end, Tero et al. (2010) allowed *Physarum* to forage in an environment modeled after the Tokyo rail system, and compared the resulting transport network to the real rail system, finding comparable cost-effectiveness, travel efficiency and fault-tolerance between the two. Following this, Tero et al. (2010) proposed an algorithm for simulating such transport network creation. Although promising, their model was created using a mostly top-down modelling approach, and lacks the primary intuitions behind foraging; a dense network of fine tubes is initialized over the entire space, and the thicknesses of tubes are iteratively updated in proportion to the rate of protoplasmic flow through them at the current timepoint. At each timepoint, two food sources (representing areas of dense population where stations would be present) are randomly selected, where one is set to drive flow through the network at the current time point, and the other is the sink, keeping the total amount of fluid constant. This approach disregards the chemotaxis of *Physarum* towards food sources (Chet et al., 1977), and does not allow for any true "foraging"; all possible paths for the simulation to take are initialized at the first timepoint.

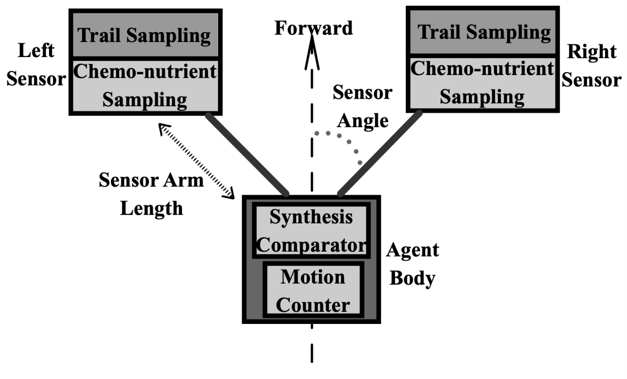
Alternative, bottom-up particle-based *Physarum* models with much fewer underlying assumptions have been proposed. Building upon initial work by Jones (2010), Wu et al. (2012) proposed a particle-based model for *Physarum* transport network generation, where individual particles ("agents") move stochastically while depositing a chemo-attractant trail in their current location. Agents are attracted towards other agents' trails as well as chemo-attractants diffusing from food sources, constructing a network of agents bridging the space between the food sources. Wu et al. (2012) show a significant degree of overlap between their model and the behavior of *Physarum in vivo*, however the authors do not formally quantify properties of the resulting transport networks, nor do they thoroughly investigate the influence of hyperparameter choice on said network properties. This latter facet is crucial, as our experiments have shown the behavior of their model to be very sensitive to hyperparameter values. In response, we modify the model proposed by Wu et al. (2012) to be compatible with a novel transport network generation task, and quantify the cost, transportation efficiency, and fault tolerance of the resulting networks with various hyperparameter value combinations.

2 Methods

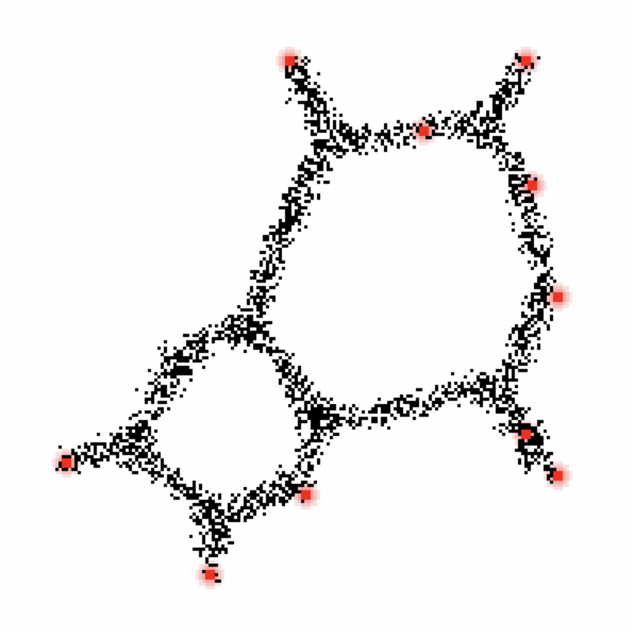
2.1 *Physarum* model

Our model extends that proposed by Wu et al. (2012). We first outline the model properties copied from Wu et al. (2012), and then discuss the differences introduced in our approach. All experiments are carried out in a field of 165 $\times$ 165 pixels, where agents are represented using single-pixel particles that behave according to the properties of their local environment. Each agent is equipped with two sensors at 45 degrees relative to the agent's forward direction, where the length of the sensor arm is controlled via a parameter `sensor_length` (see Figure 1**(a)**). Agents are uniformly initialized within the field with a probability of `initial_population_density`, with an initial direction randomly sampled from the possible 8. At each update iteration, an agent attempts to move forward one step along its direction vector within the field. If there is not another agent in this position and the movement would not drive the agent outside the boundaries of the field, the movement is performed, and the agent deposits an amount `trail_deposit` of trail chemo-attractant at its new location. Following this, the agent computes a weighted sum of trail and food chemo-attractants at each sensor position (weights of the two terms are set as `trail_weight` and 1−`trail_weight`, respectively), and turns towards the direction of the sensor with the highest value. It can happen that either sensor is inside a wall or outside of boundaries, in this case the agent rotates 90 degrees in the opposite direction, when this holds for both sensors the agent rotates 180 degrees. If another agent is present at the prospective new position or moving towards this position would cause the agent to go out of bounds, the agent is instead rotated randomly towards another direction. All agents have an internal counter parameter which is incremented by 1 when the agent moves successfully and decremented by 1 otherwise. If this internal counter exceeds the `reproduction_threshold`, a new child agent is generated in the parent agent's old position. If the counter decreases below the `elimination_threshold`, the agent is removed from the field. Together, these rules allow the survival and reproduction of agents which successfully keep moving, using the locations with the largest chemo-attractant concentrations as direction guides.

A complete update iteration occurs after all aforementioned update steps have been computed for all agents. Food sources are initialized within the field by depositing an amount `food_deposit` at specific pixels at the beginning of the simulation. After each update iteration, the trail and food chemo-attractant grids are each convolved with an average filter of sizes `trail_filter_size, food_filter_size`, respectively, and are multiplied by factors of $(1 -$ `trail_damping`$)$, $(1 -$ `food_damping`$)$, respectively. Furthermore, the food sources are kept to a constant value of `food_deposit` across the entire simulation. These operations cause diffusion of food and trail chemo-attractant across the grid over time, while also preventing chemo-attractant buildup. See Figure 1**(b)** for an example of the types of transport networks this specification can generate.



(a)



(b)

Figure 1: **(a):** Sensor structure as proposed by Wu et al. (2012), figure from original authors. Each sensor computes a weighted sum of the food and trail chemo-attractants, and both sensors are placed at a 45 degree angle relative to the agent's forward direction. **(b):** Example network generated with the exact specification of Wu et al. (2012). Black pixels are individual agents, red pixels are food sources.

While running simulations with the aforementioned specifications under the constraints listed in Section 2.2, we found the resulting networks to have an extreme degree of redundancy regardless of hyperparameter choice. This is because Wu et al. (2012)'s specification leads to agents surviving indefinitely as long as they can continue to move, with no regard for whether these agents have access to food or not. As this is biologically implausible (the entire motivation behind transport network creation is foraging) and leads to undesirable behavior, we implemented additional logic which penalizes agents for not spending time in proximity of food chemo-attractant. At each update iteration, if the food chemo-attractant at the agent's position is smaller than some `starvation_threshold`, the agent's internal counter gets decremented by some amount `starvation_penalty`, bringing it closer to elimination. This brings our simulations closer to the pruning behavior found in real *Physarum* networks (Nakagaki et al., 2004a,b). Finally, we implement additional logic allowing agents to "pick up" food and deposit it in their trail, as a rough approximation for signal propagation in biological *Physarum* (Alim et al., 2017). Agents have an internal food counter which is incremented by the value of the food grid at their current position, if the value at the current position exceeds the parameter `food_pickup_threshold`. This value is capped at `food_deposit`, such that agents cannot hold excessive amounts of food. When possible, each agent will drop `food_drop_amount` food from its internal counter at its current position at each iteration. This encourages networks to form bridges between food sources which are close together, which mimics biological *Physarum* (Nakagaki et al., 2004a,b).

2.2 Transport network property quantification

Tero et al. (2010) quantify the fault tolerance, cost, and transport performance of networks using 3 metrics, all operating on graph-based network representations. Fault tolerance (FT) is the probability that nodes in the network remain connected when a randomly-selected edge is removed. This metric is bounded between 0 and 1, where high FT scores indicate multiple redundant paths between nodes, and a network robust to local damage. The cost of the network ($\mathrm{TL}_{\mathrm{MST}}$) is quantified by dividing the sum of the lengths of all edges by the length of the corresponding minimal spanning tree (MST). This normalizes the total length of the network by the minimum possible length, and has a minimum value of 1 for valid (i.e. connected) networks. Transport performance ($\mathrm{MD}_{\mathrm{MST}}$) is computed by dividing the total length of the MST by the sum of the minimal network distances between all pairs of nodes. The metric indicates the efficiency of transportation relative to the most efficient solution possible, and is also constrained between 0 and 1.

Computing these metrics requires graph-based network representations. Given that our final networks are represented via pixels in the field (see Figure 1**(b)**, note in particular that the process generates Steiner points with non-fixed locations in addition to traditional graph edges between pre-specified nodes), modifications had to be made to the experimental protocol such that graphs could reliably be extracted. Instead of generating graphs in empty fields, we constrain the shapes of possible generated graphs by placing immovable square walls with even spacing inside the field. Doing this restricts the scope of the problem; the only places where graph nodes can be generated are in the locations indicated in Figure 2**(a)**. Checking for presence of nodes is a matter of verifying how many agents are present within each square-shaped patch, and this information can also be used to determine whether nodes are connected by slime mold or not. See Figure 2**(b)** for an example simulation in this new environment. Formally, our simulations no longer create networks, but rather rectilinear Steiner trees (Garey and Johnson, 1977).

As simulations are highly dynamic, we extract rectilinear Steiner tree representations based on a pooled agent history. We examine the last 30 simulation timepoints, and mark all positions with agents in them across any of the 30 timepoints as "filled". We then examine each non-wall square patch in this pooled history (each node in Figure 2**(a)** is at the center of one such patch), and mark it as "filled" if there is at least one agent particle within the patch. We then attempt to connect adjacent filled patches, by dividing each patch into four triangles (see Figure 3**(a)**). If, within the pooled history, there are at least two agents in two adjacent triangles, we deem the corresponding nodes to be connected. Figures 3**(b)**, **(c)** show the results of running this graph extraction protocol on a simulation. From this representation, network properties can be quantified.

"Filled" patches without any edges towards their corresponding nodes are removed in post-processing, and we filter out all non-connected graphs prior to analysis. Furthermore, modifications to the standard ways of computing $\mathrm{MD}_{\mathrm{MST}}$ and $\mathrm{TL}_{\mathrm{MST}}$ are necessary due to the way our networks are represented. In Tero et al. (2010), the only network nodes were the food sources, with variable-length edges connecting them. Our representations use unit-length edges with extra nodes in addition to the food sources (see Figure 2**(a)** for an overview of all possible nodes). While the MST for a standard graph is polynomial-time computable, finding the minimum rectilinear Steiner tree (MRST) is an NP-complete problem (Garey and Johnson, 1977). As such, we use the method proposed by Mehlhorn (1988) to calculate the approximate length of the MRST connecting food source nodes. This approximates the theoretical minimum MRST size of the problem instance, by which we divide the sum of edge lengths in the resulting *Physarum* network to compute the $\mathrm{TL}_{\mathrm{MRST}}$ metric. $\mathrm{MD}_{\mathrm{MST}}$ is adapted similarly, dividing the total length of the approximate MRST computed on the entire field

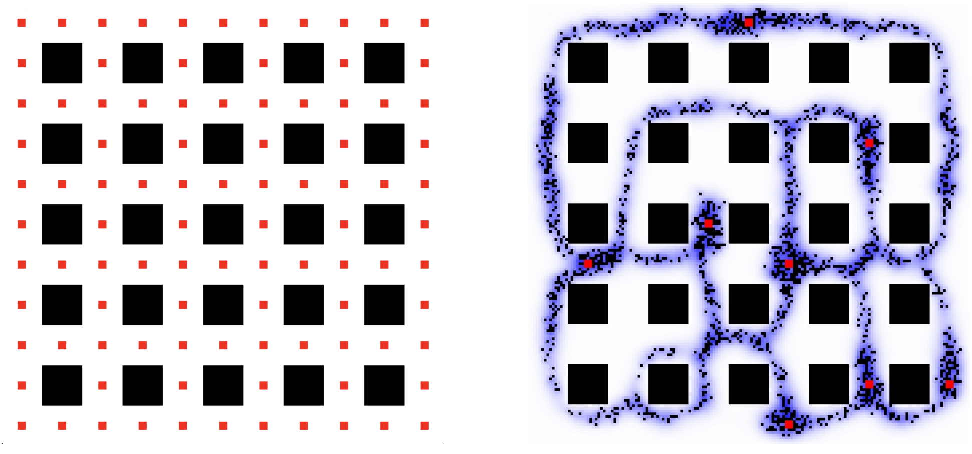

(a) (b)

Figure 2: **(a):** Field with square walls initialized (black), with red points indicating all possible graph node locations. With this restriction, network representations are easy to extract, while also allowing the computation of the metrics used in Tero et al. (2010). **(b):** Example simulation with square walls. Red points indicate food sources, agents are in black, and blue indicates agent trails.

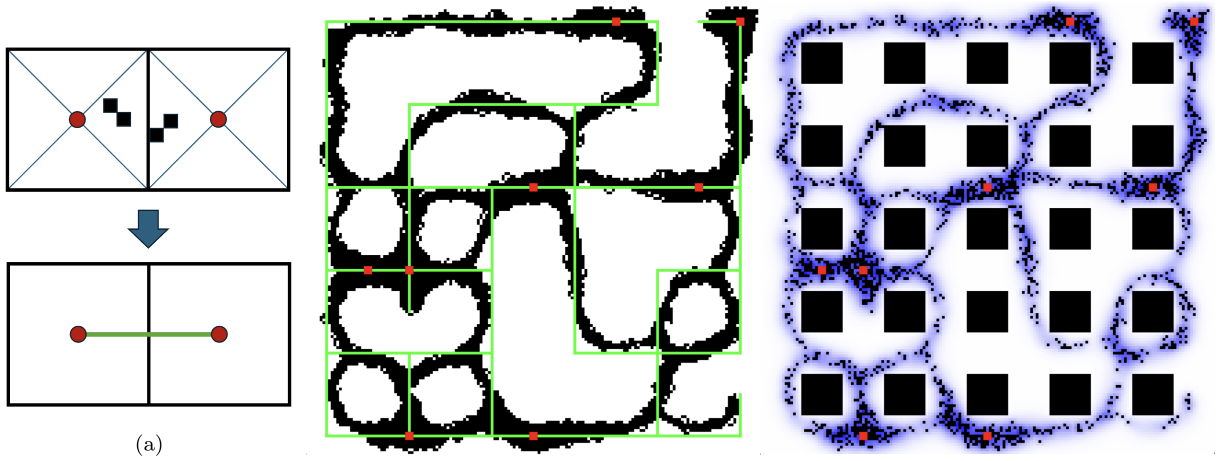

(a) (b) (c)

Figure 3: **(a):** Visualization of the method we use to determine whether nodes in the graph are connected. We deem two adjacent nodes as connected if and only if two adjacent patch triangles both have at least two agents present within them. This method is applied not only to the food source nodes, but all nodes for which the corresponding patch has been deemed "filled" (see Section 2.2). **(b):** Resulting rectilinear Steiner tree representation extracted from the pooled simulation history across 30 timepoints. Agent simulation history is indicated in black, food sources are red, and the network created via the process in Section 2.2 is shown in green. **(c):** Simulation state at the last timepoint, corresponding to the history and graph in **(b)**.

by the total length of the approximate MRST computed on the *Physarum* network solution to the problem (this adapted version is defined as $\mathrm{MD}_{\mathrm{MRST}}$). We still compute FT as the probability of food source nodes remaining connected when an edge is removed, but note that the metric measures something slightly different in rectilinear Steiner trees. In Tero et al. (2010), all edges are between food nodes, whereas edges also span non-food-source-nodes in our representations. As a result, the FT metric in this context is blind to the difference between "useful" edges (those connecting food source nodes) and "useless" edges (redundant parts of the network which do not contribute to food source connectivity, e.g. dead ends). Under our specification, a network with many useless edges is still highly fault-tolerant, despite not being viable in practice. We rely on $\mathrm{TL}_{\mathrm{MRST}}$ to distinguish between these cases; a network with many useless edges will be highly fault-tolerant, but also likely very expensive to build. Additionally, FT in this context also penalizes long non-branching paths between food sources, as disconnecting any of the edges contributing to this path would disconnect the food sources. The longer the path, the more edges which can lead to disconnection, and the lower the FT metric. We discuss the validity of this approach in Section 4.

2.3 Experiments

We find a baseline configuration of parameters which leads to reasonable transport network generation via manual tuning, and then test the effects of modifying certain parameter values independently to see what the effects on the resulting network metrics are. See Table 1 for an overview of the baseline parameters and the values we test. For each simulation, we quantify FT, TL_{MRST} and MD_{MRST}. For each hyperparameter specification, we test the same set of 14 different food source initializations, where each initialization has 8 food sources. We initialize food source positions by uniformly sampling from the graph node positions, and only accept initializations where the food sources span at least 80% of the field area. All simulations are run for 1000 timepoints, and each of the 14 food source initializations has 5 separate simulation trials conducted to account for stochasticity. We do not analyze graphs which are not fully connected, while keeping track of the number of such invalid instances that each parameter combination generates. All simulations are run on a grid of 165×165, with evenly-spaced walls of size 15×15, for a total of 5×5 walls.

Table 1: Baseline parameter values for all simulations, and parameter values modified during experimental runs. For each experimental run, we modify a single parameter value in isolation. Dashes indicate that the baseline parameter value was kept for all simulations. For all experiments manipulating `trail_weight`, `food_weight` was modified accordingly.

Parameter	Baseline value	Experimental values
General		
`sensor_length`	4	-
`reproduction_threshold`	15	10, 15, 20, 25, 30, 35
`elimination_threshold`	-10	-30, -25, -20, -15, -10, -5
`initial_population_density`	0.5	0.01, 0.04, 0.07, 0.1, 0.3, 0.5
Trail-specific		
`trail_deposit`	5	-
`trail_weight`	0.1	0.1, 0.2, 0.3, 0.4, 0.5, 0.6
`trail_filter_size`	(3,3)	-
`trail_damping`	0.1	-
Food-specific		
`food_deposit`	10	-
`food_weight`	1-`trail_weight`	1-`trail_weight`
`food_filter_size`	(3,3)	-
`food_damping`	0.1	-
Own modifications		
`starvation_threshold`	0.1	-
`starvation_penalty`	0.5	0.3, 0.4, 0.5, 0.6, 0.7, 0.8
`food_pickup_threshold`	1	-
`food_drop_amount`	0.3	0, 0.1, 0.2, 0.3, 0.4, 0.5
`food_pickup_limit`	`food_deposit`	-

The majority of baseline parameter values follow the suggestions of Wu et al. (2012), adapted to our smaller grid size. We chose to not modify filter sizes or damping coefficients, as these pertain more to the physics of chemical gradients than behavior of the slime mold itself. All other explored parameters were investigated as they had the most interesting effects in our simulations; the parameters kept constant either had no major effects on the simulation outcome, or were not modifiable without severely compromising the resulting network. Due to the high dimensionality of the resulting parameter space, we could not explore combinations of these parameter values. All code used for running simulations, collecting data for experiments, running statistical analyses, and displaying results can be found at `https://github.com/Borroot/slime-mold`.

3 Results

The average network metrics resulting from running all experiments are shown in Figure 4. The plotted values are the result of averaging across the 5 trials conducted for each food position initialization, and then computing both the average and standard deviation over the 14 resulting values. We only display data where more than 70% of the corresponding networks are fully-connected, and also quantify the extent to which disconnections occur. We perform this filtering step as we are only interested in networks which adequately solve the problem (connecting all food sources without separate islands of edges in the graph). In addition

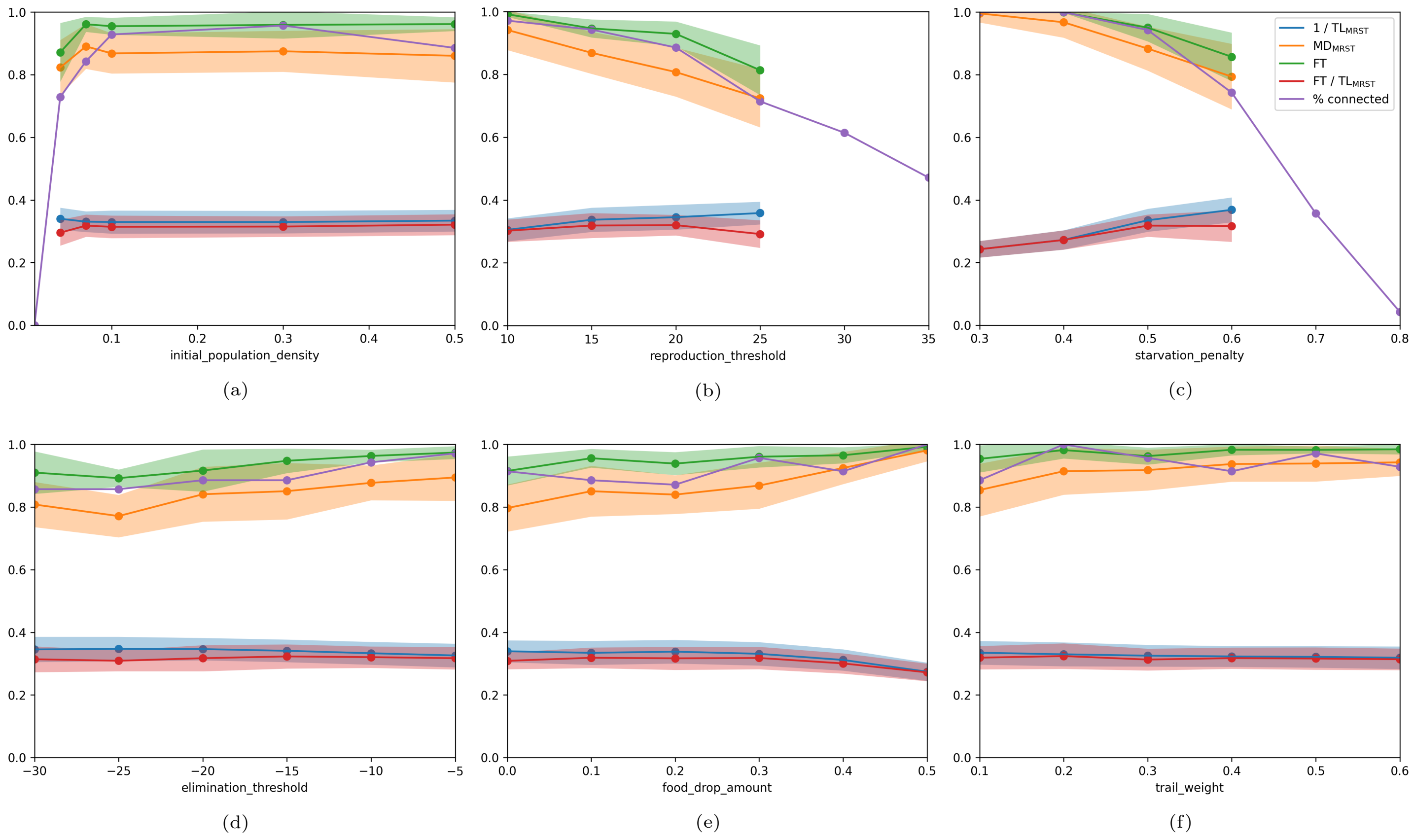


Figure 4: Visualization of network properties resulting from different hyperparameter value combinations. Aside from the parameter values shown on the x-axis, all simulations use the baseline parameter values shown in Table 1. Shaded areas indicate standard deviations. We do not plot data for parameter values where fewer than 70% of the resulting networks were fully-connected, as the smaller sample size may significantly skew statistical results. The percentage of fully-connected networks resulting from each condition has been plotted in purple. We plot the inverse of $\mathrm{TL}_{\mathrm{MST}}$ to constrain its value between 0 and 1 (higher = shorter total network length).

Parameter	$1/\mathrm{TL}_{\mathrm{MRST}}$	$\mathrm{MD}_{\mathrm{MRST}}$	FT	$\mathrm{FT}/\mathrm{TL}_{\mathrm{MRST}}$
`initial_population_density`	$F(4,13) = 0.207, p = 0.93$	$F(4,13) = 1.44, p = 0.23$	$\boldsymbol{F(4,13) = 8.04, p < 0.001}$	$F(4,13) = 1.01, p = 0.41$
`trail_weight`	$F(5,13) = 0.35, p = 0.88$	$\boldsymbol{F(5,13) = 3.39, p = 0.008}$	$\boldsymbol{F(5,13) = 3.33, p = 0.009}$	$F(5,13) = 0.16, p = 0.98$
`elimination_threshold`	$F(5,13) = 0.66, p = 0.65$	$\boldsymbol{F(5,13) = 4.67, p < 0.001}$	$\boldsymbol{F(5,13) = 6.72, p < 0.001}$	$F(5,13) = 0.22, p = 0.95$
`reproduction_threshold`	$\boldsymbol{F(3,13) = 4.74, p = 0.005}$	$\boldsymbol{F(3,13) = 19.11, p < 0.001}$	$\boldsymbol{F(3,13) = 34.47, p < 0.001}$	$F(3,13) = 1.66, p = 0.19$
`starvation_penalty`	$\boldsymbol{F(3,13) = 37.70, p < 0.001}$	$\boldsymbol{F(3,13) = 22.51, p < 0.001}$	$\boldsymbol{F(3,13) = 29.68, p < 0.001}$	$\boldsymbol{F(3,13) = 12.85, p < 0.001}$
`food_drop_amount`	$\boldsymbol{F(5,13) = 6.65, p < 0.001}$	$\boldsymbol{F(5,13) = 13.37, p < 0.001}$	$\boldsymbol{F(5,13) = 8.41, p < 0.001}$	$\boldsymbol{F(5,13) = 4.02, p = 0.003}$

Table 2: Statistical analysis for the impact of varying hyperparameter values on resulting *Physarum* transport network metrics. Aside from the listed parameter, all other parameters were set to the baseline values in Table 1. All analyses involved a one-way repeated measures ANOVA, testing for the effect of an individual parameter's value on each resulting metric across the 14 different food source initializations.

to computing FT, $\mathrm{MD}_{\mathrm{MRST}}$ and $\mathrm{TL}_{\mathrm{MRST}}$, we also computed the ratio $\mathrm{FT}/\mathrm{TL}_{\mathrm{MRST}}$, in order to quantify the extent to which added fault tolerance is worth potential additional costs.

We used a one-way repeated measures ANOVA to test our findings for significance. We chose this approach due to the small number of values tested for each parameter (6 or fewer), and the difficulty with specifying a regression model that could parsimoniously fit all of the required trendlines without overfitting. We tested the effects of all viable hyperparameter values, for each of the tested hyperparameters, by comparing the groups of averages computed on each of the 14 different food source initializations. The shared food source initializations across conditions motivate the use of a repeated measures analysis. The results of the analysis describe the significance of modifying each of the tested hyperparameters on all 4 computed network metrics, the results of which are shown in Table 2.

Table 2 indicates four main patterns. `initial_population_density` appears to not significantly affect our measured network properties of interest, provided its value exceeds 0.1. Smaller values than this cause an extremely sharp drop-off to no fully-connected networks at all, and excluding the datapoints at `initial_population_density = 0.01` removes the one instance of significance found for this parameter dataset (i.e. FT). The most promising results are found for `elimination_threshold` and `trail_weight`, where increasing these parameter values causes a significant change in both fault tolerance (FT) and transport performance ($\mathrm{MD}_{\mathrm{MRST}}$) with no significant change in cost-effectiveness ($1/\mathrm{TL}_{\mathrm{MRST}}$). Although the

ANOVA does not explicitly test for directionality, Figures 4**(d)**,**(f)** suggest significant relationships to be positive. This is especially promising given that the networks generated under these circumstances generally appear quite sparse; as seen in Figure 5, networks generated with hyperparameter values at both extremes are not excessively-connected. Together with Figures 4**(d)**,**(f)**, these properties suggest that higher values of `elimination_threshold` and `trail_weight` promote beneficial modifications to transport network structure.

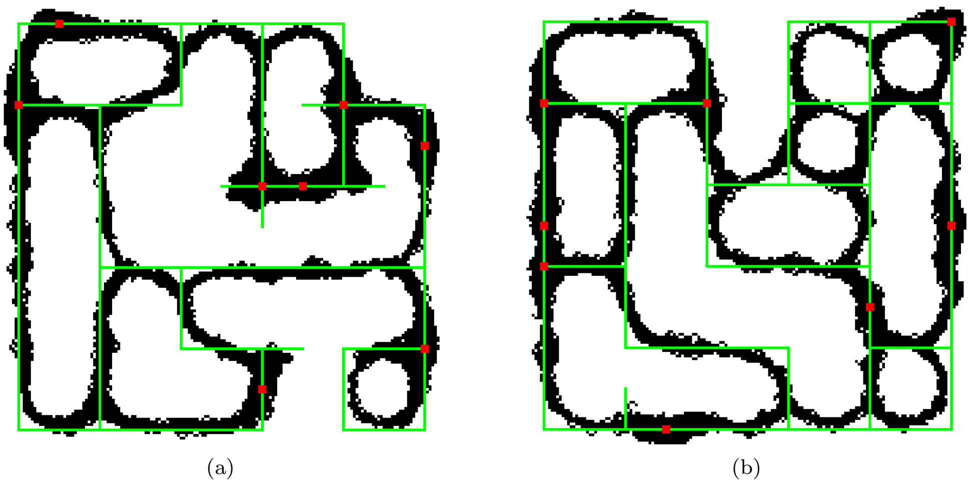


Figure 5: Example networks generated with `elimination_threshold` of -30 (left) and -5 (right). Both networks are rather sparse, despite the large difference in hyperparameter values between them. These same patterns held for `trail_weight`.

In contrast, increasing values of `reproduction_threshold` and `starvation_penalty` have mostly detrimental effects on transport network generation. Aside from larger values leading to too few networks available to include all the intended parameter values in the analysis, network metrics also appear poorer. In both cases there is a significant decrease in fault tolerance and network performance. Although there is a significant increase in cost-effectiveness associated with an increased `reproduction_threshold`, it is not clear if this translates to an increase over the baseline parameter configuration or significantly worse cost-effectiveness with lower `reproduction_threshold` values than the baseline. The same can be said regarding the significant effects of `starvation_penalty` on cost-effectiveness and fault-tolerance/cost ratio (FT/TL$_{\mathrm{MRST}}$). Finally, although results suggest increasing values of `food_drop_amount` to lead to increased fault tolerance and transport performance, this comes at the cost of significantly more expensive networks. Additionally, this increased cost significantly outweighs the potential added fault tolerance, indicated by the significant decrease in FT/TL$_{\mathrm{MRST}}$ ratio with increasing values of `food_drop_amount`. Essentially, increasing this value beyond a certain threshold encourages the creation of densely-connected, expensive networks (see Figure 6).

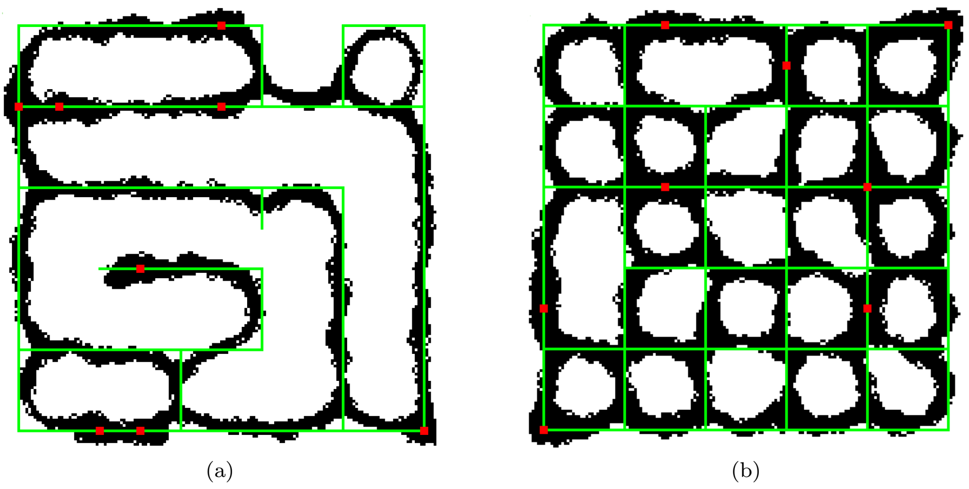


Figure 6: Example networks generated with `food_drop_amount` of 0 (left) and 0.5 (right). Increasing values of `food_drop_amount` leads to networks with more fault-tolerance and better transportation performance, but only because the filling of all possible edges is promoted. This an extremely cost-ineffective solution.

4 Discussion and extensions

As covered in Section 3, we find that increasing `elimination_threshold` and `trail_weight` values generally has beneficial effects on the resulting transport networks, leading to improved fault tolerance and higher transport efficiency between nodes without significantly affecting cost efficiency. By contrast, increasing values of `reproduction_threshold` and `starvation_penalty` have opposite effects on these metrics. Although in both of the latter cases the decreased fault-tolerance was associated with increased cost-efficiency, this difference does not appear to massively improve over the baseline. We argue that the combination of decreased network performance and robustness in conjunction with the sharp decrease in the probability that the simulation produces a usable network in which all nodes are connected makes this potential benefit not worth the tradeoff. We also found that increasing the `food_drop_amount` promoted the formation of densely-connected networks. While this was beneficial for fault tolerance and transport performance, the significantly increased costs associated with this modification make this an undesirable solution, particularly when similar benefits were found with higher values of `elimination_threshold` and `trail_weight` without increased cost. It is worth noting that the individual impacts of these parameters on network metrics cannot necessarily be used to extrapolate about the effects of combinations of parameter values. It is possible that combinations of certain values have unexpected synergistic or destructive effects. In this sense, our method is severely limited; we only explore a handful of parameter values, and we also constrain ourselves to a small island within the possible search space. A possible extension to this would involve using a genetic algorithm to cover more of the parameter search space, though this requires having an objective function which rewards the correct balance of fault tolerance, cost-effectiveness, and transport performance. To an extent, these metrics are contradictory (e.g. fault tolerance requires poorer cost-effectiveness than a minimum spanning tree), which likely makes this balance very difficult to correctly specify. Finally, it is unclear to what extent our findings generalize to scenarios with different numbers of food sources with varying densities across the field. It is plausible that simulating these different conditions would lead to different simulation dynamics, particularly given how dynamic the food chemo-attractant information is across time (see Section 2.1).

Our findings are heavily dependent on many assumptions, many of which we did not formally evaluate. We selected to run our simulations for 1000 timepoints as this balanced computational feasibility with good results in the simulations we tried manually, however no formal checks for evolution of network properties over time were done to check for any convergence behavior. It may well be that networks develop more favorable properties if simulations are conducted over long periods of time. Quantifying these properties over time represents an important immediate extension. Furthermore, we were limited to using approximate methods for calculating our network metrics due to the NP-complete nature of the problems (see Section 2.2). Although the method we used has a formal performance guarantee (Garey and Johnson, 1977), we did not empirically test the performance of this metric on our dataset, and can therefore not speculate anything regarding the absolute values of metrics involving MRST length. On a similar note, although we rely on Tero et al. (2010) for inspiration for our metrics, our modifications to the metrics to make them work on rectilinear Steiner trees are exclusively based on intuition, and not rigorously validated. This is particularly problematic for our metric of fault tolerance, as disconnecting edges in rectilinear Steiner trees has very different effects on food source node connectivity than performing this operation on a traditional graph (see Section 2.2). In particular, our metric of fault tolerance cannot discriminate between "useful" edges (i.e. those connecting food sources, directly or indirectly) and "useless" edges not truly contributing to fault tolerance (e.g. dead ends). We hoped that our metric of cost-effectiveness would be sufficient to differentiate between these cases (where highly fault-tolerant networks with a high cost would indicate many such useless edges being present), however the trends in terms of cost-effectiveness did not allow us to draw any such distinctions. A useful future approach would be to develop a post-processing method for pruning such undesirable edges from networks prior to quantification, though this would only be useful in situations where the only quality of the network is important, as it does nothing to show how *Physarum* is solving the problem.

All previous limitations aside, there are other general methodological decisions which limit the applicability of our findings. The inclusion of walls in our simulations was done out of necessity of quantifying network properties, but this forces the organism to behave in unnatural ways. In the original specification by Wu et al. (2012), *Physarum* explores the field and removes redundant branches by fusing existing branches together. In our specification, the only way for redundant branches to be removed is for the constituting agents to die, which the simulation is not naturally inclined towards. We had to introduce additional rules to get the desired pruning behavior, but doing so likely limits the parallels which can be drawn between our work and biological *Physarum*. Furthermore, it is highly unlikely that our findings translate to networks created by the corresponding simulations when walls are not involved. Formally investigating this in wall-less environments represents an important extension. This might be addressed using a supervised learning approach for extracting networks from agent scenes, though time and resource constraints did not allow us to experiment with this.

5 Conclusion

We present a novel way of exploring and quantifying the properties of transport networks created by simulations of the slime mold *Physarum polycephalum.* We explored the effects of selecting different simulation hyperparameter values, showing that increasing the thresholds of agent elimination and prioritizing the weight agents allocate to other agents' trails when deciding where to move can have beneficial effects on the resulting network properties. In particular, our results suggested that the resulting networks have shorter distances between food source nodes, while also increasing the robustness of the network against localized damage. Other parameters either did not have significant effects on network properties, or led to unfavorable compromises. Although our methodology is occasionally based on formally unvalidated assumptions, we believe this paradigm to represent a valuable starting point in applying particle-based models to transport network creation.

References

Alim, K., Andrew, N., Pringle, A., and Brenner, M. P. (2017). Mechanism of signal propagation in physarum polycephalum. *Proceedings of the National Academy of Sciences*, 114(20):5136–5141.

Chet, I., Naveh, A., and Henis, Y. (1977). Chemotaxis of physarum polycephalum towards carbohydrates, amino acids and nucleotides. *Journal of General Microbiology*, 102(1):145–148.

Garey, M. R. and Johnson, D. S. (1977). The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834.

Jones, J. (2010). Characteristics of pattern formation and evolution in approximations of physarum transport networks. *Artificial Life*, 16(2):127–153.

Mehlhorn, K. (1988). A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters*, 27(3):125–128.

Nakagaki, T., Kobayashi, R., Nishiura, Y., and Ueda, T. (2004a). Obtaining multiple separate food sources: Behavioural intelligence in the physarum plasmodium. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 271(1554):2305–2310.

Nakagaki, T., Yamada, H., and Hara, M. (2004b). Smart network solutions in an amoeboid organism. *Biophysical Chemistry*, 107(1):1–5.

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

Wu, Y., Zhang, Z., Deng, Y., Zhou, H., and Qian, T. (2012). An enhanced multi-agent system with evolution mechanism to approximate physarum transport networks. pages 27–38.

A Work division

We first started with implementing all experiments in Jax, as we were worried about computational feasibility. Bram began this iteration of the implementation, and Victor wrote most of the logic in Wu et al. (2012) in this format as well as getting the underlying graphical representations. Having found Jax to be an obstacle for rapid prototyping, Bram created a corresponding Numpy-based version and quantified the speed differences between it and our initial Jax implementation. We found the speed benefits to not outweigh the downsides, particularly given that the specification of Wu et al. (2012) does not easily lend itself to parallelization on a GPU. Both members contributed to developing the final network extraction pipeline, with Victor implementing most of it. Bram wrote the code for quantifying network properties and displaying corresponding visualizations. Bram conducted all of the experiments, and further developed the user interface, allowing for interactive visualization of scene properties. Victor conducted all statistical testing and wrote the report. Both members contributed equally to the project conception, experimental design, and interpretation of results.