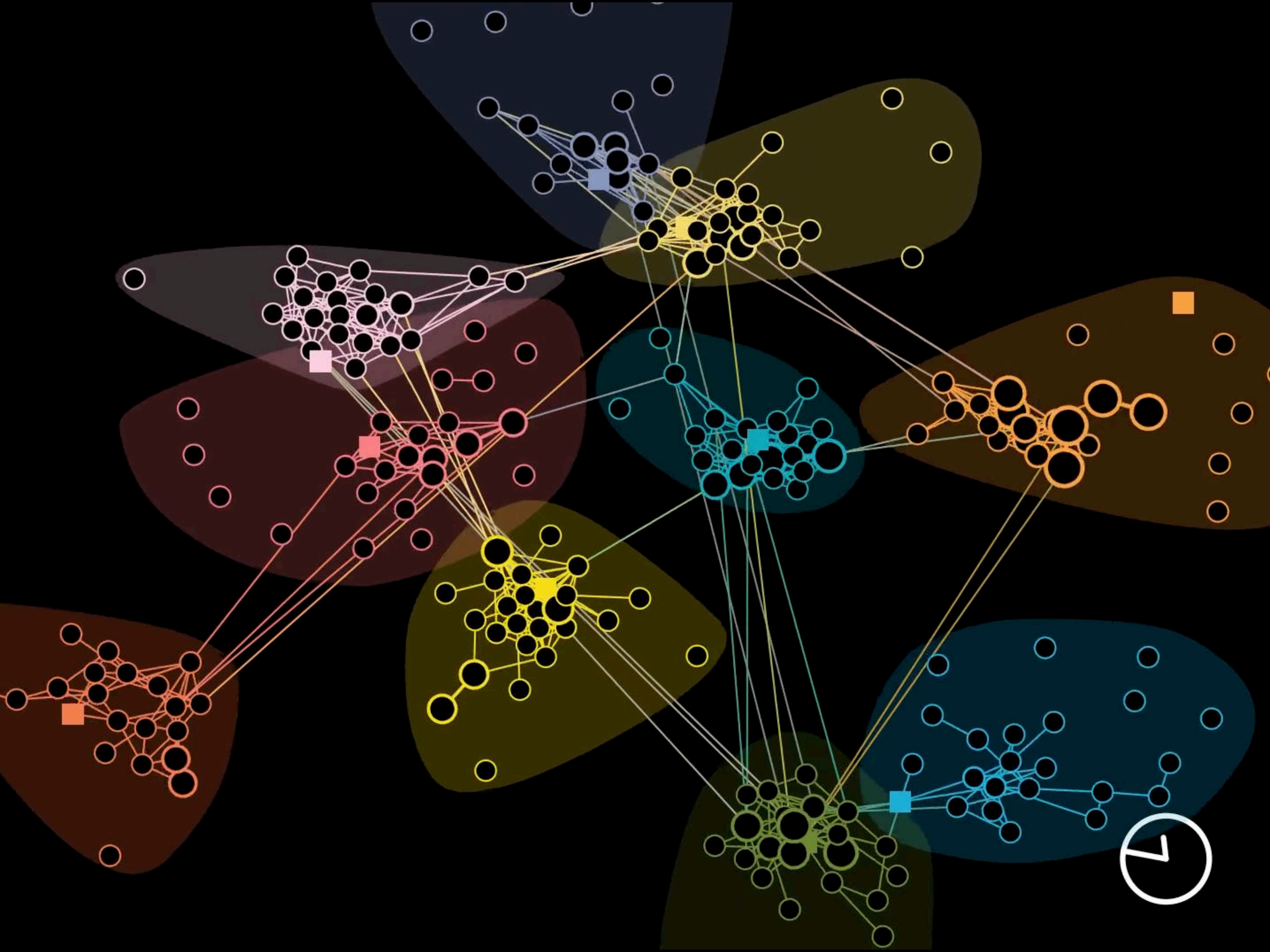
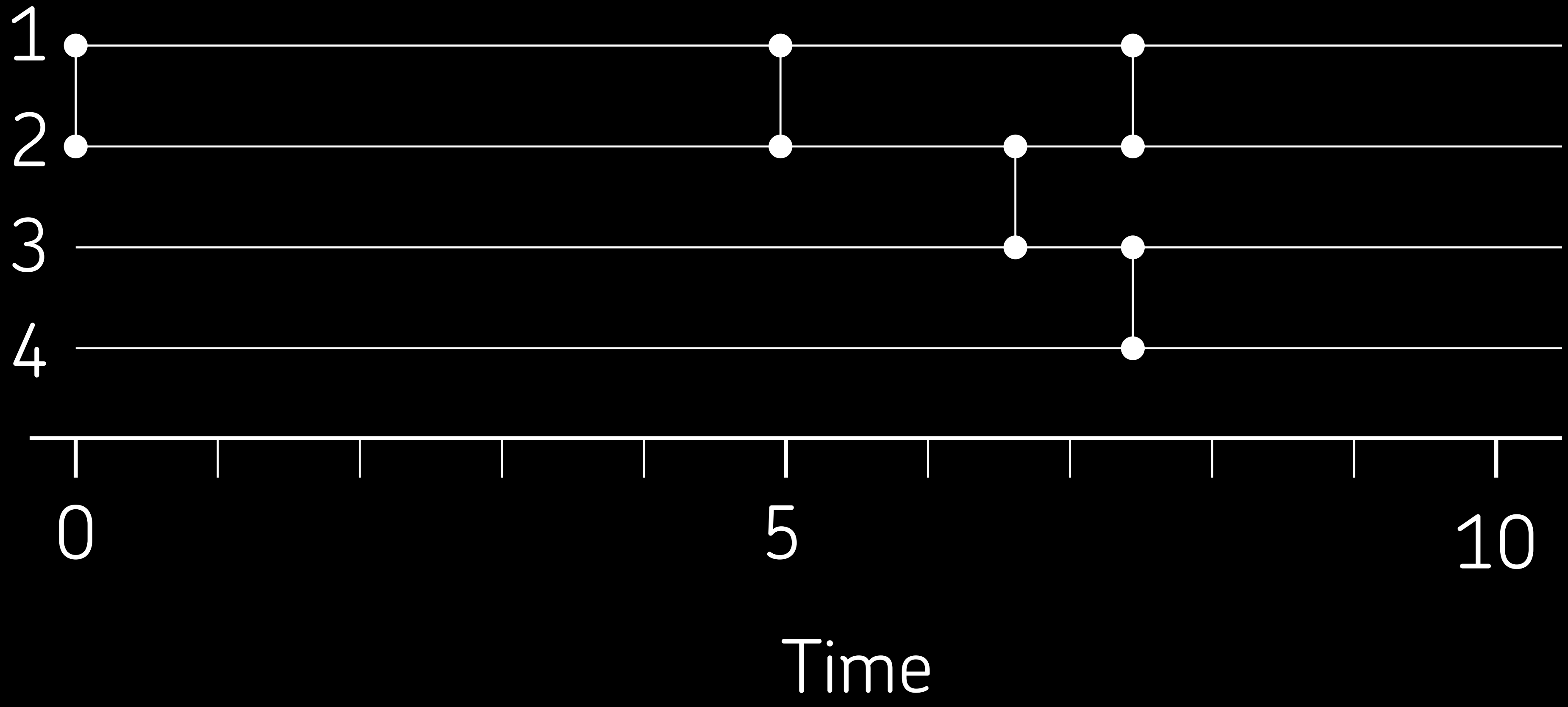
*Petter Holme*

# Temporal network epidemiology: Subtleties and algorithms



## Temporal networks

| $i$ | $j$ | $t$ |
|---|---|---|
| 1 | 2 | 0 |
| 1 | 2 | 5 |
| 2 | 3 | 7 |
| 1 | 2 | 8 |
| 3 | 4 | 9 |
| ⋮ | ⋮ | ⋮ |



## SIR on static graphs

- Infection rate $\beta$ → Infection across an SI link* is a Poisson process for its duration.
- . . . *Edge in a static graph.
- Constant recovery rate $\nu$ → Exponentially distributed durations of infection.
- One seed chosen at random among all vertices.

## Algorithms

- Gillespie
- Event-driven algorithm (Kiss, Miller, Simon, 2017), a.k.a. Next-reaction
- Composition / rejection (St-Onge, *Comp. Phys. Comm.* 2019)

## SIR on temporal graphs

Design principles:

- **Realism** | After all, the goal is to simulate reality
- **Continuity** | It should be possible to reduce the time dimension and get static network epidemiology.
- **Simplicity** | Keep the same level of abstraction throughout the modeling.
- **Generalizability** | It should be easy to extend the model.
- **Speed** | As a tiebreaker among design principles.

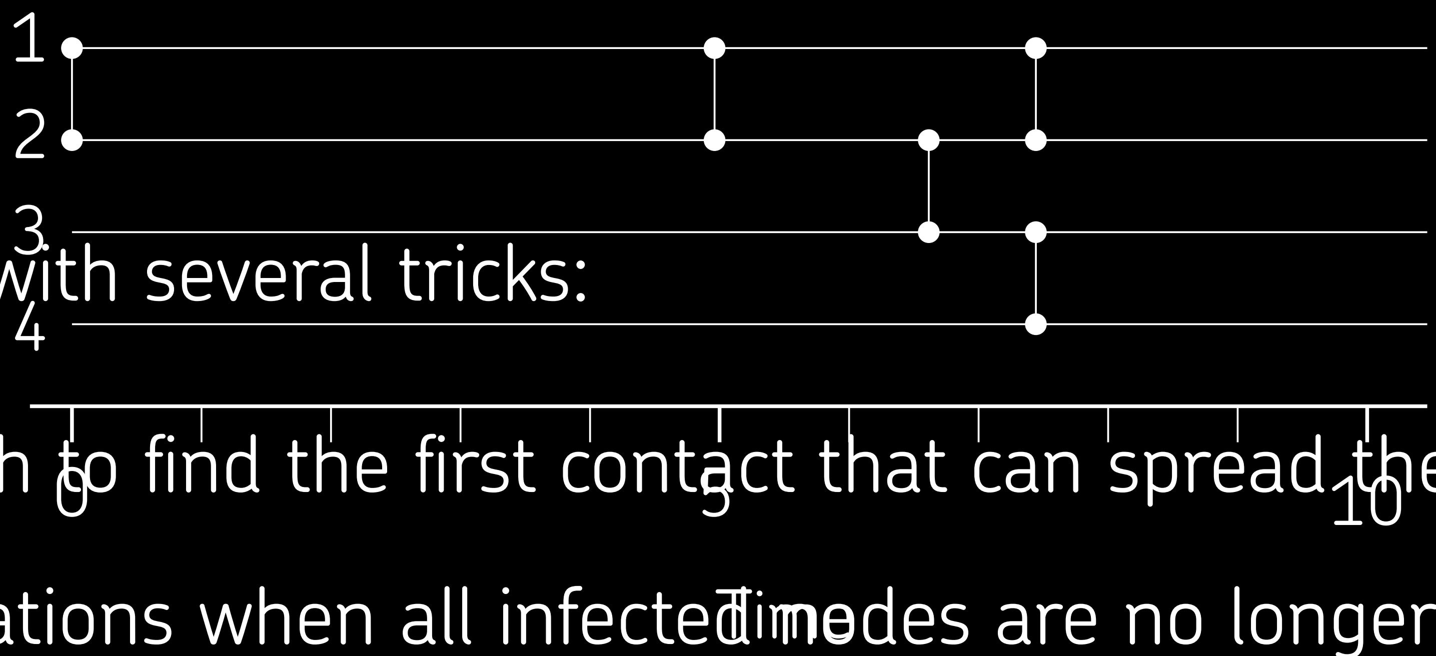
P Holme, 2021. Fast and principled simulations of the SIR model on temporal networks. *PLoS ONE* 16(2): e0246961.

## Algorithmic model formulation

- **Initialization** | Initialize all individuals to susceptible.
- **Seeding** | Pick a random individual *i* and a random time $t_i$ in the interval $[0,T)$. At time $t_i$, infect *i*.
- **Recovery** | Whenever a node becomes infected, let it stay infected for an exponentially distributed time $\delta$ before it recovers.
- **Contagion** | If *i* got infected at time $t_i$ and is still infected at time $t > t_i$, and *j* is susceptible at time *t*, then a contact $(i,j,t)$ will infect *j* with probability $\beta$.

## Straightforward algorithm

1. Initialize all nodes as susceptible.
2. Run through the contacts in increasing order of time.
3. If a there is a contact between a susceptible and infectious node, then infect the susceptible node with probability β.
4. Whenever a node gets infected (including the source), then draw its time to recovery from an exponential distribution, and change its state to I.
5. Stop the simulation when there are no infectious nodes.



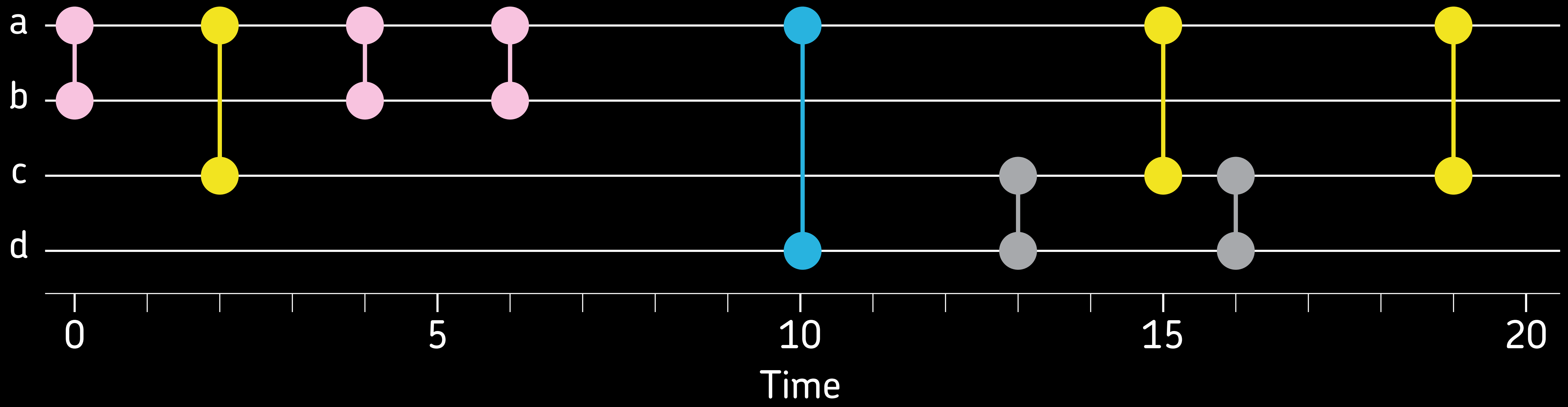
Could be sped up with several tricks:

1. Bisection search to find the first contact that can spread the disease.
2. Stop the simulations when all infected nodes are no longer active.

## Event-based algorithm

Internal representation of the temporal network

**A**



**B**

a:

| | | | |
|---|---|---|---|
| c | 2 | 15 | 19 |
| d | 10 | | |
| b | 0 | 4 | 6 |

b:

| | | | |
|---|---|---|---|
| a | 0 | 4 | 6 |

c:

| | | | |
|---|---|---|---|
| a | 2 | 15 | 19 |
| d | 13 | 16 | |

d:

| | | |
|---|---|---|
| c | 13 | 16 |
| a | 10 | |

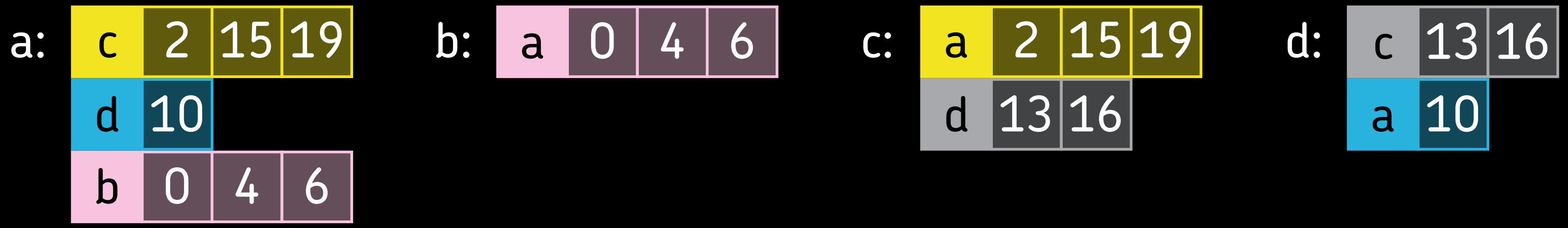
Contact lists ordered in decreasing order of the last element.

## Event-based algorithm

Contagious contact

Finding what contact between two nodes *i* and *j* that would spread the disease, if any.

1. Use bisection search to find the smallest index *k* of $\mathbf{t}_{ij}$ such that $t_i < \mathbf{t}_{ij}(k)$. Where $\mathbf{t}_{ij}(k)$ denotes the *k*'th contact of $\mathbf{t}_{ij}$.
2. Add a random number *K* generated by $\lfloor \log(1-X) / \log(1-\beta) \rfloor$ to *k* and denote the sum by $k'$. (The probability of the *k*'th event of a Bernoulli process occurring.) *X* is a uniform random number in $[0,1)$.
3. If $k'$ is larger than $\mathbf{t}_{ij}$'s number of elements, then return some out-of-bounds value (to signal that no contact will spread the disease). Otherwise, return $k'$—the contact between *i* and *j* that could be contagious.



## Event-based algorithm

<u>Infect</u>

Handling the infection of one node *i*.

1. Pop the individual *i* with the earliest infection time from the heap.
2. Iterate through the neighbors *j* of *i*.
    a. If *j* is susceptible, get the time $t_j$ when it would be infected by *i* (by calling <u>contagious-contact</u>).
    b. If it simultaneously holds that
        i. There is no earlier infection event of *j* on the heap.
        ii. *i*'s recovery time is not earlier than $t_j$.

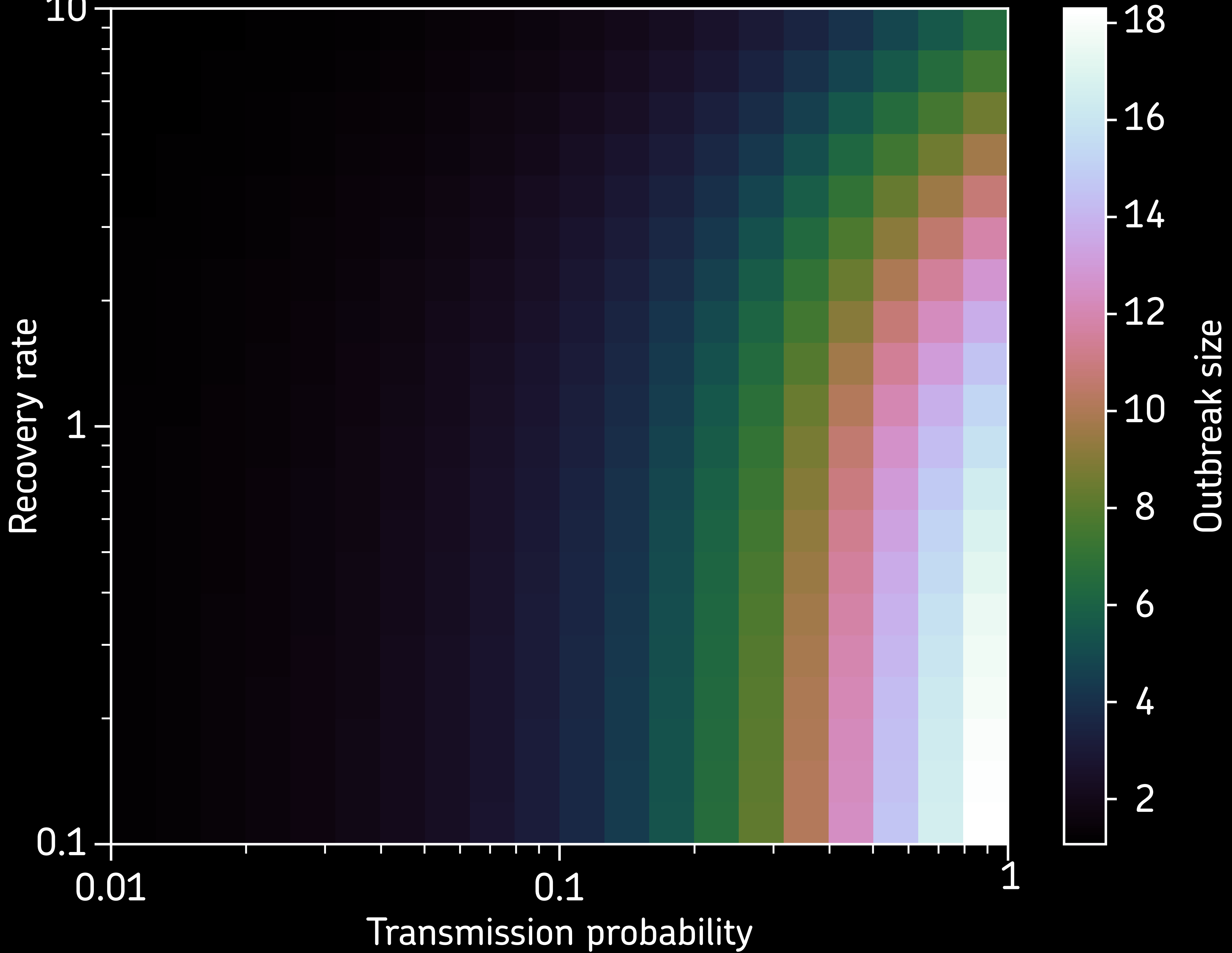
    then put the contagion (*i* infects *j* at time $t_j$) on the heap.

## Event-based algorithm

Taken together:

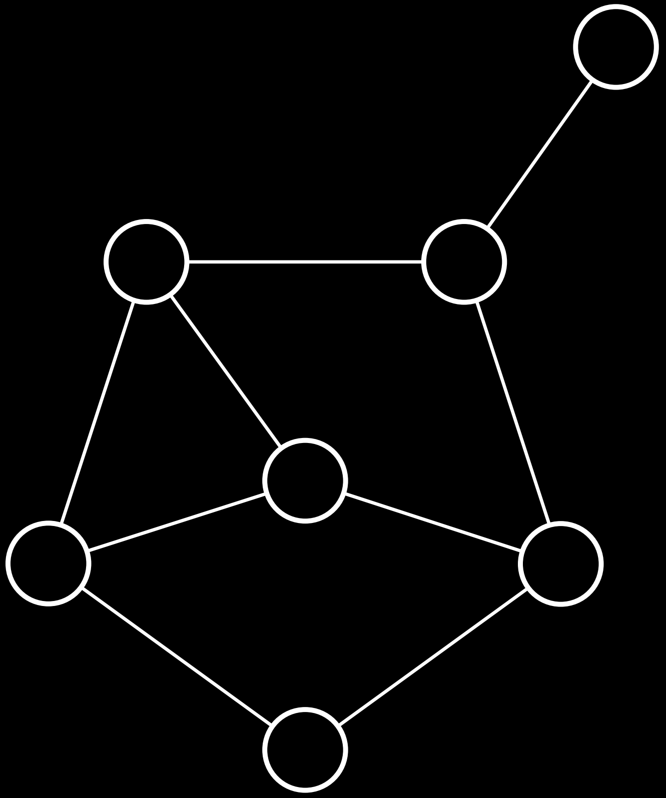
1. Read the network and initialize everything.
2. Infect the source node.
3. While there are any nodes left on the heap, call <u>infect</u>.
4. Reset the simulation.
5. Go to 2 until you have enough averages.
6. Evaluate the output.

# Example output: SocioPatterns Gallery day 1

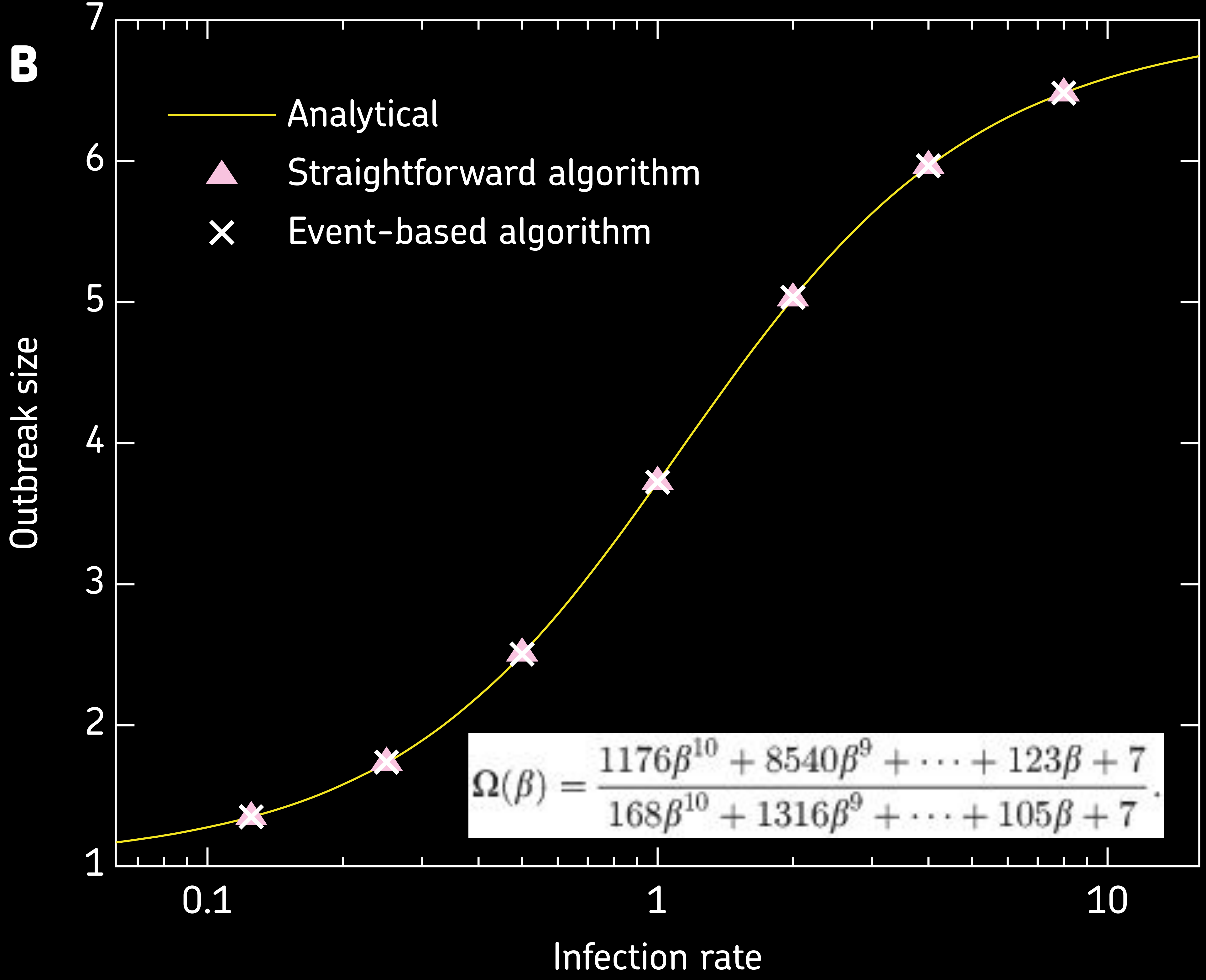


# Validation of the program

A



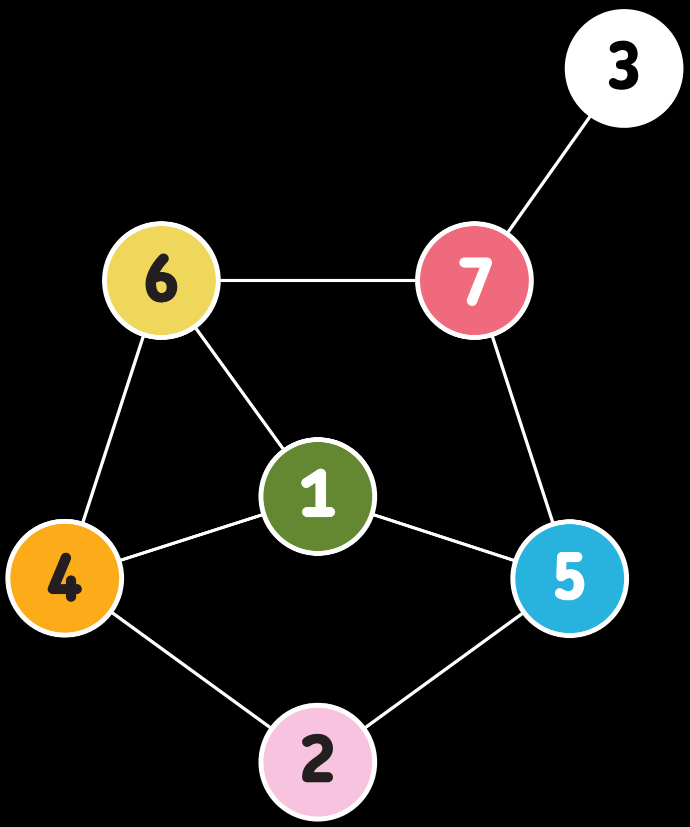
B

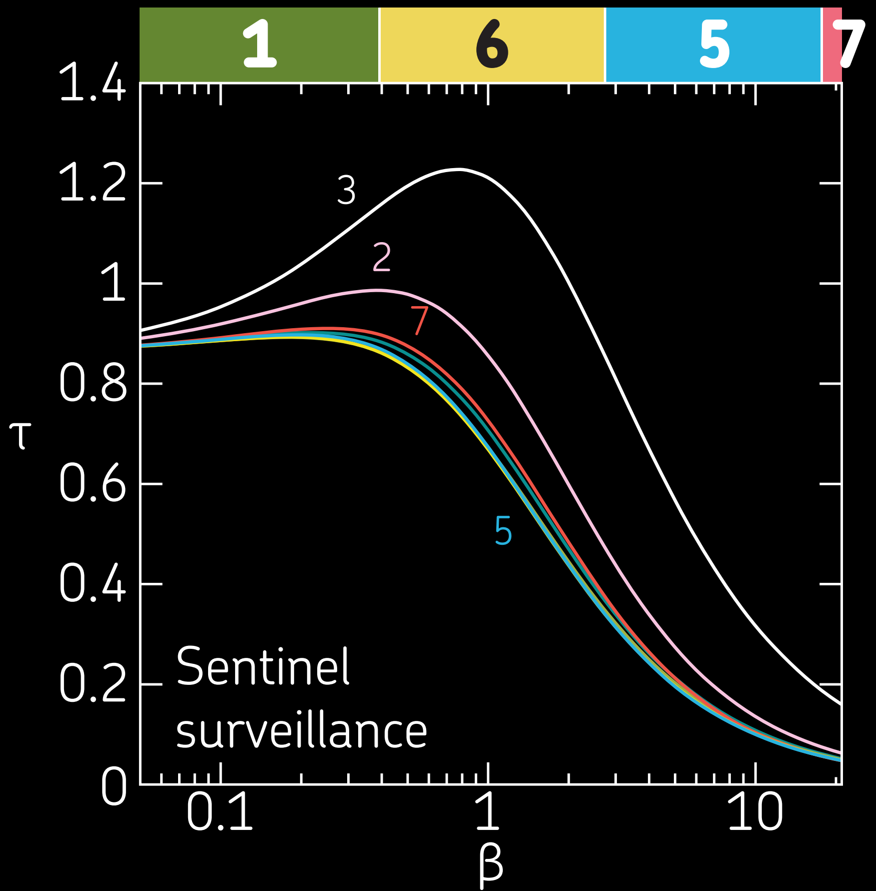


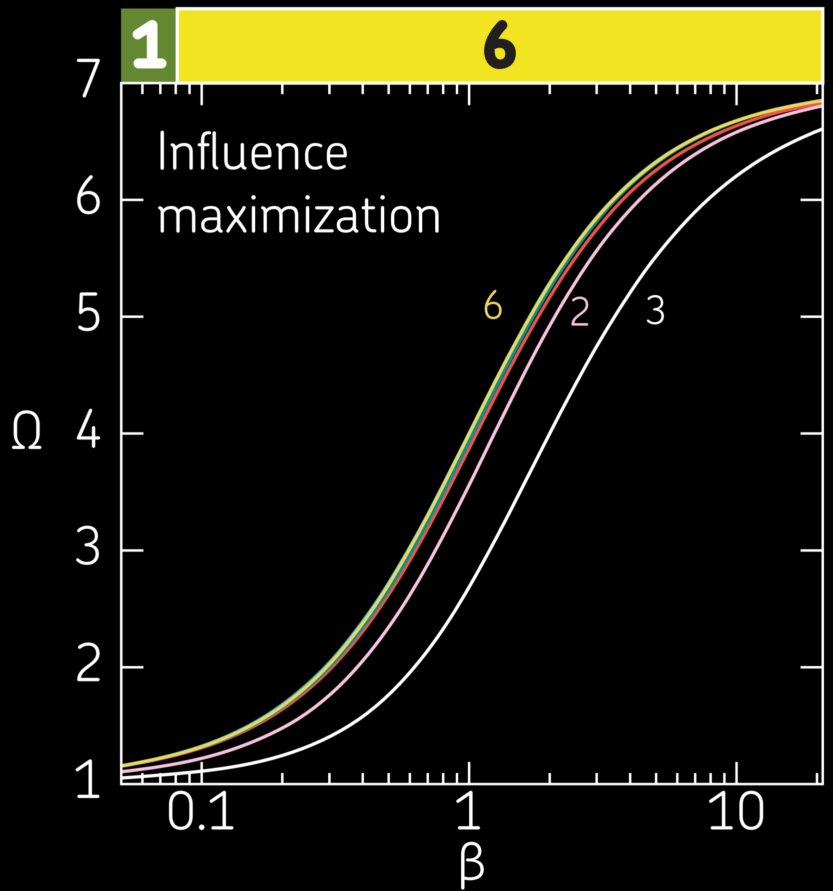
$$\Omega(\beta) = \frac{1176\beta^{10} + 8540\beta^{9} + \cdots + 123\beta + 7}{168\beta^{10} + 1316\beta^{9} + \cdots + 105\beta + 7}.$$

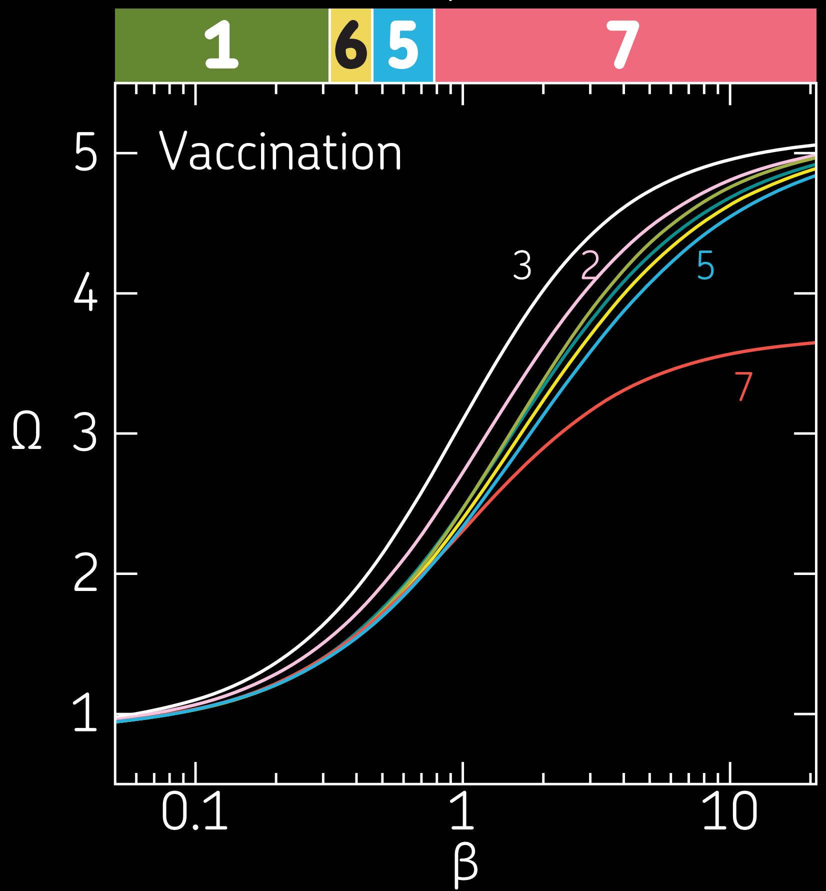
# A graph with complex behavior w.r.t SIR

P Holme, *Phys. Rev. E* 96, 062305 (2017).







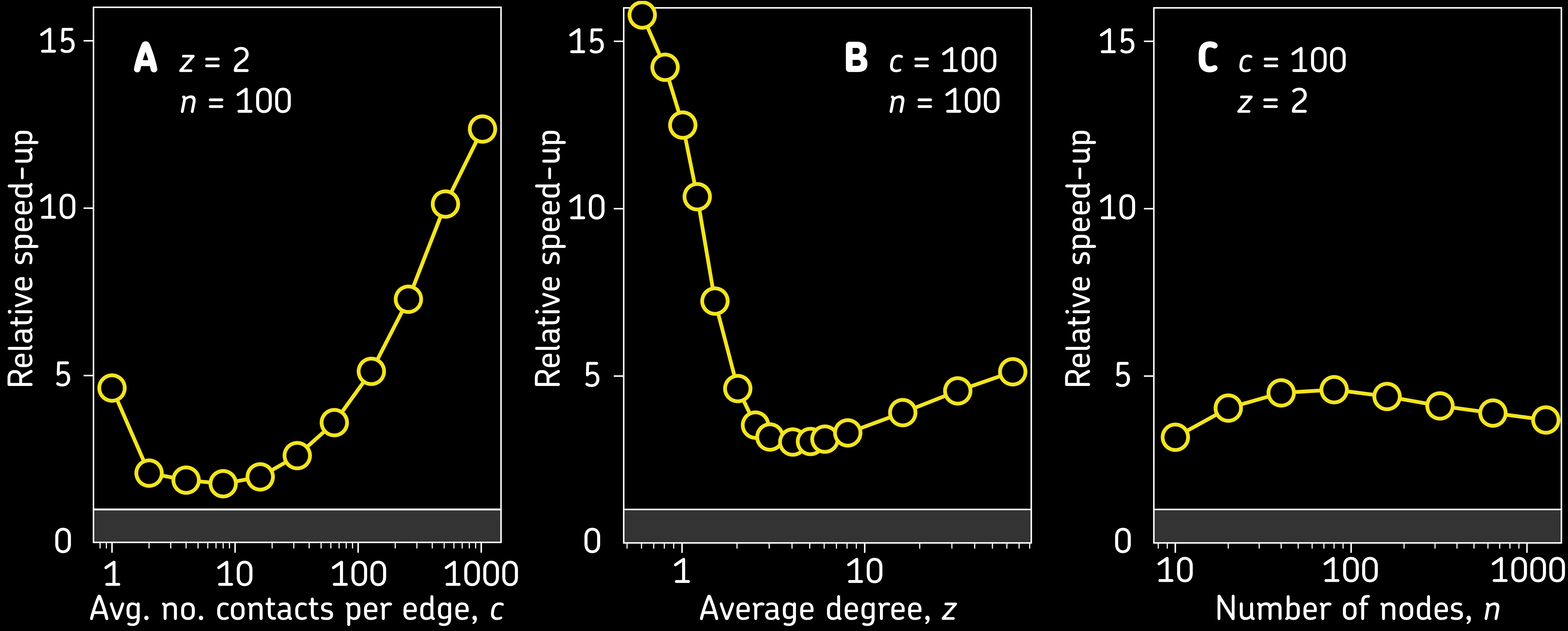


## Complexity

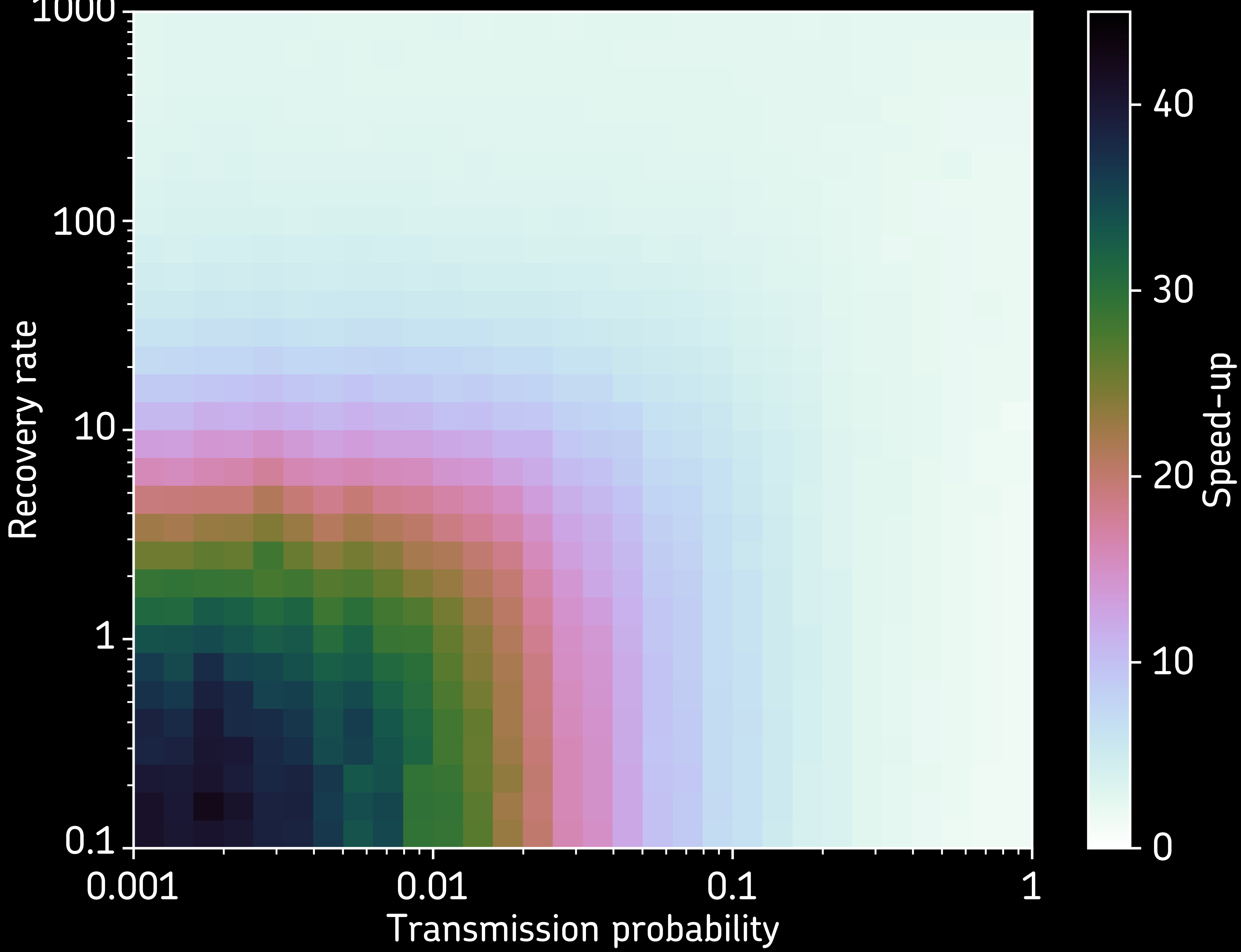
$N$ = number of nodes
$M$ = number of edges (node pairs with at least one contact)
$C$ = number of contacts

Straightforward algorithms: $O(N + C)$
Event-driven algorithm: $O(L \log N \log C)$

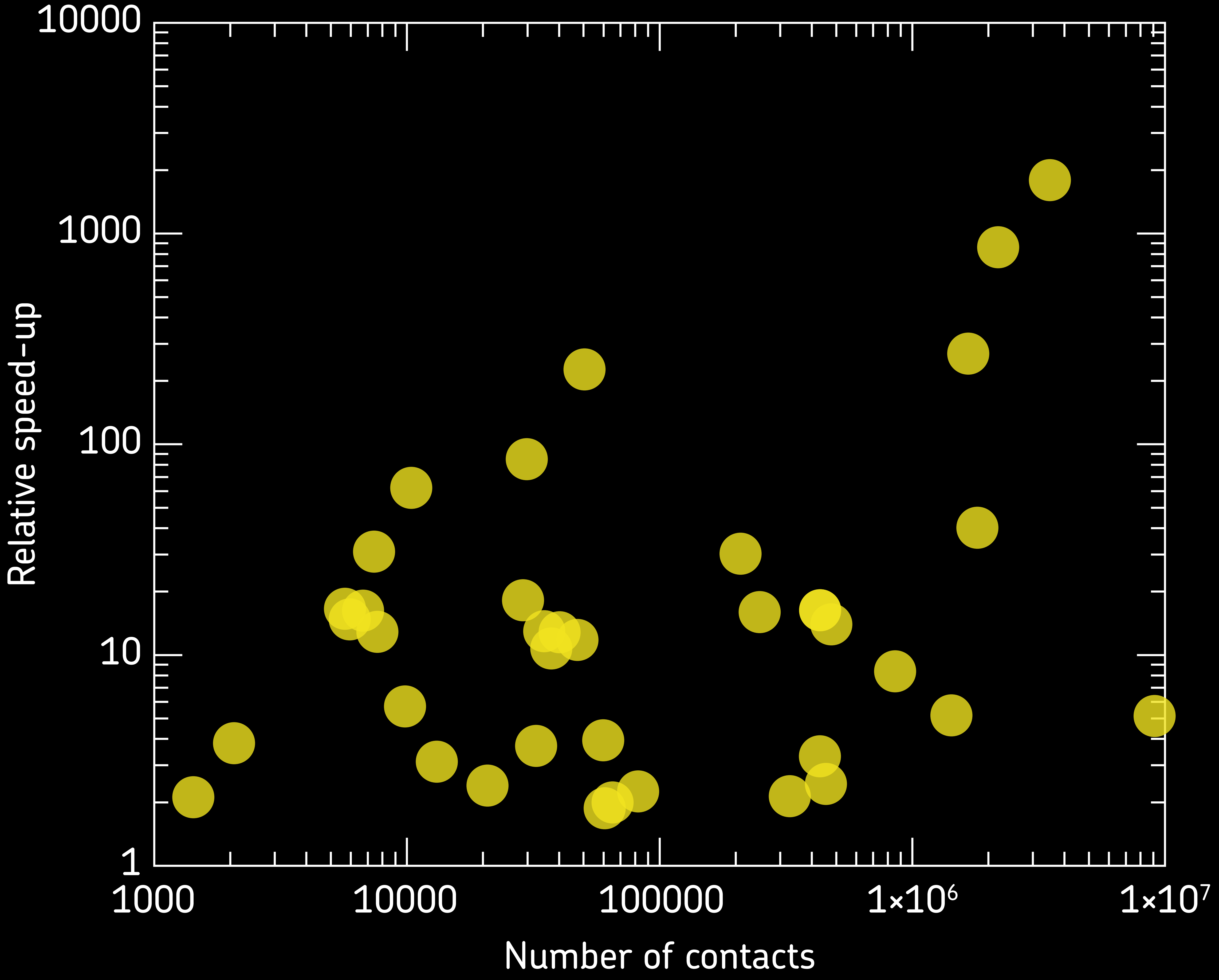
# Speed-up relative to the straightforward algorithm of artificial networks



Speed-up relative to the straightforward algorithm



## Speed-up relative to the straightforward algorithm



## SIR with fixed infection-duration

Work

Everything like before, except

Could we then compute the infection probability of a node for all β at once?

in

progress

## SIR with fixed infection-duration

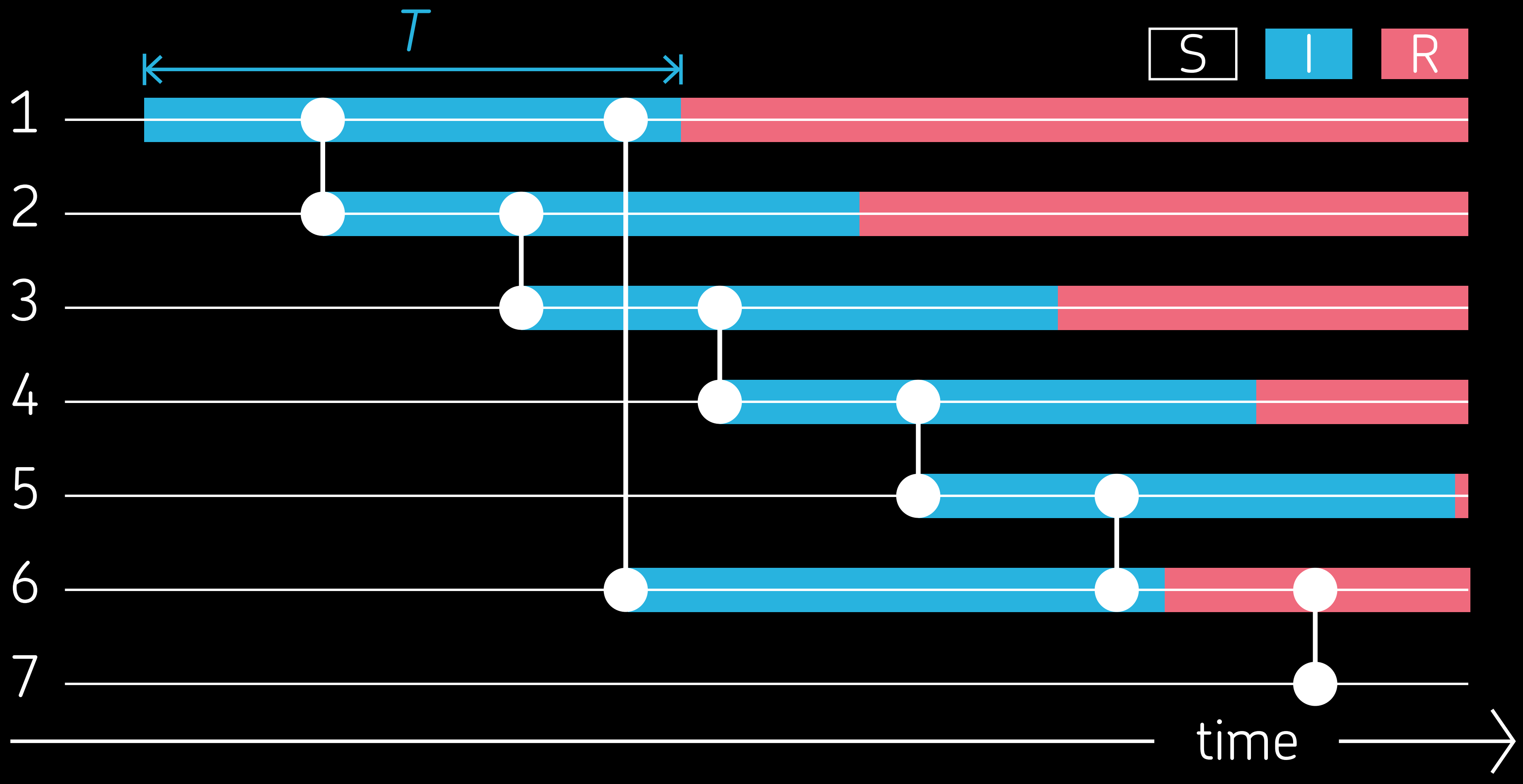
Everything like before, except the disease lasts a fixed duration $T$.

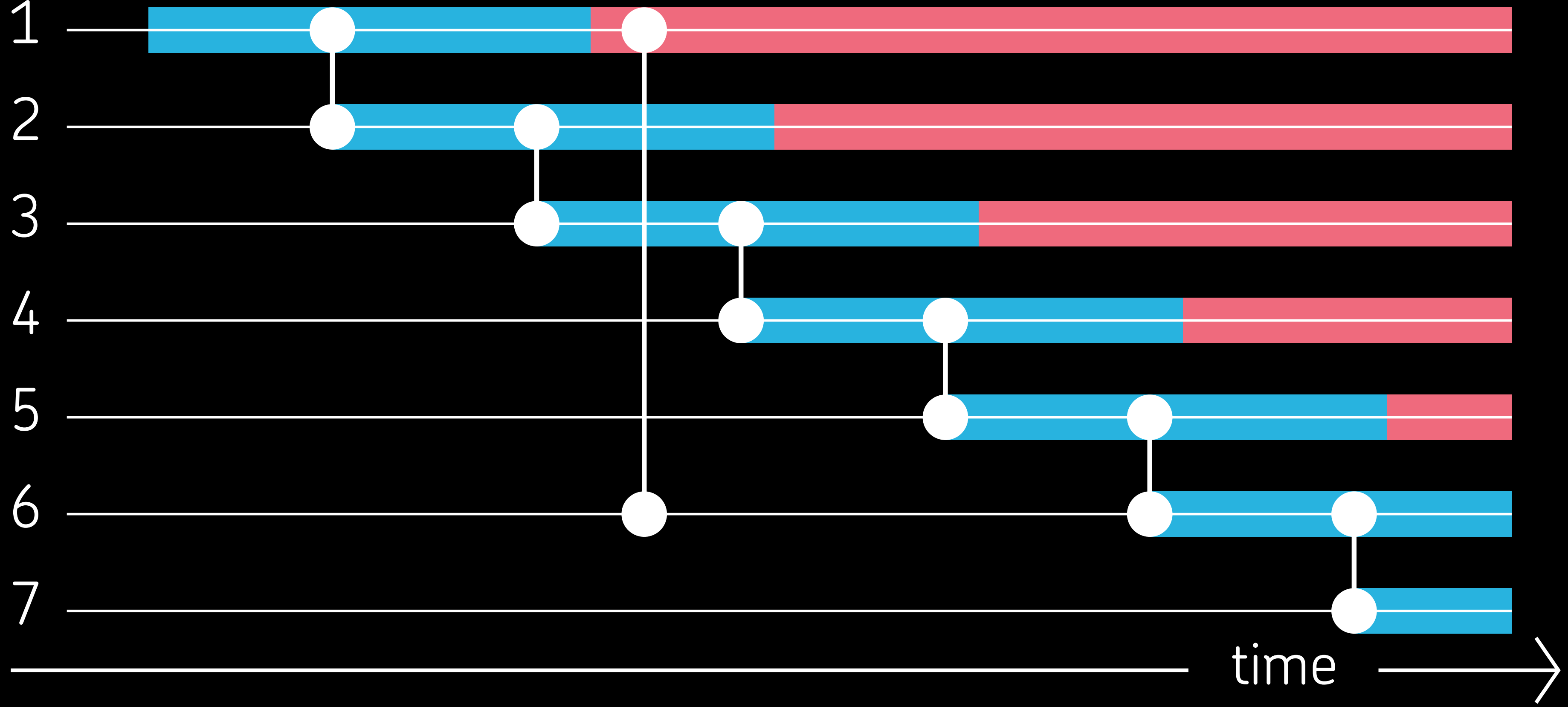
This makes the problem more structured and should be much faster. For example, we then compute the infection probability of a node for all β at once?

Naïve algorithm idea:

1. For every node $i$, let a variable $x_i$ representing the minimum value of $T$ needed to reach $i$.
2. Go through every contact $(i,j,t)$ in increasing $t$. Update $x_i$ to $\max(x_i,t_j+x_j)$ where $t_j$ is the time since $j$ was infected, and similarly for $x_j$.

## The catch





petterhol.me

Code: github.com/pholme/tsir/

*Thank you!*