Algorithms in Nature

Image source: https://www.youtube.com/watch?v=VJkJbM3y5R4

Algorithms in nature: not a new idea

Algorithm in nature: an approach for "solving" some problem that is inspired by nature.

Algorithms in nature: not a new idea

Algorithm in nature: an approach for "solving" some problem that is inspired by nature.

STOP: Where have we seen algorithms in nature already in this course?

Algorithms in nature: not a new idea

Algorithm in nature: an approach for "solving" some problem that is inspired by nature.

STOP: Where have we seen algorithms in nature already in this course?

Answer: Three primary answers.

1. Neural networks.
2. "Genetic" algorithms!
3. Bacterial chemotaxis.

ANT FORAGING WITHOUT SPATIAL INFORMATION

Remember our "Be An Ant" Strat for the *Lost Immortals* Hypothetical

If you have no information, walk at random, leaving a trail of stone markers, each one pointing to the next. For every day that you walk, rest for three. Periodically mark the date alongside the cairn. It doesn't matter how you do this, as long as it's consistent. You could chisel the number of days into a rock, or lay out rocks to plot the number.

If you come across a trail that's newer than any you've seen before, start following it as fast as you can. If you lose the trail and can't recover it, resume leaving your own trail.

You don't have to come across the other player's current location; you simply have to come across a location where they've been. You can still chase one another in circles, but as long as you move more quickly when you're following a trail than when you're leaving one, you'll find each other in a matter of years or decades.

And if your partner isn't cooperating—perhaps they're just sitting where they started and waiting for you—then you'll get to see some neat stuff.

A little about ants

Most ants explore using:

1. trails of pheromones;
2. vision.

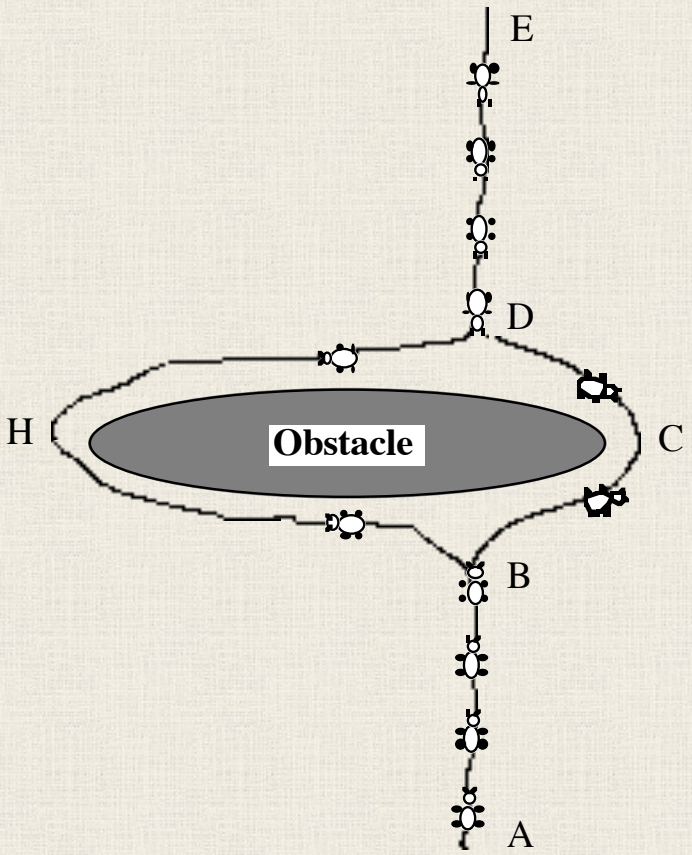
Allelomimesis: One organism performing a behavior makes others more likely to perform it as well.

A little about ants

Allelomimesis: One organism performing a behavior makes others more likely to perform it as well.

Allelomimesis in ants

Ants on the path from *A* to *E* will reach *D* faster if they go through *C*.

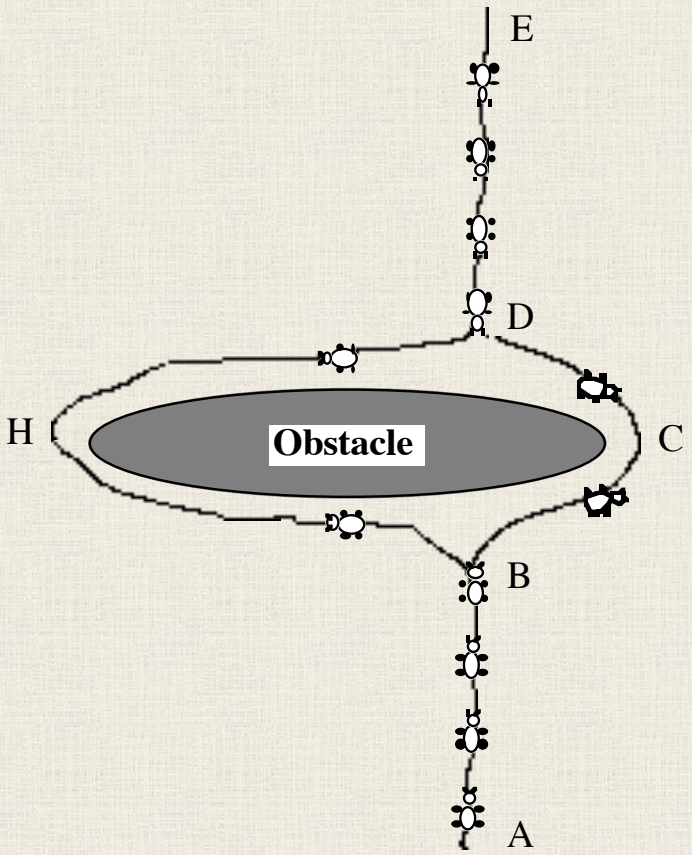


Colorni, Dorigo, Maniezzo 1991

Allelomimesis in ants

Ants on the path from *A* to *E* will reach *D* faster if they go through *C*.

Ants headed from *E* to *D* will find the path through *C* stronger. They are therefore *more likely* to take this path, laying down more pheromone → feedback loop!

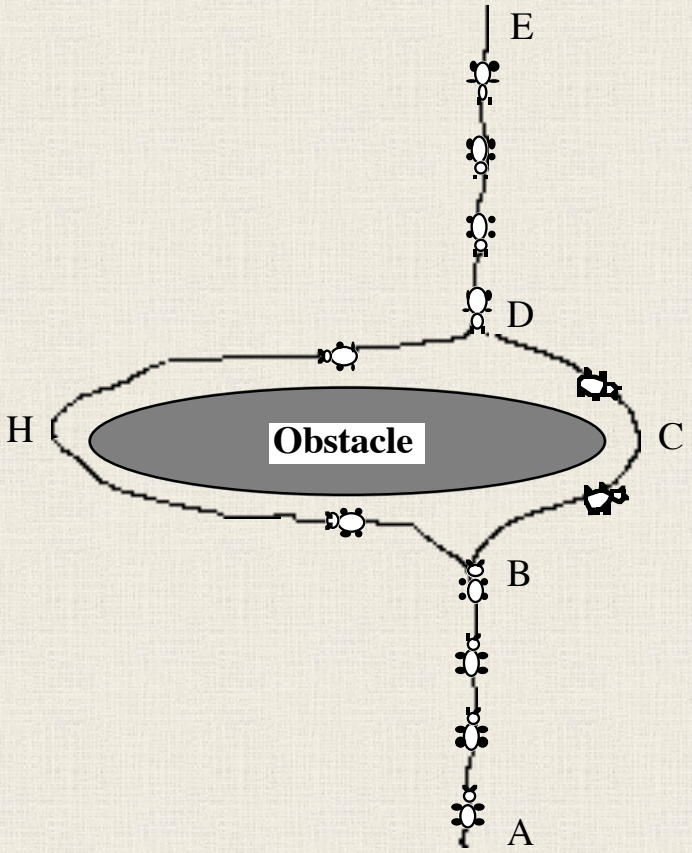


Colorni, Dorigo, Maniezzo 1991

Allelomimesis in ants

STOP: What does this exploration algorithm remind us of?

Ants headed from *E* to *D* will find the path through *C* stronger. They are therefore *more likely* to take this path, laying down more pheromone → feedback loop!

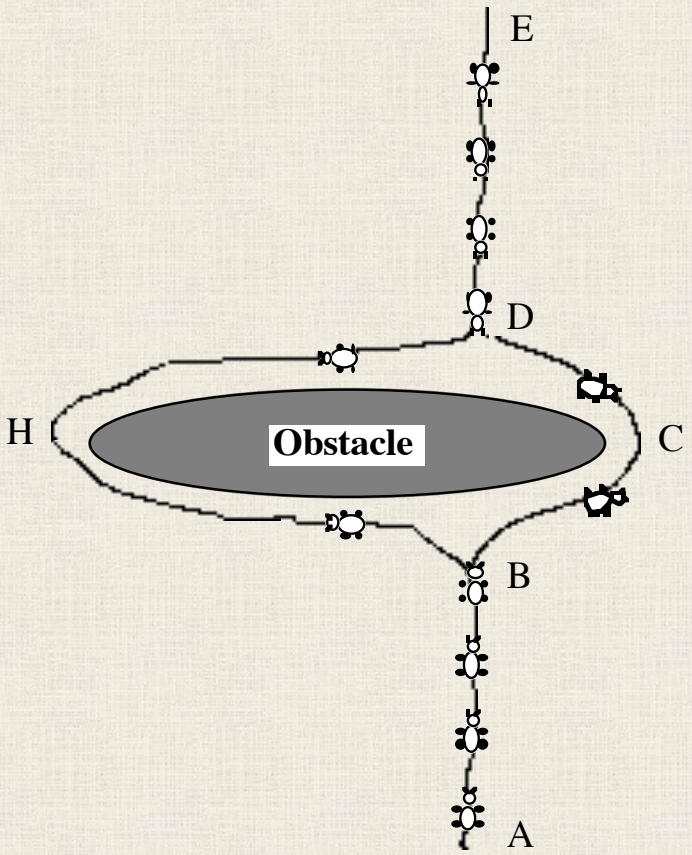


Colorni, Dorigo, Maniezzo 1991

Allelomimesis in ants

STOP: What does this exploration algorithm remind us of?

Answer: Gibbs sampling (motif finding) and finding the best abundance vector (RNA-seq). A small "signal" could be detected and reinforced probabilistically.



Colorni, Dorigo, Maniezzo 1991

Harvester ants don't work this way

Harvester ants (*P. barbatus*) don't use pheromones. They search for seeds in the desert, a sparse environment, and the ants typically scatter to do so.

Image courtesy: Bob Peterson

Harvester ants don't work this way

Harvester ants (*P. barbatus*) don't use pheromones. They search for seeds in the desert, a sparse environment, and the ants typically scatter to do so.

Their exploration is more individualized. So how interesting could their algorithm be?

Image courtesy: Bob Peterson

Exploration is a function of arriving ants

The more food that is currently available, the more the ants should want to explore, since exploring a barren waste would lead to a quick death.

Exploration is a function of arriving ants

The more food that is currently available, the more the ants should want to explore, since exploring a barren waste would lead to a quick death.

Key insight: The ants use the frequency of *arriving* ants to determine whether to explore further.

[HTML] The regulation of ant colony foraging activity without spatial information

B Prabhakar, KN Dektar, DM Gordon - PLoS Comput Biol, 2012 - journals.plos.org

Many dynamical networks, such as the ones that produce the collective behavior of social insects, operate without any central control, instead arising from local interactions among individuals. A well-studied example is the formation of recruitment trails in ant colonies, but many ant species do not use pheromone trails. We present a model of the regulation of foraging by harvester ant (Pogonomyrmex barbatus) colonies. This species forages for scattered seeds that one ant can retrieve on its own, so there is no need for spatial …

☆ 99 Cited by 94 Related articles All 16 versions »

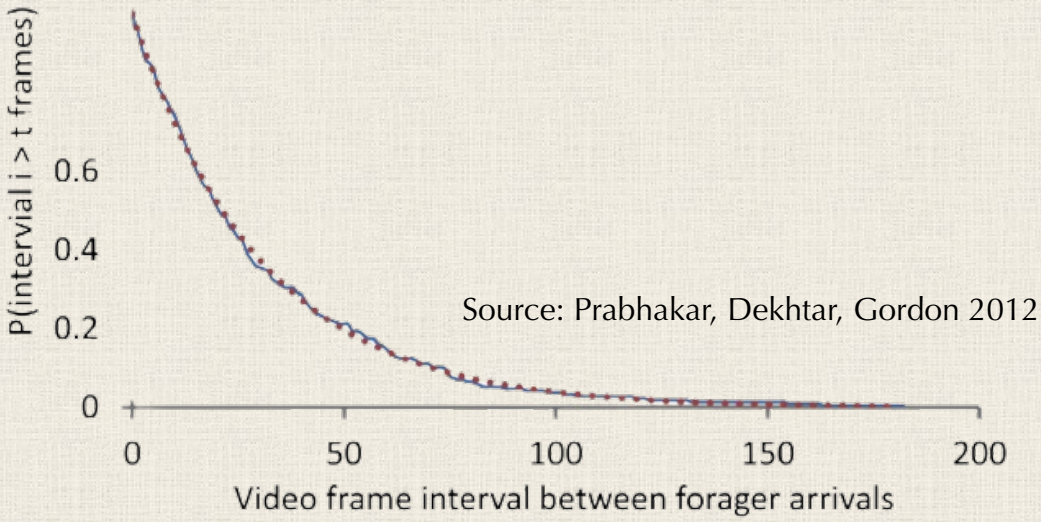
Modeling ant arrivals

STOP: Say that we want to model arriving ants. Any ideas based on (hint hint) what we already know?

Modeling ant arrivals

STOP: Say that we want to model arriving ants. Any ideas based on (hint hint) what we already know?

Answer: Let's use a Poisson distribution! Indeed, the times between real ant arrivals are exponential. ☺



Source: Prabhakar, Dekhtar, Gordon 2012

Modeling ant *departures*

STOP: If this is how ants arrive, how could we model the number of departing ants in a given time interval?

Modeling ant *departures*

STOP: If this is how ants arrive, how could we model the number of departing ants in a given time interval?

Answer: One solution would be to use a Poisson distribution, but with the caveat that the mean should vary based on the number of arrivals.

Modeling ant *departures*

Let D_n be the number of predicted departures in interval n, and let A_n denote the number of arrivals.

Modeling ant *departures*

Let D_n be the number of predicted departures in interval n, and let A_n denote the number of arrivals.

D_n should be a Poisson variable with mean $f\alpha_n$, where α_n depends on A_n in the following way.

$$\alpha_n = \max(cA_n, \alpha')$$

where c is a constant and α' is a "baseline" that will prevent the departure from becoming zero.

The paper's model is a little more complicated

To allow the signal to last more than one interval, α_n should depend on α_{n-1} minus some constant d.

The paper's model is a little more complicated

To allow the signal to last more than one interval, α_n should depend on α_{n-1} minus some constant d.

Finally, the entrance can get crowded, so α_n should vary inversely with D_{n-1}.

The paper's model is a little more complicated

To allow the signal to last more than one interval, α_n should depend on α_{n-1} minus some constant d.

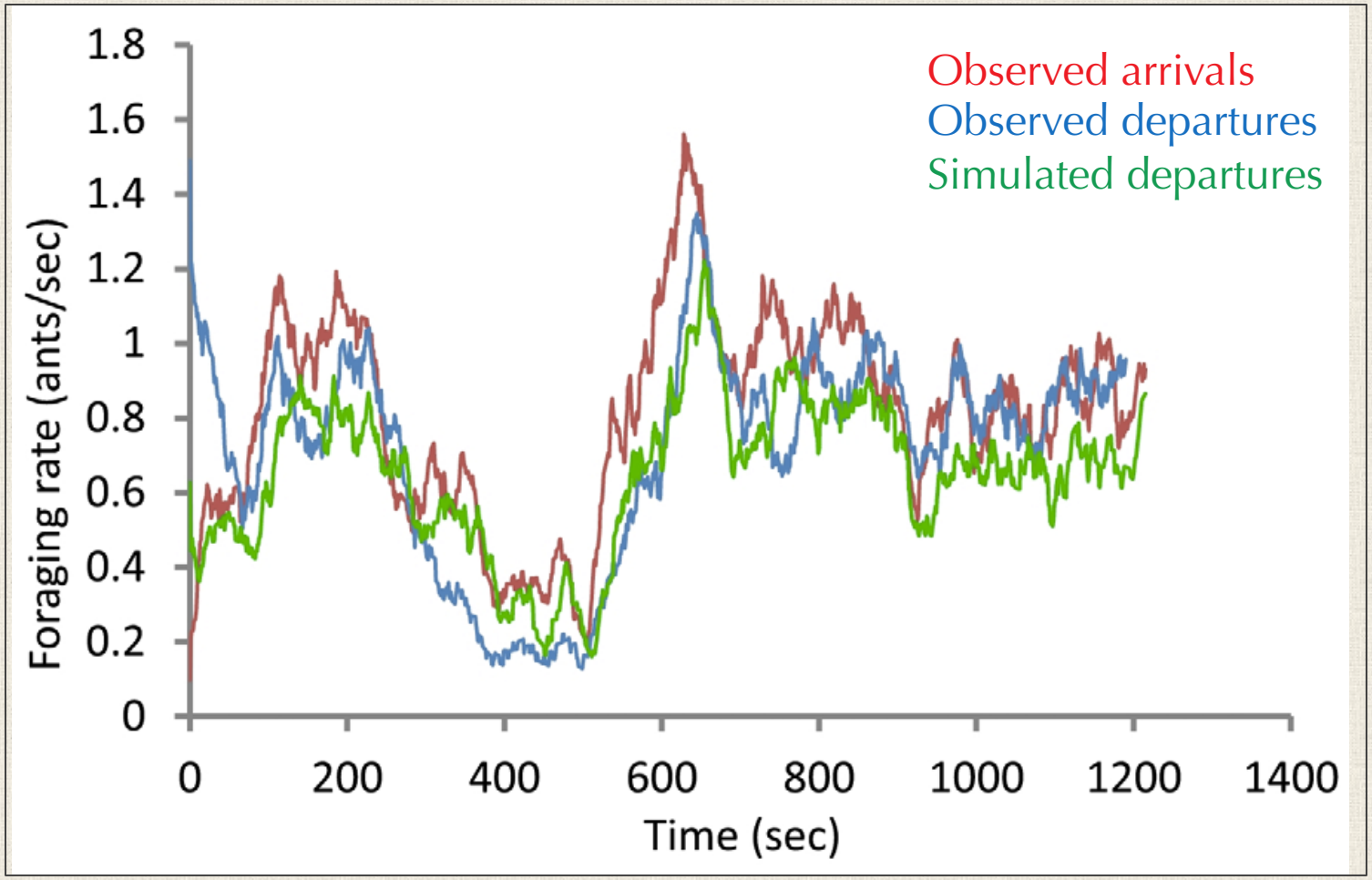
Finally, the entrance can get crowded, so α_n should vary inversely with D_{n-1}.

D_n should be a Poisson variable with mean $f\alpha_n$, where α_n depends on A_n in the following way.

$$\alpha_n = \max(\alpha_{n-1} - d - qD_{n-1} + cA_n, \alpha')$$

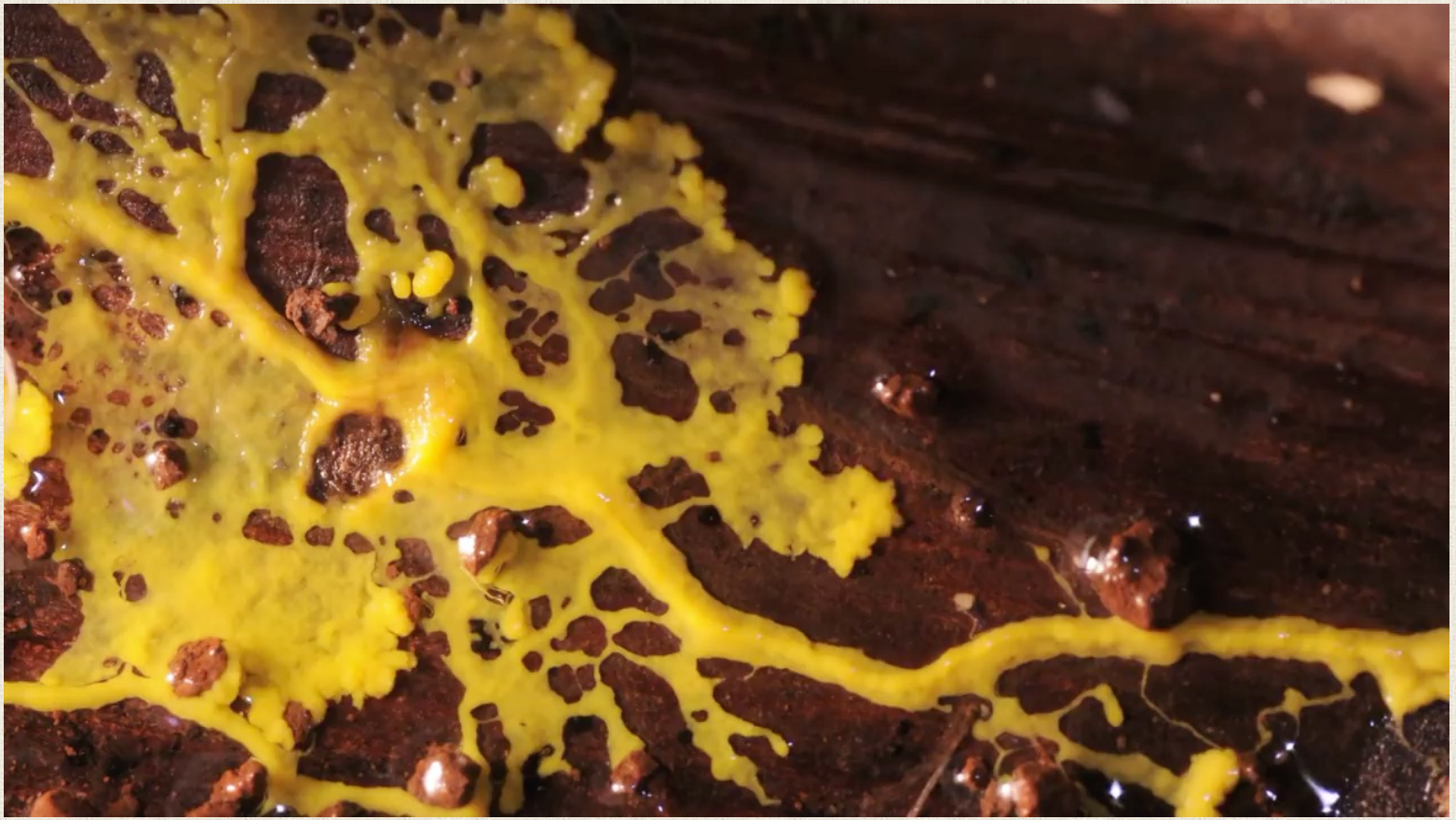
where c is a constant and α' is a "baseline" that will prevent the departure from becoming zero.

For right parameters, observed/predicted departure rates are correlated



SLIME MOLDS AND TRANSPORTATION NETWORKS

Slime molds are foragers too



https://www.youtube.com/watch?v=VJkJbM3y5R4

Goal: study how slime molds form networks.

Toward a computational problem

When the slime mold encounters a food source, the food becomes a "hub" for distributing resources.

Toward a computational problem

When the slime mold encounters a food source, the food becomes a "hub" for distributing resources.

The mold reinforces pathways when there is a lot of food and trimming away connections where there isn't any. Can we model its heuristic?

Toward a computational problem

When the slime mold encounters a food source, the food becomes a "hub" for distributing resources.

The mold reinforces pathways when there is a lot of food and trimming away connections where there isn't any. Can we model its heuristic?

STOP: What computational problem do you think that the mold is solving? What is it optimizing?

Steiner trees model connected networks of minimal transportation cost

A **Steiner tree** of a collection of points is a tree including the points as nodes and minimizing the sum of edge lengths in the tree.

Steiner Tree Problem:

- **Input:** A collection of *n* points in 2-D space.
- **Output:** A Steiner tree containing the *n* points.

Steiner trees model connected networks of minimal transportation cost

A **Steiner tree** of a collection of points is a tree including the points as nodes and minimizing the sum of edge lengths in the tree.

Steiner Tree Problem:

- **Input:** A collection of *n* points in 2-D space.
- **Output:** A Steiner tree containing the *n* points.

STOP: Why are we looking for a tree?

Steiner trees model connected networks of minimal transportation cost

A **Steiner tree** of a collection of points is a tree including the points as nodes and minimizing the sum of edge lengths in the tree.

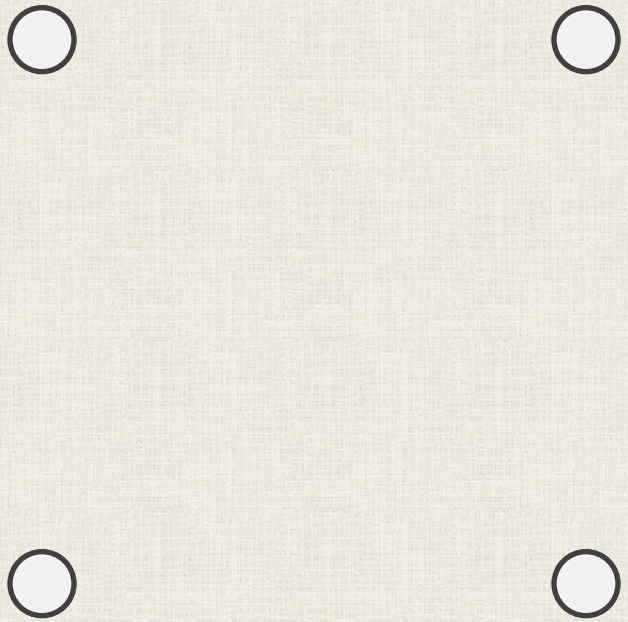
Steiner Tree Problem:

- **Input:** A collection of *n* points in 2-D space.
- **Output:** A Steiner tree containing the *n* points.

Answer: If we have a cycle, then removing an edge produces a connected network with less "cost".

Let's do an example

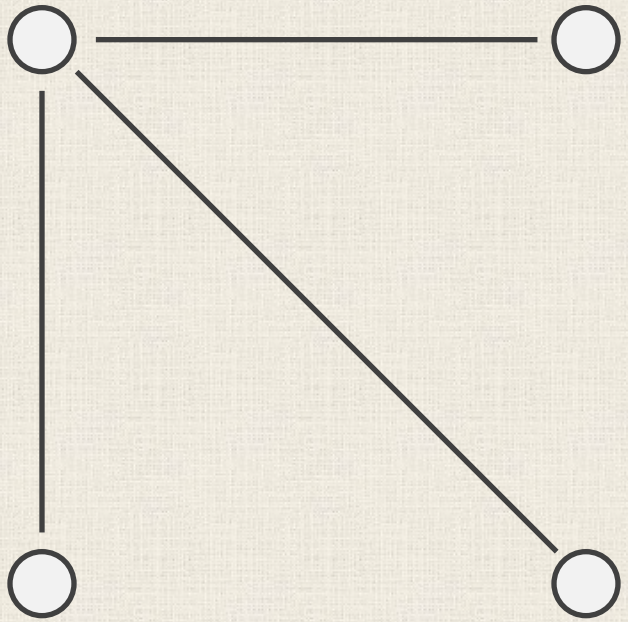
Exercise: Say that our points are the four corners of a unit square. Find a Steiner tree of these points.



Let's do an example

Exercise: Say that our points are the four corners of a unit square. Find a Steiner tree of these points.

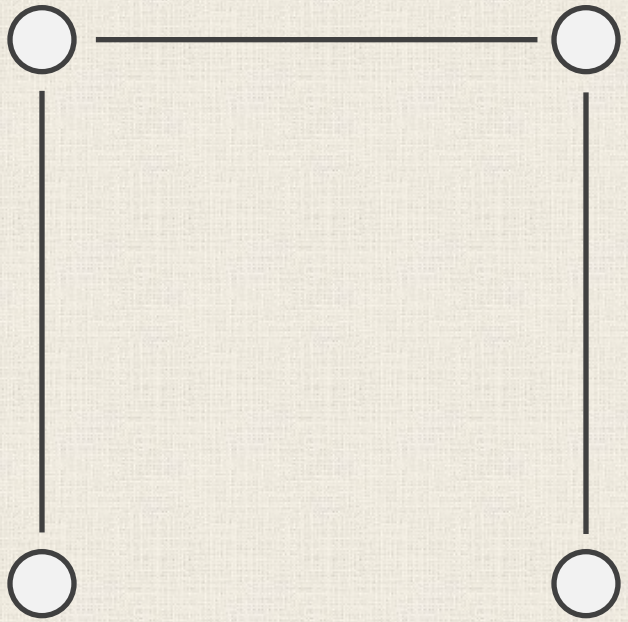
Total distance: $2 + \sqrt{2}$



Let's do an example

Exercise: Say that our points are the four corners of a unit square. Find a Steiner tree of these points.

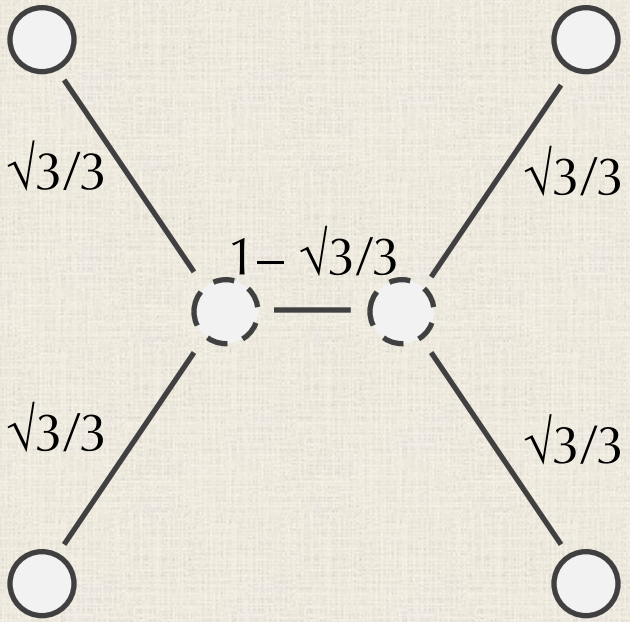
Total distance: 3



Let's do an example

Answer: The correct answer in this case adds two new nodes that do not occur in the original set.

Total distance: $1 + \sqrt{3}$



Finding a Steiner tree is hard

A **Steiner tree** of a collection of points is a tree including the points as nodes and minimizing the sum of edge lengths in the tree.

Steiner Tree Problem:

- **Input:** A collection of *n* points in 2-D space.
- **Output:** A Steiner tree containing the *n* points.

Note: As you might expect, this problem is *NP*-Complete. But slime molds don't know this!

Finding a Steiner tree is hard

A **Steiner tree** of a collection of points is a tree including the points as nodes and minimizing the sum of edge lengths in the tree.

Steiner Tree Problem:

- **Input:** A collection of *n* points in 2-D space.
- **Output:** A Steiner tree containing the *n* points.

Also, this isn't an ideal problem; edges have varying thicknesses (i.e., weights), and the network needs to be somewhat fault tolerant (so a tree isn't the best).

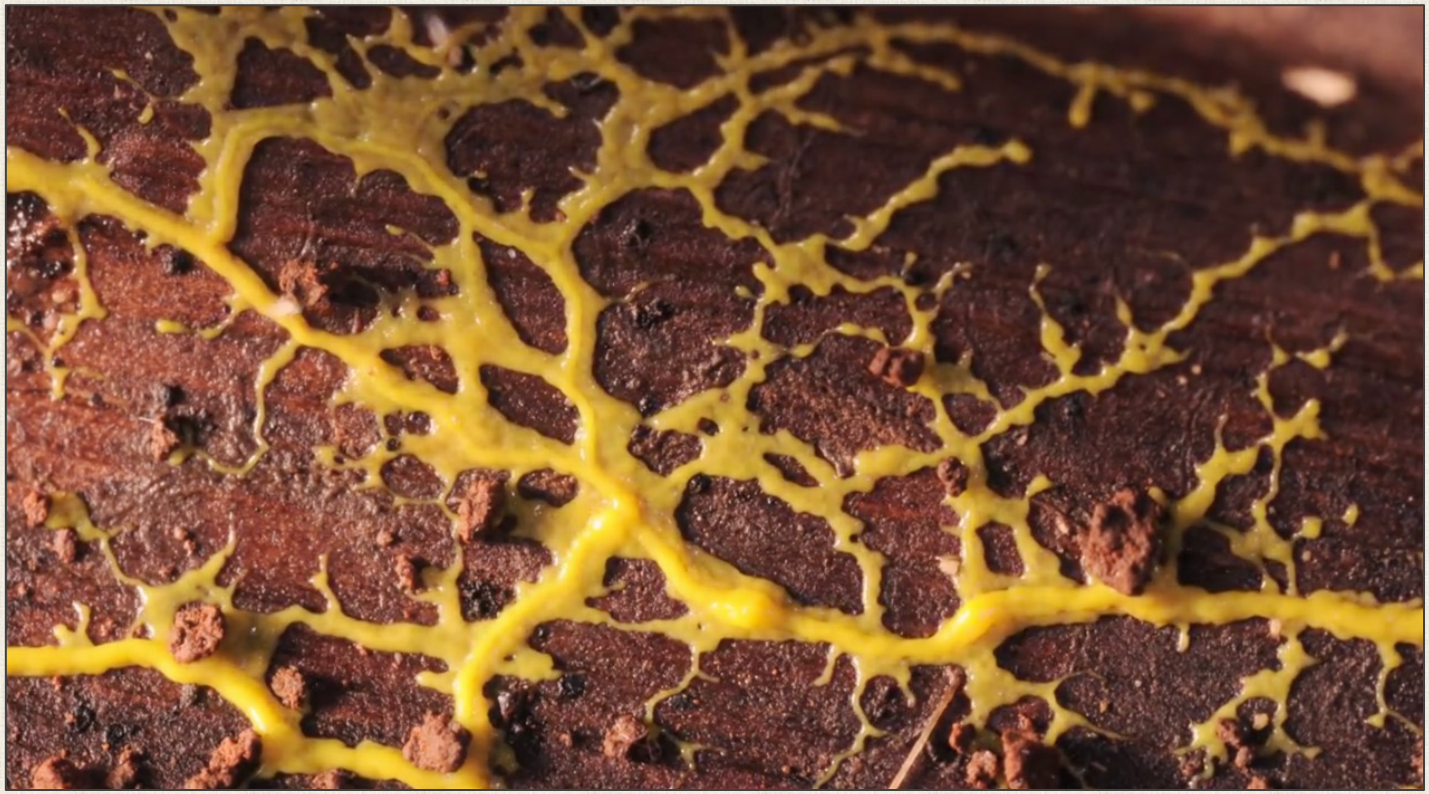
How do slime molds explore?

The mold starts as a thinly meshed lattice with many thin edges to explore an area completely.

How do slime molds explore?

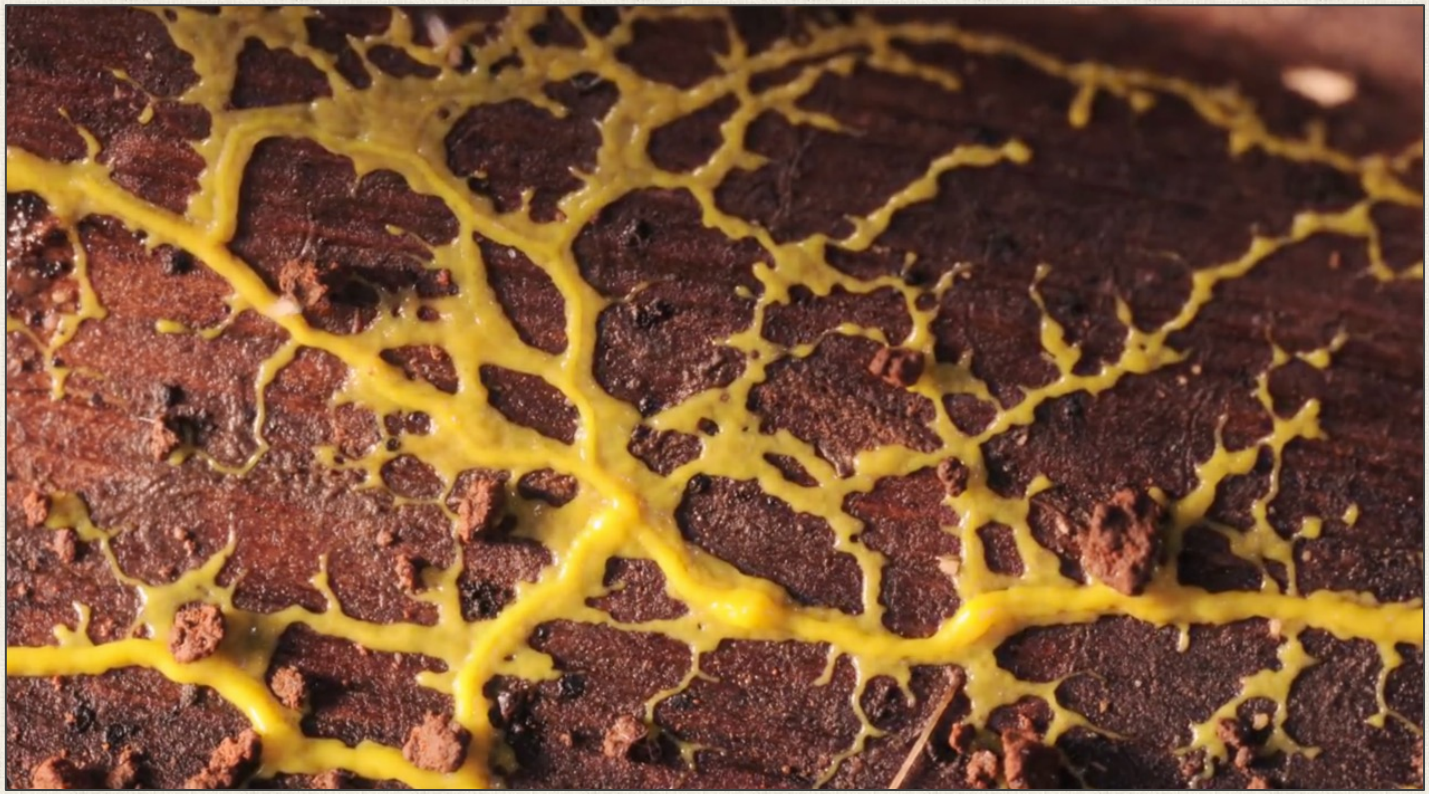
As the slime mold grows, it reinforces successful tubes (i.e., ones with lots of flow per unit length).

How do slime molds explore?



The slime mold also allows tubes that have not been successful to die out, which produces the network.

How do slime molds explore?



Note: Using feedback to make small changes should remind us of some things…

Fun idea: grow "Tokyo" slime molds and compare against real rail network

Rules for biologically inspired adaptive network design

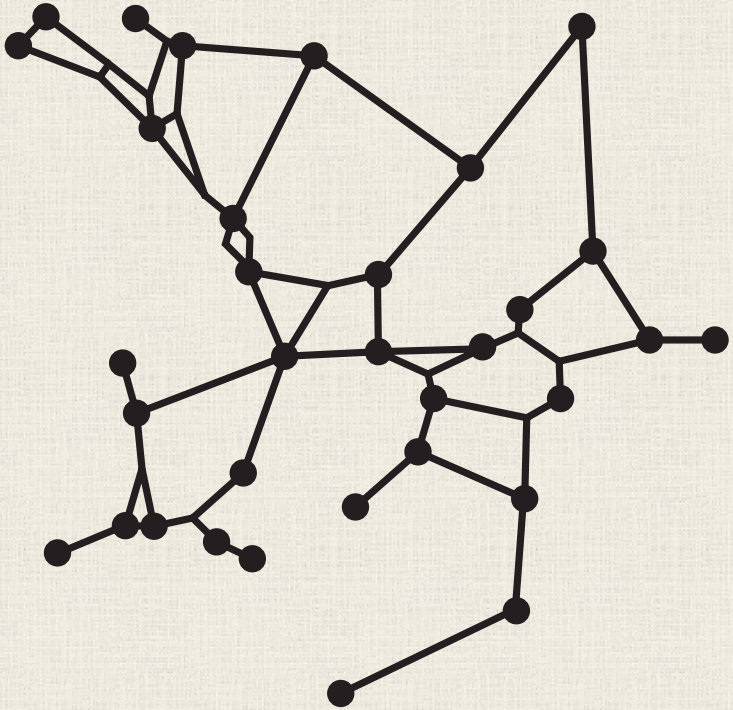
A Tero, S Takagi, T Saigusa, K Ito, DP Bebber… - …, 2010 - science.sciencemag.org

Transport networks are ubiquitous in both social and biological systems. Robust network performance involves a complex trade-off involving cost, transport efficiency, and fault tolerance. Biological networks have been honed by many cycles of evolutionary selection pressure and are likely to yield reasonable solutions to such combinatorial optimization problems. Furthermore, they develop without centralized control and may represent a readily scalable solution for growing networks in general. We show that the slime mold Physarum …

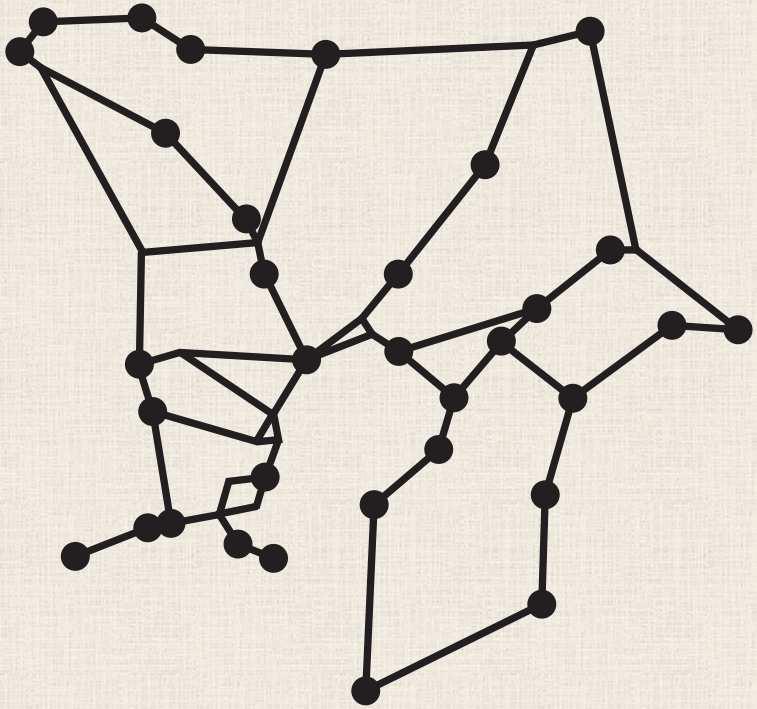
☆ 99 Cited by 847 Related articles All 36 versions

Researchers placed food at locations corresponding to cities around Tokyo and used variable light to simulate geography.

Slime mold network is very similar to real Tokyo rail network



Slime mold network



Tokyo rail network

Next, let's mimic the slime mold's algorithm

Let $Q_{i,j}$ denote the **flux** through the tube connecting node i to j, the volume moving through a fixed part of the tube per unit time. Physics tells us that $Q_{i,j}$ is:

Next, let's mimic the slime mold's algorithm

Let $Q_{i,j}$ denote the **flux** through the tube connecting node i to j, the volume moving through a fixed part of the tube per unit time. Physics tells us that $Q_{i,j}$ is:

- directly proportional to the thickness of the tube;

Next, let's mimic the slime mold's algorithm

Let $Q_{i,j}$ denote the **flux** through the tube connecting node i to j, the volume moving through a fixed part of the tube per unit time. Physics tells us that $Q_{i,j}$ is:

- directly proportional to the thickness of the tube;
- directly proportional to the pressure differential between nodes i and j;

Next, let's mimic the slime mold's algorithm

Let $Q_{i,j}$ denote the **flux** through the tube connecting node *i* to *j*, the volume moving through a fixed part of the tube per unit time. Physics tells us that $Q_{i,j}$ is:

- directly proportional to the thickness of the tube;
- directly proportional to the pressure differential between nodes *i* and *j*;
- inversely proportional to the length of the tube.

Next, let's mimic the slime mold's algorithm

Putting this together gives us that the flux $Q_{i,j}$ is

$$Q_{i,j} = D_{i,j} (p_i - p_j) / L_{i,j}$$

- $D_{i,j}$ is the thickness of the tube;
- p_i is the pressure at node i;
- $L_{i,j}$ is the length of the tube.

Next, let's mimic the slime mold's algorithm

1. Build a finely meshed lattice covering the space.
2. Repeat the following steps *n* times:
   i. Pick two random nodes as source and sink.
   ii. "Push" the source's total outgoing flux toward sink, setting pressure values to ensure that the incoming/outcoming flux balance at every other node. (This is a linear algebra problem.)
   iii. Update $D_{i,j}$ along every edge to reinforce high-flux edges and allow low-flux edges to decay.

Updating the thickness $D_{i,j}$

To update $D_{i,j}$ after we have updated the $Q_{i,j}$, we use the following differential equation:

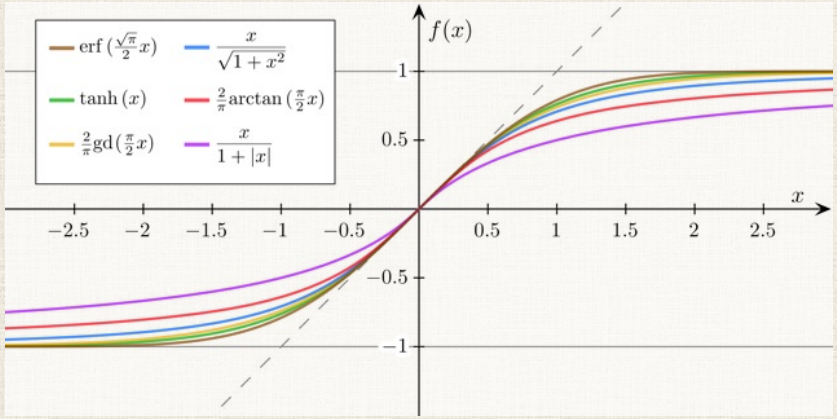
$$\mathrm{d}D_{i,j}/\mathrm{d}t = f(|Q_{i,j}|) - D_{i,j}$$

Updating the thickness $D_{i,j}$

To update $D_{i,j}$ after we have updated the $Q_{i,j}$, we use the following differential equation:

$$\mathrm{d}D_{i,j}/\mathrm{d}t = f(|Q_{i,j}|) - D_{i,j}$$

The function should be an increasing function, so that the tube will become thicker if we have more flux. One choice is a sigmoid function.



https://en.wikipedia.org/wiki/Sigmoid_function

Updating the thickness $D_{i,j}$

To update $D_{i,j}$ after we have updated the $Q_{i,j}$, we use the following differential equation:

$$\mathrm{d}D_{i,j}/\mathrm{d}t = f(|Q_{i,j}|) - D_{i,j}$$

STOP: What is the purpose of subtracting the $D_{i,j}$ term?

Updating the thickness $D_{i,j}$

To update $D_{i,j}$ after we have updated the $Q_{i,j}$, we use the following differential equation:

$$\mathrm{d}D_{i,j}/\mathrm{d}t = f(|Q_{i,j}|) - D_{i,j}$$

STOP: What is the purpose of subtracting the $D_{i,j}$ term?

Answer: If the flux is low, then we want the tube to shrink in the next step.

Measuring network quality

Note: There are two (conflicting) things we want:

1. Small network size.
2. "Fault tolerant" network robust to edge removals.

Measuring network quality

Note: There are two (conflicting) things we want:

1. Small network size.
2. "Fault tolerant" network robust to edge removals.

STOP: How could we quantify each of these properties?

Measuring network quality

Note: There are two (conflicting) things we want:

1. Small network size.
2. "Fault tolerant" network robust to edge removals.

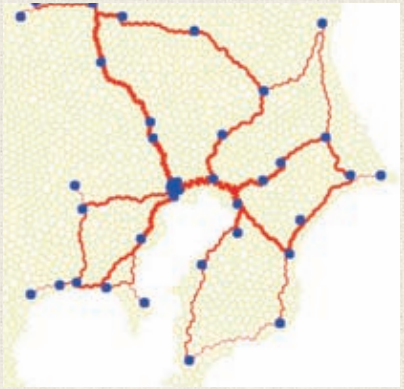
STOP: How could we quantify each of these properties?

Answer: Two simple (but good) ideas:

1. Total distance of all edges in network.
2. Chance that edge removal disconnects network.

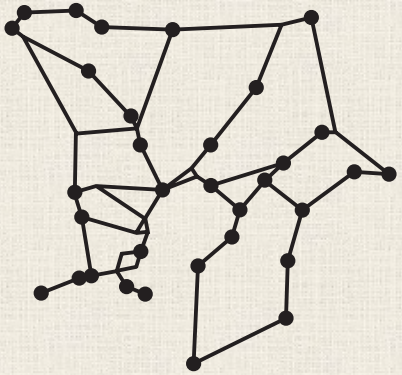
Comparing simulated network against real and slime mold networks

Simulated networks are more efficient but less fault tolerant than both the real rail network and slime mold network.



Simulated network

Slime mold network



Tokyo rail network

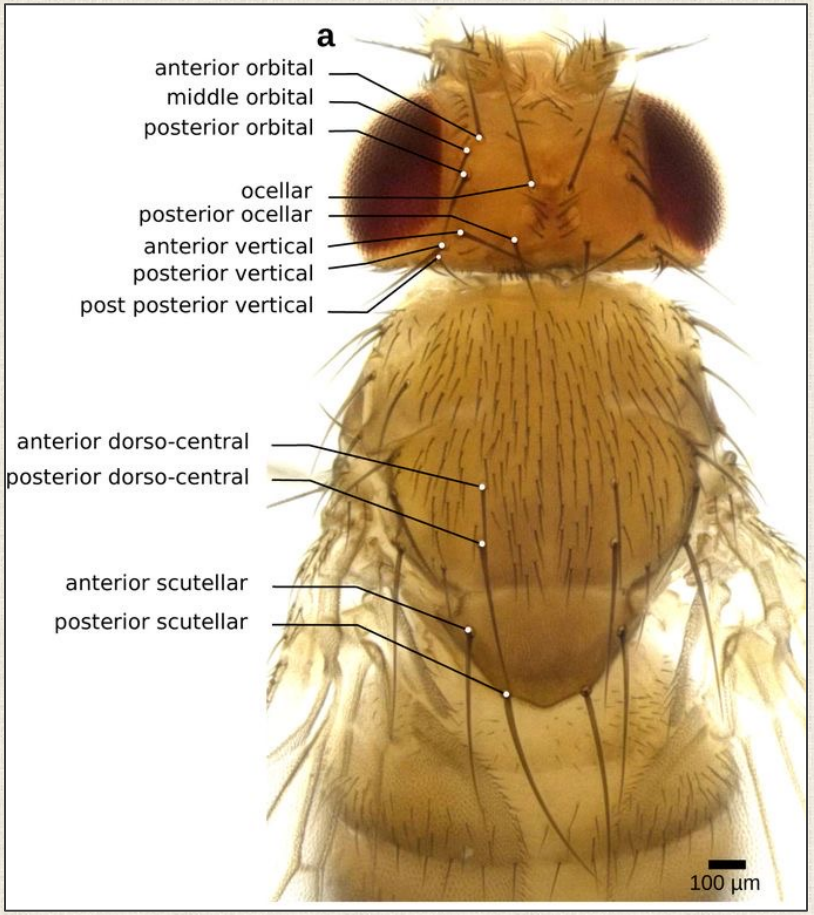
STOP: Why do you think this might be?

CHOOSING SENSORY ORGAN PRECURSOR CELLS IN DROSOPHILA

Development of *Drosophila* bristles

Drosophila melanogaster is covered in sensory bristles.

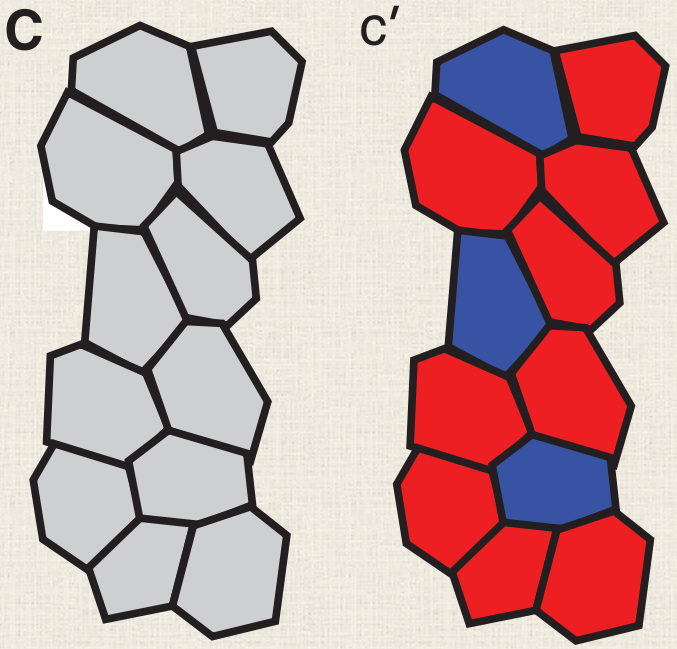
During development, the fly has clusters of cells, some of which become **sensory organ precursors (SOPs)**, which become bristles.



https://www.biorxiv.org/content/10.1101/295485v1

Activating SOPs

When an SOP starts to appear, it expresses a membrane-bound protein Delta, which inhibits its neighboring cells. As a result, we cannot have neighboring SOPs.

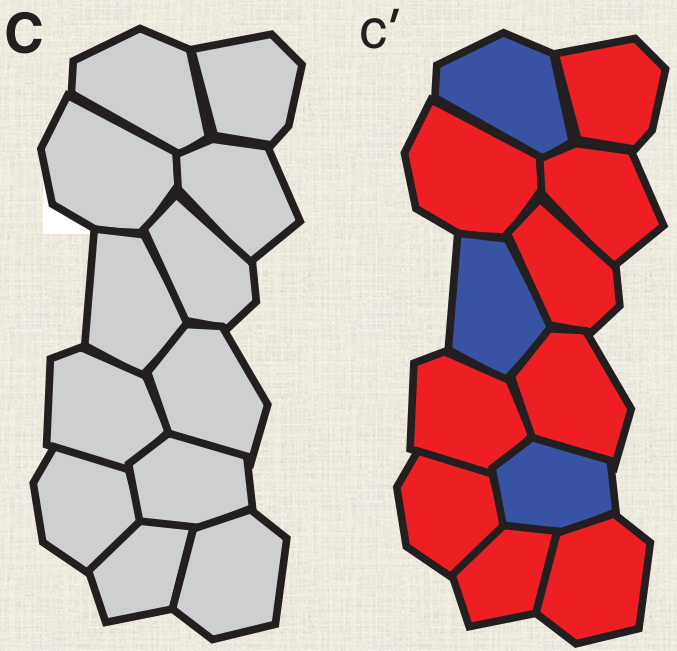


https://science.sciencemag.org/content/331/6014/183

At right is a hypothetical coloring of SOPs (blue).

Activating SOPs

STOP: Thinking in terms of evolution, given a cluster of cells, what problem is the fly optimizing?

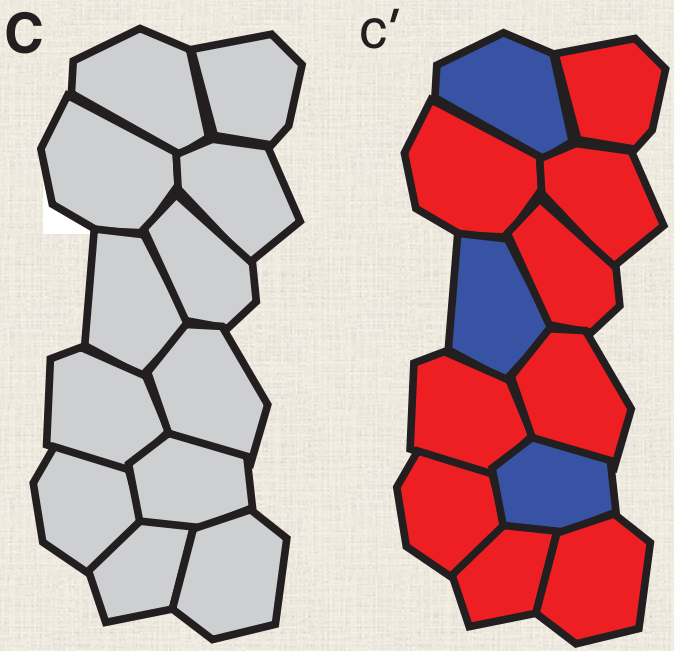


https://science.sciencemag.org/content/331/6014/183

Activating SOPs

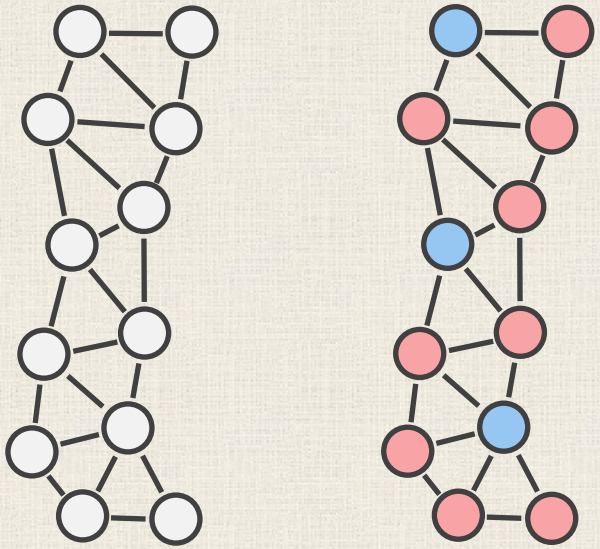
STOP: Thinking in terms of evolution, given a cluster of cells, what problem is the fly optimizing?

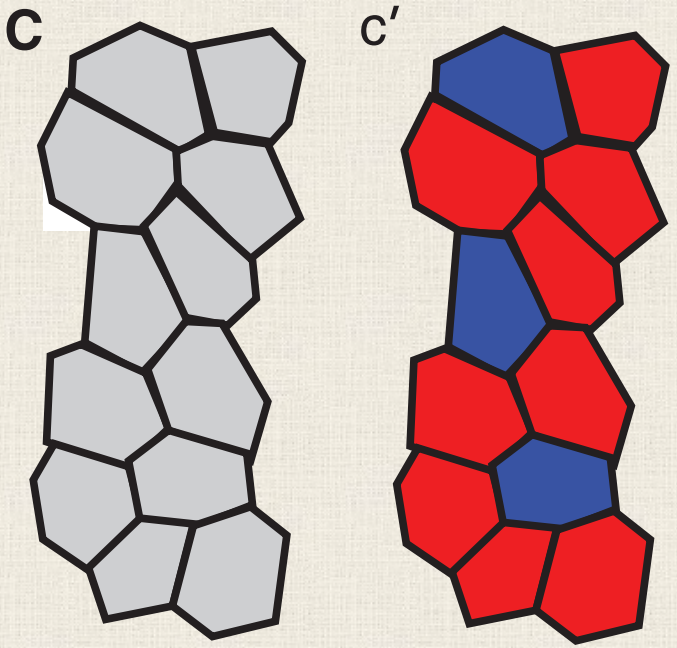
Answer: Turn "on" as many SOPs possible, subject to the constraint that neighbors cannot both be SOPs …



https://science.sciencemag.org/content/331/6014/183

Framing our biological problem as a network problem



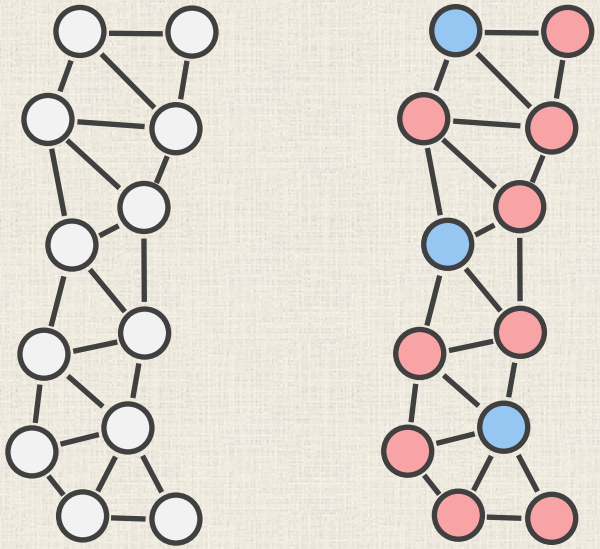


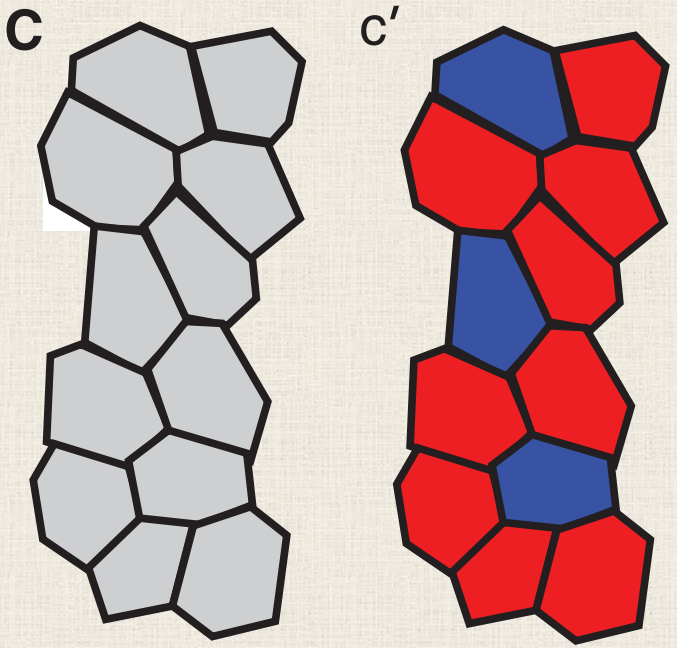
https://science.sciencemag.org/content/331/6014/183

Nodes = cells; edges = adjacent cells.

Nodes are colored blue if an SOP, red if not.

Framing our biological problem as a network problem





https://science.sciencemag.org/content/331/6014/183

Exercise: Formulate a network problem corresponding to finding as many SOPs as possible.

This network problem is not new

A set of nodes in a graph is an **independent set** if there are no edges connecting two nodes in the set.

Maximum Independent Set Problem:

- **Input:** A graph.
- **Output:** An independent set maximizing the number of nodes over all independent sets.

This network problem is not new

A set of nodes in a graph is an **independent set** if there are no edges connecting two nodes in the set.

Maximum Independent Set Problem:

- **Input:** A graph.
- **Output:** An independent set maximizing the number of nodes over all independent sets.

This problem is *NP*-hard, and it's fair to assume that *Drosophila* is not solving it to pick SOPs…

This network problem is not new

A set of nodes in a graph is an **independent set** if there are no edges connecting two nodes in the set.

Maximal Independent Set Problem:

- **Input:** A graph.
- **Output:** An independent set that is not a subset of another independent set.

Note: The problem becomes tractable if we change "maximum" to "maximal".

This network problem is not new

A set of nodes in a graph is an **independent set** if there are no edges connecting two nodes in the set.

Maximal Independent Set Problem:

- **Input:** A graph.
- **Output:** An independent set that is not a subset of another independent set.

STOP: How can we solve this problem?

A sequential optimal solution to this problem is "trivial"…

MIS(G)

    $S \leftarrow$ empty set
    while there are still nodes in G
        select an arbitrary node v in G
        append v to S
        remove v and its neighbors from G
    return S

A sequential optimal solution to this problem is "trivial"…

MIS(G)

$S \leftarrow$ empty set

while there are still nodes in G

select an arbitrary node v in G

append v to S

remove v and its neighbors from G

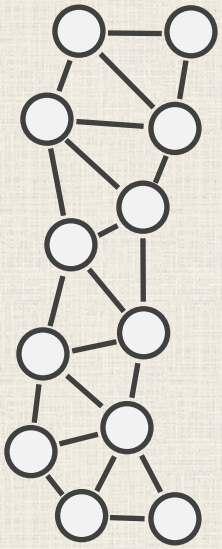
return S

But nature solves this problem in a *distributed* manner in which individual cells have limited info.

A lot of lab work was done to identify the following algorithm

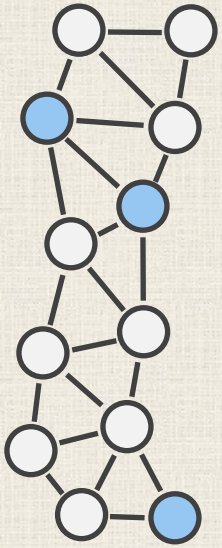
Each node v starts as OFF. Over *n* steps:

- *v* turns ON with probability *p*, which increases over time.
- If *v* is ON, then it broadcasts this (1-bit) message to its neighbors.
- If *v* is ON:
  - If *v* did not receive a message, then *v* becomes an SOP, broadcasts to its neighbors, and the algorithm halts for *v*.
  - Else, turn *v* OFF and continue.
- If *v* is OFF and received an ON message in second broadcast, then *v* remains OFF permanently, and the algorithm halts.



An illustrated example

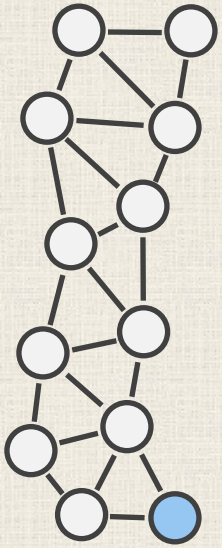
First, turn ON some number of the cells currently under consideration.



An illustrated example

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

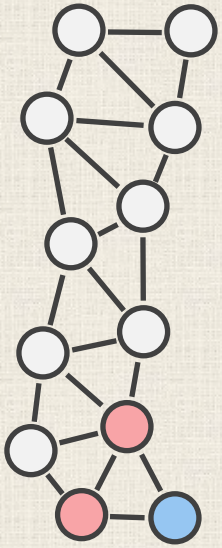


An illustrated example

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

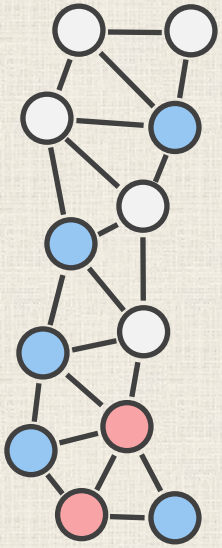


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

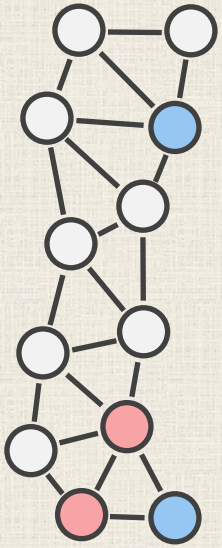


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

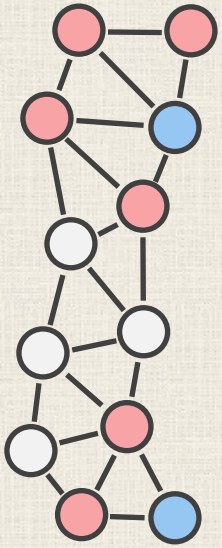


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

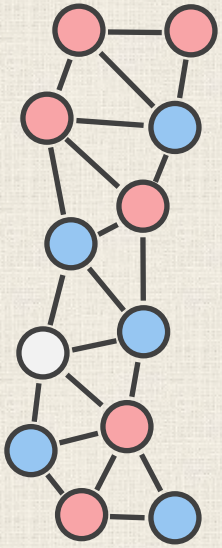


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

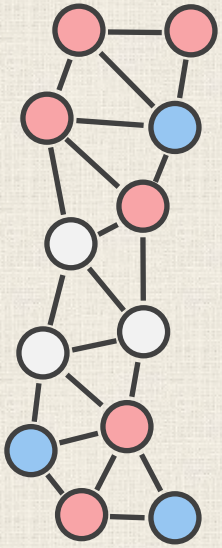


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

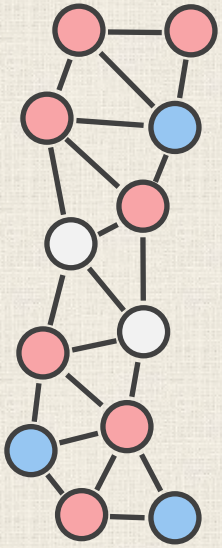


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

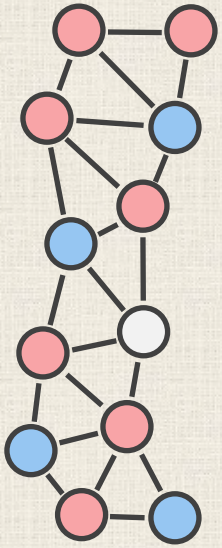


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

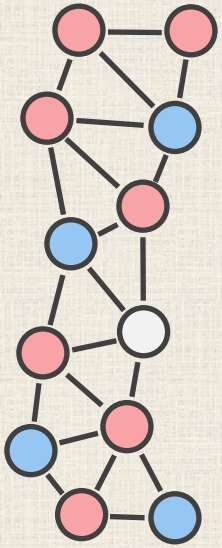


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.

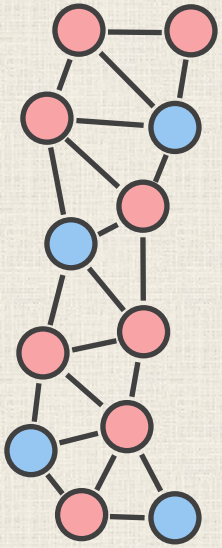


Now we simply repeat these steps

First, turn ON some number of the cells currently under consideration.

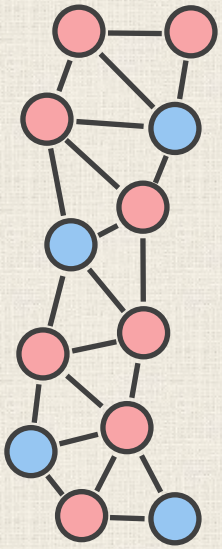
Then, each cell broadcasts, so that if two ON neighbors receive a broadcast, then they turn back OFF.

Surviving ON cells become SOPs and broadcast to neighbors, which turn OFF permanently and are no longer considered.



Looking to nature reveals elegant solutions outside normal CS thinking

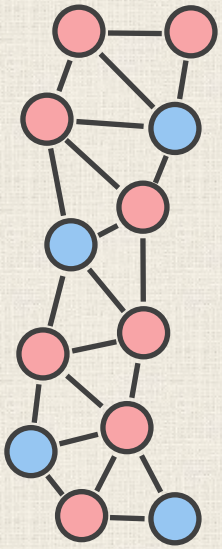
Key point: This is a *new* solution to an *old*, well-researched problem in CS, which can inform non-biological applications (e.g., mesh Wi-Fi).



A biological solution to a fundamental distributed computing problem
Y Afek, N Alon, O Barad, E Hornstein, N Barkai… - …, 2011 - science.sciencemag.org
Computational and biological systems are often distributed so that processors (cells) jointly solve a task, without any of them receiving all inputs or observing all outputs. Maximal independent set (MIS) selection is a fundamental distributed computing procedure that seeks to elect a set of local leaders in a network. A variant of this problem is solved during the development of the fly's nervous system, when sensory organ precursor (SOP) cells are chosen. By studying SOP selection, we derived a fast algorithm for MIS selection that …
☆ 99 Cited by 151 Related articles All 33 versions

Looking to nature reveals elegant solutions outside normal CS thinking

Key point: This is a *new* solution to an *old*, well-researched problem in CS, which can inform non-biological applications (e.g., mesh Wi-Fi).



Classic CS view:

- all global info is known.
- communication is free.
- problems solved exactly.

Nature's view:

- limited local information.
- communication is simple.
- heuristics and probability rule!

NOVEL SMELL DETECTION IN DROSOPHILA

Detecting New Objects

"Is this image / smell / taste / sound new?"

This is the same question that many database search algorithms ask.

Detecting New Objects

"Is this image / smell / taste / sound new?"

And both algorithms and our brains can be fooled …

Courtesy: Michael Rivera

The Novel Query Problem

Novel Query Problem:

- **Input:** a set of n objects S and an object x.
- **Output:** "Yes" if x is in S, and "no" otherwise.

STOP: How could we solve this problem?

The Novel Query Problem

Novel Query Problem:

- **Input:** a set of n objects S and an object x.
- **Output:** "Yes" if x is in S, and "no" otherwise.

STOP: How could we solve this problem?

Answer: Maintain S as a sorted list, and given x, perform binary search (as we did with the suffix array). Runtime: $O(\log(|S|))$. We're done!

The Novel Query Problem

Novel Query Problem:

- **Input:** a set of n objects S and an object x.
- **Output:** "Yes" if x is in S, and "no" otherwise.

Yet we *know* that this is not how our brains work (memories fade). S can simply be too large to store.

Answer: Maintain S as a sorted list, and given x, perform binary search (as we did with the suffix array). Runtime: $O(\log(|S|))$. We're done!

The Novel Query Problem

Novel Query Problem:

- **Input:** a set of n objects S and an object x.
- **Output:** "Yes" if x is in S, and "no" otherwise.

We will use a **Bloom filter**, a database of $m > |S|$ bits used to detect membership in S heuristically.

0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

For a given s in S, we set value at index i of the Bloom filter to 0 if there is some j with $h_j(s) = i$.

0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

For a given s in S, we set value at index i of the Bloom filter to 0 if there is some j with $h_j(s) = i$.

h_1("wet dog")=8; h_2("wet dog")=1; h_3("wet dog")=6

0	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	0	1	0	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

For a given s in S, we set value at index i of the Bloom filter to 0 if there is some j with $h_j(s) = i$.

$h_1(\text{“fish”}) = 4$;

0	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	0	1	0	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

For a given s in S, we set value at index i of the Bloom filter to 0 if there is some j with $h_j(s) = i$.

$h_1(\text{“fish”}) = 4$; $h_2(\text{“fish”}) = 3$;

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Bloom filters rely on hash functions

For a small value k, we have k **hash functions** that assign members of S to $\{0, 1, 2, \ldots, m - 1\}$.

For a given s in S, we set value at index i of the Bloom filter to 0 if there is some j with $h_j(s) = i$.

$h_1(\text{“fish”}) = 4$; $h_2(\text{“fish”}) = 3$; $h_3(\text{“fish”}) = 1$

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

Using a Bloom filter to determine novelty

STOP: How did we know that “fish” was a new smell?

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

Using a Bloom filter to determine novelty

STOP: How did we know that “fish” was a new smell?

Answer: One of the hash functions produced a value that was equal to 1 in the hash table.

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

Using a Bloom filter to determine novelty

STOP: What happens if we have the new smell h_1("Bradford pear") = 6; h_2("Bradford pear") = 4; h_3("Bradford pear") = 3?

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

Using a Bloom filter to determine novelty

STOP: What happens if we have the new smell h_1("Bradford pear") = 6; h_2("Bradford pear") = 4; h_3("Bradford pear") = 3?

Answer: The values at indices 6, 4, and 3 are all zero, so we conclude that we've already smelled this smell (which is not right in this case)!

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

Using a Bloom filter to determine novelty

Novel Query Problem:

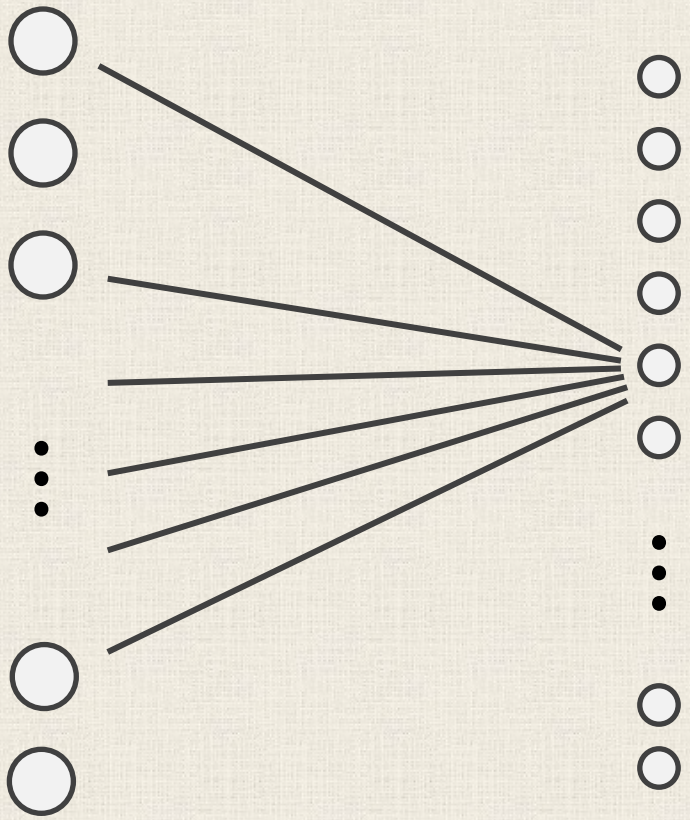
- **Input:** a set of *n* objects *S* and an object *x*.
- **Output:** "Yes" if *x* is in *S*, and "no" otherwise.

Theorem: if hashing *x* produces any Bloom filter values = 1, then *x* is not in *S*. (But if all values are 0, all we can say is that *x* is *possibly* already in *S*).

0	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	1	0	1

Already zero!

An adapted bloom filter in *Drosophila* olfactory system



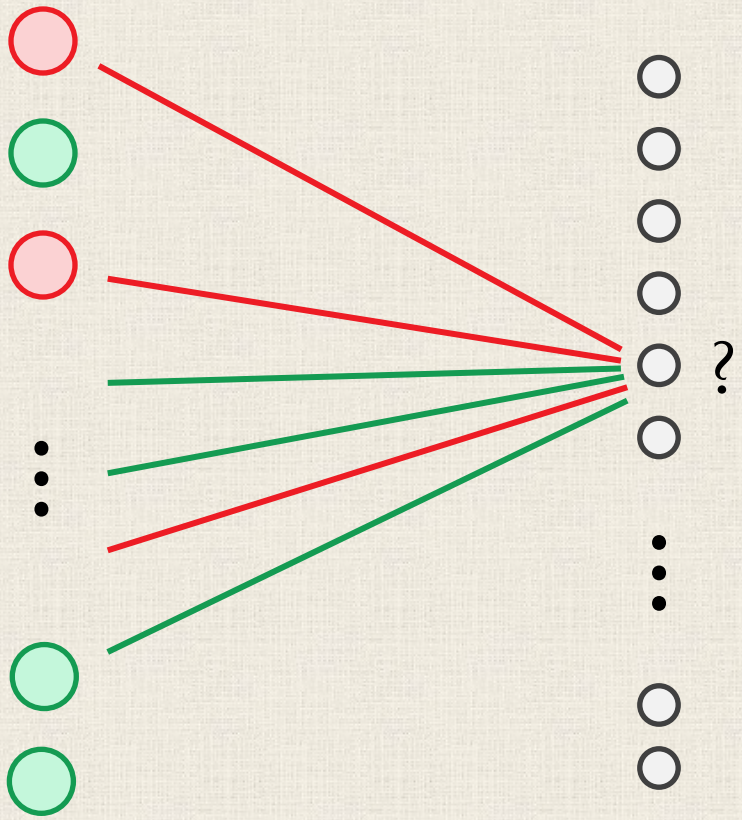
50 olfactory neurons

~2K Kenyon cells (KC)

Connections are "sparse": a KC only receives input from ~6 neurons.

Connections are "random": there is no correlation between KCs' incoming edges.

An adapted bloom filter in *Drosophila* olfactory system



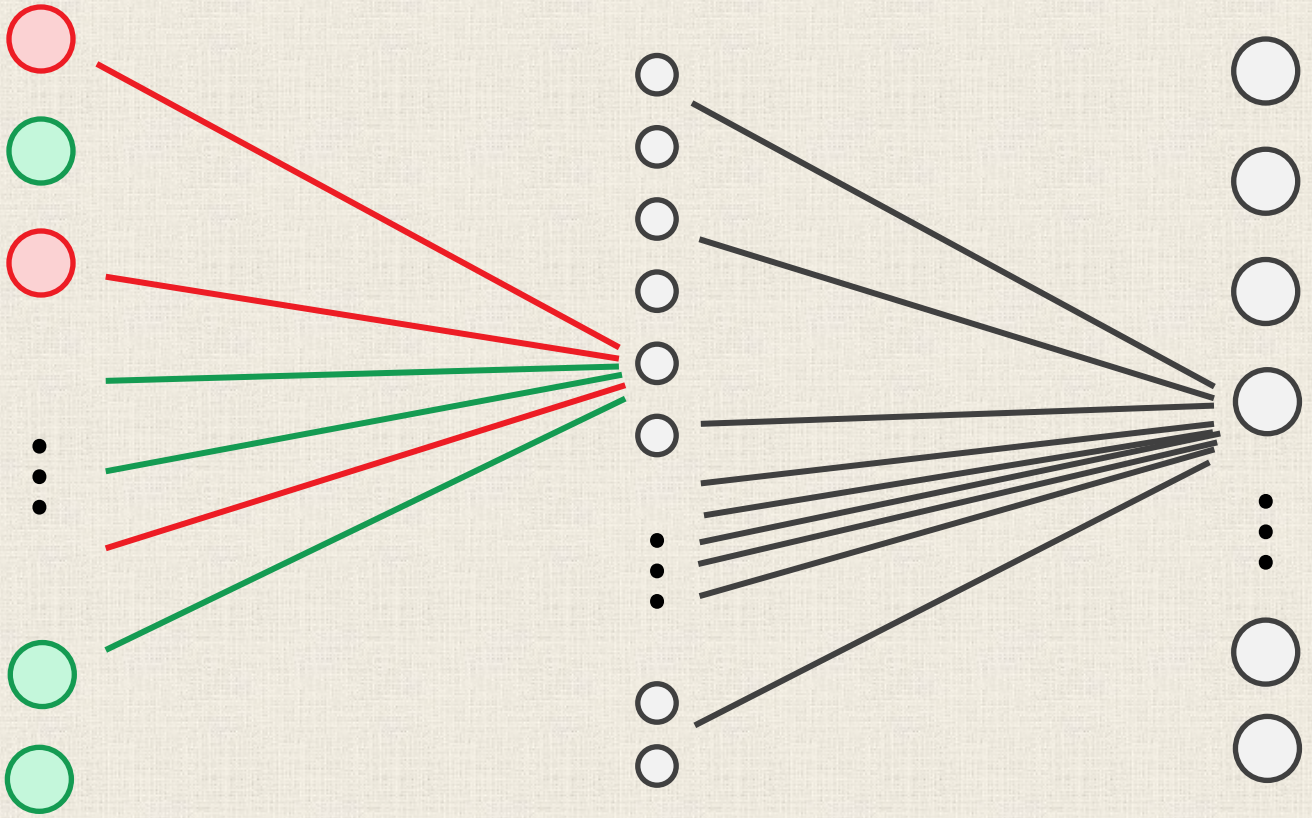
50 olfactory neurons

~2K Kenyon cells (KC)

A smell excites a subset of the neurons, and a KC is excited if enough of its incoming neurons are excited.

A smell typically excites only ~5% (100) of the KCs.

An adapted bloom filter in *Drosophila* olfactory system



50 olfactory neurons

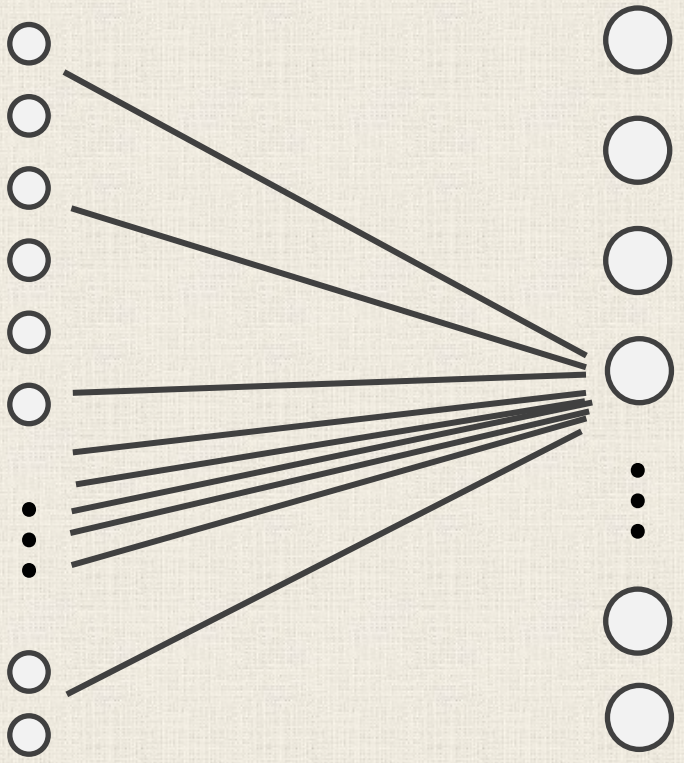
~2K Kenyon cells (KC)

34 Mushroom body output neurons (MBON)

An adapted bloom filter in *Drosophila* olfactory system

An MBON may have indegree ~400 from the KCs.

So an MBON will get an "On" message from ~20 KCs for a given smell.

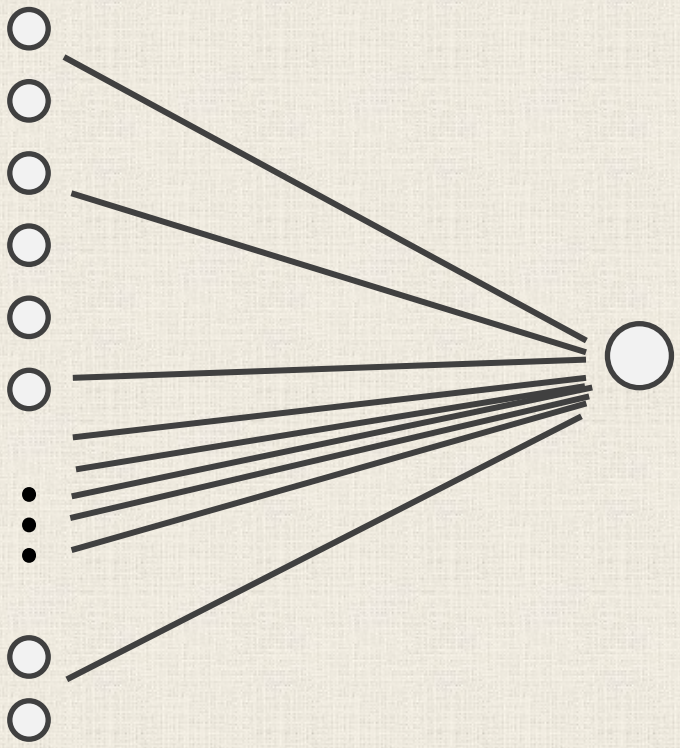


~2K Kenyon cells (KC)

34 Mushroom body output neurons (MBON)

An adapted bloom filter in *Drosophila* olfactory system

Hattori et al. 2014: A single MBON, MBON-α′3, serves as the fly's recognition of a smell's novelty. Let's see how!

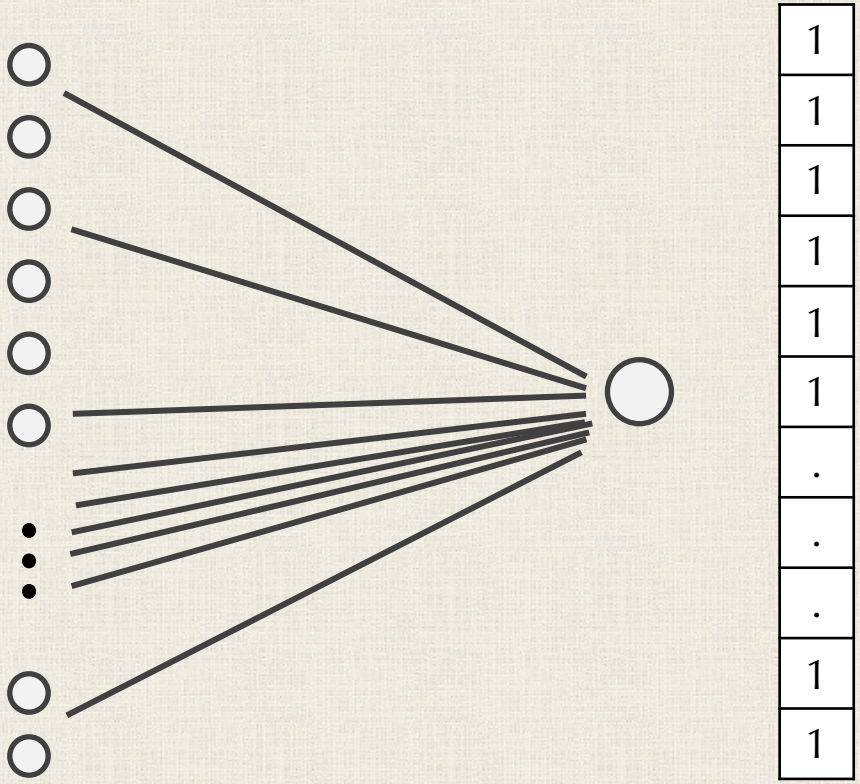


~2K Kenyon cells (KC)

MBON-α′3

An adapted bloom filter in *Drosophila* olfactory system

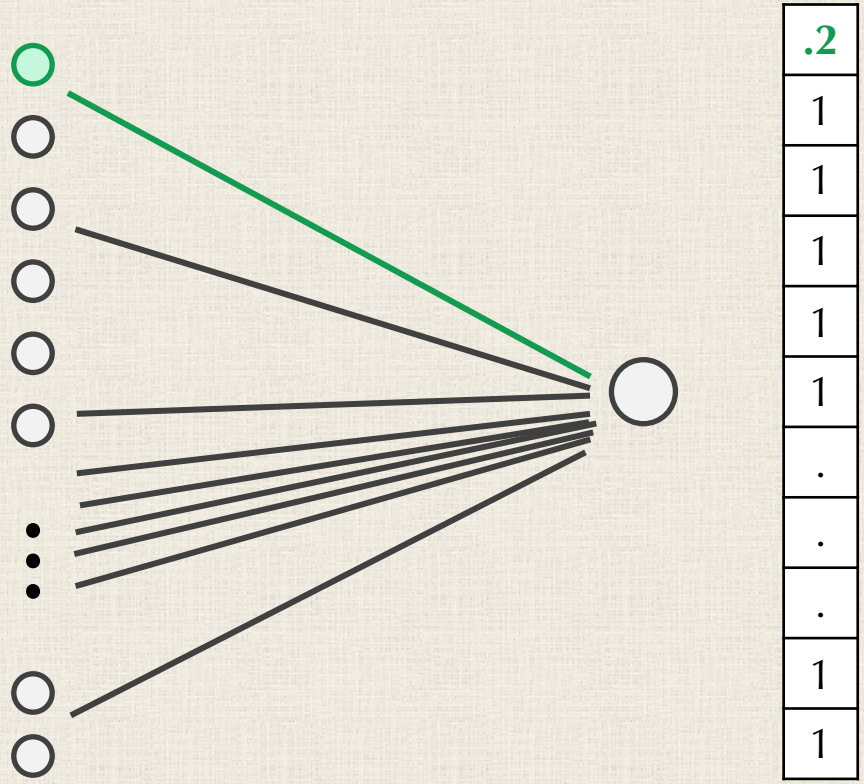
Key point: MBON-α′3 can be represented as a Bloom filter w of length $m \sim 400$ initialized with all $w(i) = 1$. Here, $w(i)$ is the weight of the i-th synapse (edge) into the MBON.



~2K Kenyon cells (KC)

What happens when *Drosophila* senses a smell

If *i*-th incoming KC is "on", set $w(i) = w(i) \cdot \boldsymbol{\delta}$ for some $\boldsymbol{\delta} > 0$.



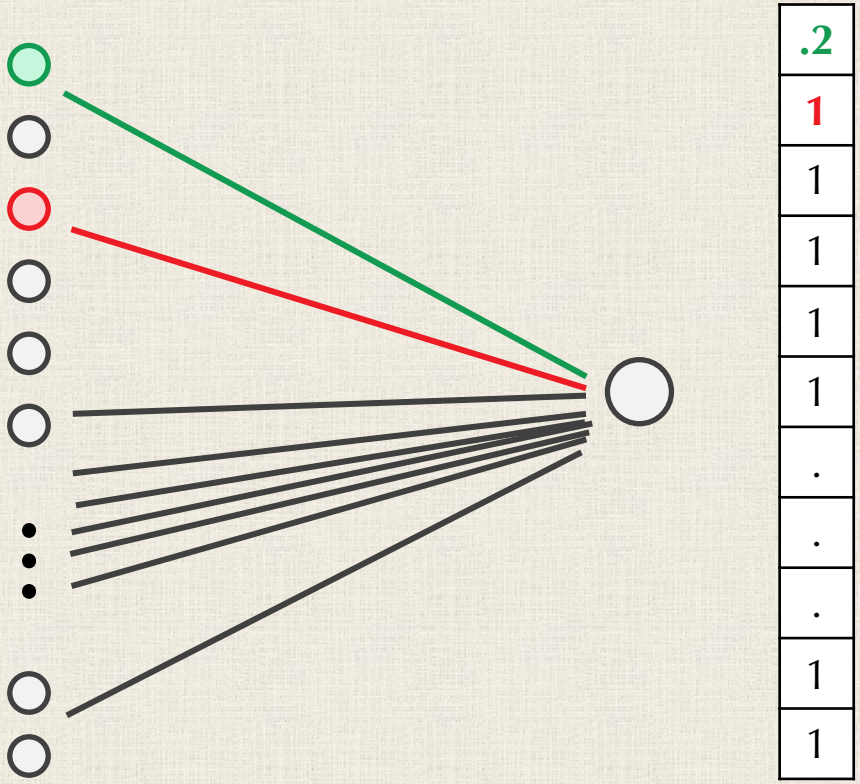
~2K Kenyon cells (KC)

MBON-α′3

What happens when *Drosophila* senses a smell

If *i*-th incoming KC is "on", set $w(i) = w(i) \cdot \boldsymbol{\delta}$ for some $\boldsymbol{\delta} > 0$.

If *i*-th incoming KC is "off", set $w(i) = \min\{1, w(i) + \varepsilon\}$ for some constant $\varepsilon > 0$.



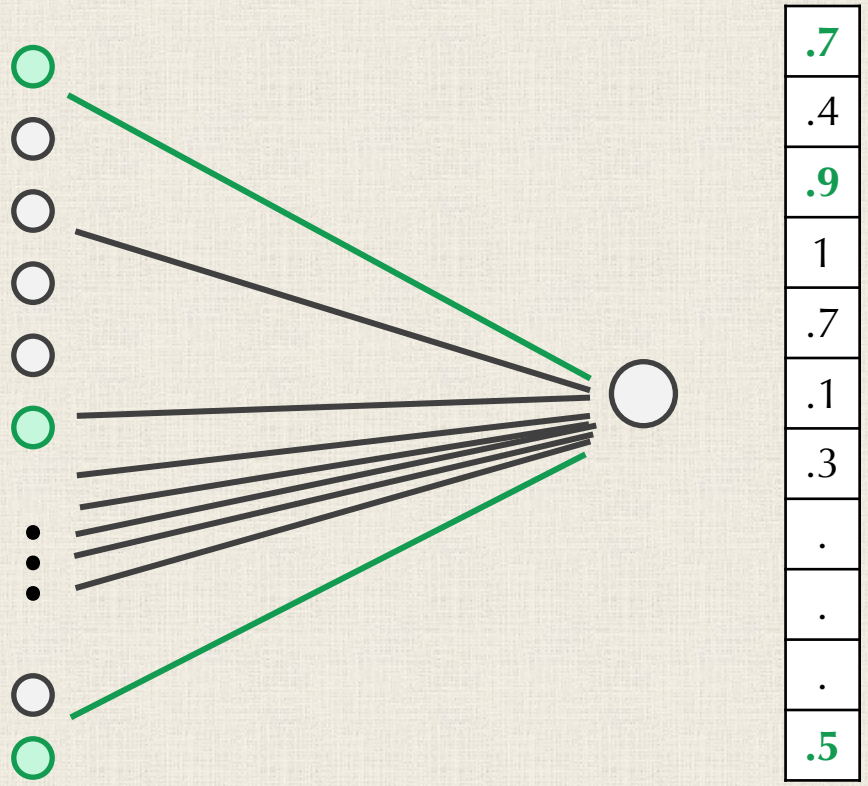
~2K Kenyon cells (KC)

MBON-α′3

What happens when *Drosophila* senses a smell

Over time, as more odors come in, the output neuron will change weight of synapses from KCs.

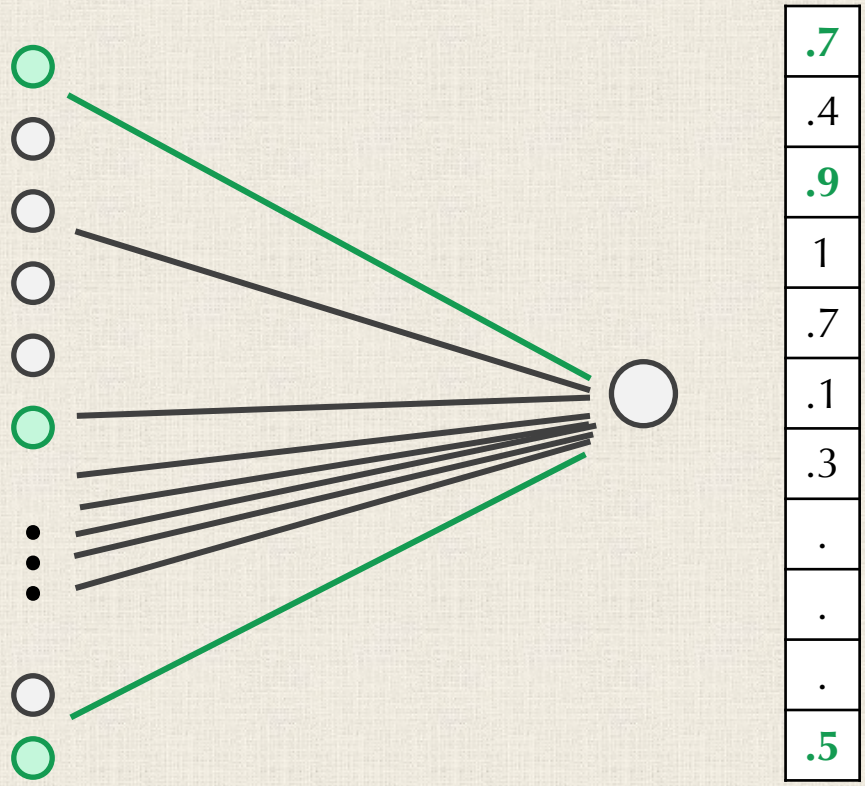
STOP: What would a "newish" odor mean in terms of a given MBON's Bloom filter?



What happens when *Drosophila* senses a smell

Over time, as more odors come in, the output neuron will change weight of synapses from KCs.

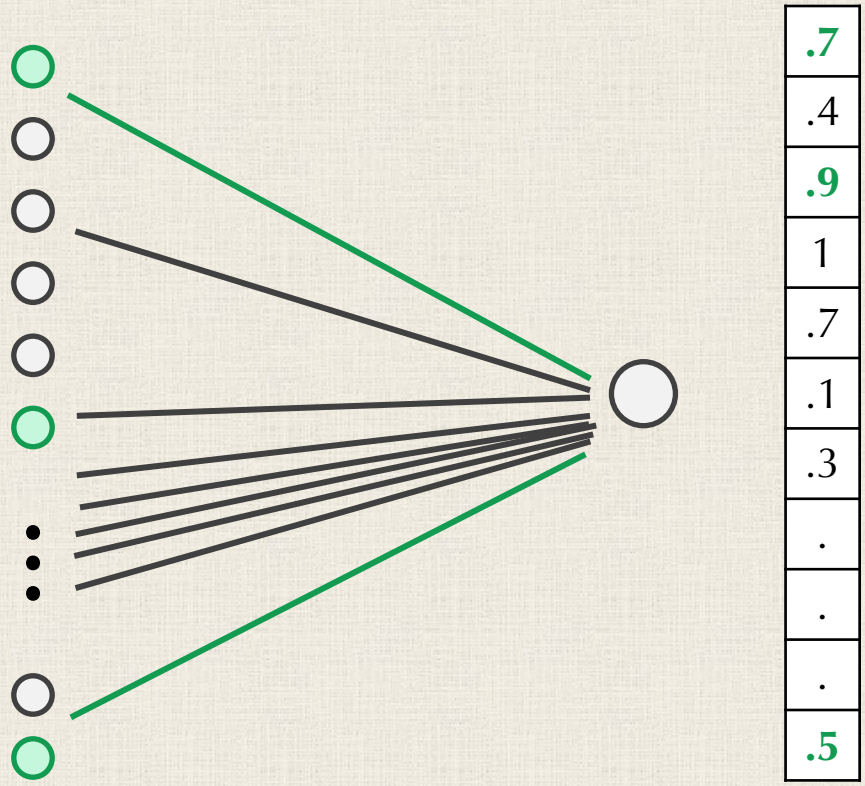
Answer: One that fires KCs with higher synapse weights (i.e., Bloom filter values).



What happens when *Drosophila* senses a smell

So we can measure a "confidence" that a smell is new by summing weights and dividing by *k*.

Answer: One that fires KCs with higher synapse weights (i.e., Bloom filter values).



~2K Kenyon cells (KC)

MBON-α′3

Features of this approach

1. Time-sensitive (memory of old smells "fades")
2. Continuous weights in [0,1] give more info
3. Outperforms other approaches (including traditional Bloom filter) for new query detection!

Proc Natl Acad Sci U S A. 2018 Dec 18;115(51):13093-13098. doi: 10.1073/pnas.1814448115. Epub 2018 Dec 3.

A neural data structure for novelty detection.

Dasgupta S[1], Sheehan TC[2], Stevens CF[3,4], Navlakha S[5].

⊕ **Author information**

Abstract

Novelty detection is a fundamental biological problem that organisms must solve to determine whether a given stimulus departs from those previously experienced. In computer science, this problem is solved efficiently using a data structure called a Bloom filter. We found that the fruit fly olfactory circuit evolved a variant of a Bloom filter to assess the novelty of odors. Compared with a traditional Bloom filter, the fly adjusts novelty responses based on two additional features: the similarity of an odor to previously experienced odors and the time elapsed since the odor was last experienced. We elaborate and validate a framework to predict novelty responses of fruit flies to given pairs of odors. We also translate insights from the fly circuit to develop a class of distance- and time-sensitive Bloom filters that outperform prior filters when evaluated on several biological and computational datasets. Overall, our work illuminates the algorithmic basis of an important neurobiological problem and offers strategies for novelty detection in computational systems.