RESEARCH ARTICLE

Fast and principled simulations of the SIR model on temporal networks

Petter Holme *

Tokyo Tech World Research Hub Initiative (WRHI), Institute of Innovative Research, Tokyo Institute of Technology, Yokohama, Japan

* holme@cns.pi.titech.ac.jp

Abstract

The Susceptible–Infectious–Recovered (SIR) model is the canonical model of epidemics of infections that make people immune upon recovery. Many of the open questions in computational epidemiology concern the underlying contact structure's impact on models like the SIR model. Temporal networks constitute a theoretical framework capable of encoding structures both in the networks of who could infect whom and when these contacts happen. In this article, we discuss the detailed assumptions behind such simulations—how to make them comparable with analytically tractable formulations of the SIR model, and at the same time, as realistic as possible. We also present a highly optimized, open-source code for this purpose and discuss all steps needed to make the program as fast as possible.

Introduction

Infectious diseases constitute a significant burden to global health and will continue to be that for the foreseeable future. To aid policy-making, one needs to test scenarios and thus run epidemic simulations. Since such simulations rely on statistics—ideally comprising millions of simulated outbreaks—often epidemic simulations can be prohibitively slow. It is thus essential to have fast algorithms to simulate epidemics.

The standard approach for epidemic modeling is *compartmental models* [1]. These are a class of models that divide a population into different classes with respect to the disease and assign transition rules between these classes. One of the most canonical of compartmental models is the susceptible–infectious–recovered (SIR) model that assumes a scenario where people get immune upon recovery.

However, there is more to the story of how to simulate epidemic spreading than just compartmental models. The ways people come in contact so that the disease can spread is crucial for epidemics, so one should not neglect them [2]. There can be structures, or regularities, in the contact patterns that affect the disease propagation. One way of addressing this problem is to base the simulations on temporal networks [3–5]. These encode who that is in contact with whom, and when these contacts happen. Temporal network epidemiology has been applied to diseases from HIV [6] to influenza [3], from COVID-19 [7] to livestock diseases [8]. It should become even more useful with the increasing availability of large-scale data set [9].

Simulating the SIR model on temporal networks may seem straightforward. One can run through contacts in order of time and let each contact be a potential infection event. Still, there are many details the modeler has to sort out: How should one initialize the epidemics? How should one deal with simultaneous contacts? These seemingly technical decisions do affect the result. They may rarely affect the experiments' qualitative conclusions, but they could be large enough to hinder comparison between different studies. Thus, it is desirable for the temporal-network modelers to agree on the above details to arrive at such a full model description, motivated by simple principles. There could be situations where these principles are invalid, and thus a different version could be motivated, but we believe the decisions we discuss should not be glossed over.

One goal for this paper is to establish an exact formulation of the SIR model in temporal networks, that could—like the Markovian SIR model on static networks—serve as a common ground for studying effects of temporal network structure, or a starting point when exploring special cases (for example what happens if the duration of infectiousness is broadly distributed). The other goal is to present an algorithm for simulating such a model.

This paper will proceed by discussing the principles of a full SIR model for temporal networks, then present an algorithm for this purpose, and finally validate and evaluate the performance of this code.

Methods

In this section, we will discuss the considerations behind our precise formulation of the SIR model on temporal networks. We will also present and evaluate an algorithm to simulate this model. We assume temporal networks can be represented as a *contact list* [10]—a list of C *contacts* (sometimes called "events") (i, j, t), meaning that individuals i and j were in contact at time t. The order of the first two arguments does not matter. We will use N to represent the number of nodes; T to represent the duration of the data set (the time between the first and last contact). We call a pair of nodes with at least one contact an *edge*.

Design principles

In a broad view, it is obvious how to simulate the SIR model on contact lists: One assigns one state—susceptible (S), infectious (I), or recovered (R)—to every individual. If a susceptible appears in contact with an infectious, the susceptible can become infectious with some probability. After some time, the infectious will recover. All temporal network studies using the SIR model follow these conventions [4]. However, when it comes to more subtle decisions, like how to initialize the network, there are many solutions in the literature. Some of these solutions are well-motivated, some are not. In either case, these inconsistencies between studies are undesirable.

In constructing a complete specification for the SIR model on temporal networks, we will pursue the following design principles.

Realism. If our goal is to simulate reality, the first guiding principle should be to make a realistic model. At the same time, we are willing to compromise. The ultimate purpose of this type of computational epidemiology is not accurate forecasting, but to enable researchers to compare scenarios or interventions. For example, how can one best identify influential spreaders [11, 12] or important contacts [13]? Therefore, we will not put this principle above the others.

Continuity. To compare results from temporal network studies with other representational frameworks—like static network epidemiology or differential equations—our simulations

should give the same results with the same assumptions. Static network epidemiology typically (often implicitly) assume the contacts to result from a Poisson process. This is mostly to ensure continuity to the well-mixed, Markovian SIR model—i.e., the most basic, textbook version [1, 14]. So when we run our simulation on data with exponentially distributed interevent times (like a Poisson process), then we should get the same result as with SIR on static networks.

Simplicity. The "continuity" principle itself entails many simplifications. The point of this design principle is to keep the same level of abstraction throughout the modeling. It means that the simulation components that do not concern the limit of static network epidemiology should be as simple as those that do.

Generalizability. Another criterion (which in most cases overlap with simplicity), is that it should be possible to extend the model. Relaxing one of the assumptions—say that all contacts between susceptible and infectious are equally likely to spread the disease—should not conflict with another component of the model.

Speed. The above principles suffice to derive most of our detailed formulation of the model. As a tiebreaker principle, we advocate choosing options that make the simulation as fast as possible.

Precise model formulation

As mentioned, we assume a population of n individuals whose contacts are described by a contact sequence. Every individual is in precisely one state S, I, or R. If i is infectious and j susceptible at time t, then a contact (i, j, t) can cause i to become infectious. An infectious individual will eventually recover.

Mixed discrete and continuous times. In most empirical temporal networks, time is discretized. For example, the widely used Sociopatterns data report contacts at 20 seconds intervals [15]. The standard in static network epidemiology, on the other hand, is to use continuous time. This is necessary to make the model Markovian so that it reduces to the standard differential-equation version of the SIR model if the network is fully connected. Although this mix of discrete and continuous times may seem strange, it does not pose any technical or conceptual problem (cf. Ref. [16]). From a conceptual point of view, we can assume time is continuous—the contacts happen at integer times. It is easy, to extend the algorithm to handle floating point times. The main reason we use integer times internally is that almost all data set have times specified by integers. Note that the program still follows a continuous-time algorithm in the sense that it does not progress time step by time step.

Contagion. We will assume that every contact represents the same probability β of the contagion (i.e., that a contact spreads the disease). In other words, we model the contagion as a Bernoulli process on the contacts with some additional conditions—a contagion takes place at the first non-zero event after a node pair becomes SI, if the involved nodes are still SI at the time of that event.

Relative to a true epidemic situation, these assumptions are radical simplifications since many effects could cause the transmission probability to vary: The amount of pathogens emitted by different infectious individuals, or at different times by the same individual, can vary greatly [17]. The susceptibility also varies much, not only between people but also e.g., with the time of the day [18, 19]. Finally, our contact sequences do not encode the intensity of the contacts. Clearly, there is a good case for a more complex model of contagion events. The motivation we keep it this simple is the continuity principle.

In empirical data sets, nodes typically can have simultaneous contacts. With the assertion that the contagion is instantaneous in the SIR model, simultaneous contacts become a conceptual problem. A simple solution, and the one we advocate, is to prohibit the infection from spreading further than graph-distance one per time unit of the input data. In other words, if there are contacts (i, j, t) and (j, k, t), but no (i, k, t), then i cannot infect k (via j) at time t. This solution, technically speaking, makes the model a susceptible–exposed–infectious–recovered (SEIR) model where the exposed state lasts a time less than the data's resolution. However, it simplifies the code a great deal and probably makes the simulation more realistic (since SEIR is always more accurate than SIR). Moreover, once again, except for some extreme cases, this decision will not significantly affect the output.

A different principle one could potentially follow would be to assume that the contacts happen at different (continuous) times but that these times have been truncated to integers. Then the principled approach would be to sample the contacts of nominally the same time in random order and average over different realizations of this random sampling [20]. This approach will make more sense if the data set's time resolution is relatively low compared to the propagation of the disease. However, in such a case, one should probably instead consider a static network model since the temporal information would be less critical.

Recovery. For the time to recovery $\delta \geq 0$, we simply follow the standard Markovian SIR model for static networks. I.e., we assume the recovery can be described by a Poisson process and thus we sample δ from an exponential distribution

$$\mathrm{Prob}(\delta) = v \exp\left(-v\delta\right). \tag{1}$$

If one represents times as integers internally, one needs to round the sampled times down to the nearest integer.

The duration of real infections is typically not exponentially distributed [21], so this is not a choice made for realism, but to conform to the standard in static network epidemiology. If one would want the recovery times to follow a particular distribution other than exponential, there is no problem to just replace the exponential random numbers when obtaining a recovery time. (This is unlike static network epidemiology where using a different distribution of recovery rate demands a different algorithm. [22]).

Some papers use a fixed time for the infection duration [11, 20]. This does not simplify anything, it is probably not more realistic either, as realistic distributions of infectiousness tend to be peaked and skewed [23]. Furthermore, they could cause unrealistic threshold effects (when a gap in the contacts is very close to δ). So since there are no major advantages with this approach, exponentially distributed infection times must be preferable.

Number of sources. For several reasons, we recommend starting the outbreak at one node, rather than many. The main reason for this is that medical epidemiology is usually concerned with the outbreak of one pathogen or entering the population, typically via one external (zoonotic) interaction, or arising from a mutation in one host. Starting the epidemics at different sources, one would be assuming that there had been some spreading outside of the considered network. In other words, that one is modeling an open system. In that case, one would also need to model the influx of pathogens during the outbreak, which adds another level of complexity to the problem.

Another conceptual problem with having many sources is how to choose them. Unfortunately, there is no rationale to follow that is both simple and consistent with fundamental epidemiological facts. If one, for example, is modeling bioterrorism, having many seeds could make sense. However, then a modeler needs to know whether the adversary chooses seeds to optimize the damage [24], just at random or in some cluster of the network.

Furthermore, and maybe most importantly, by using many sources, one misses the early die-outs characteristic of epidemic models [20] and presumably also real epidemics. An outbreak typically either dies very early or takes off to follow a predictable curve [25]. With several sources, the early die-offs become inaccessible.

Finally—and related to the previous point—with only one seed, one can measure the basic reproductive number R_0 directly [26]. This is one of the most fundamental epidemiological quantities defined as the expected number of others that the source will infect. Note that one cannot avoid stochastic simulations to calculate this number because neighbors of the source could get infected by other nodes than the source and would not contribute to R_0.

Initialization. Now that we established the need for only one infection source, then how should we choose it? In the spirit of simplicity, we chose it with uniform randomness. This is also related to realism—there might be some correlation between network positions and the chance of acquiring a zoonotic infection. However, without additional knowledge, we cannot do better than choosing it randomly.

In the spirit of simplicity, we also choose the time of the infection uniformly at random between the beginning and end of the contact data set. Introducing the infection at a time related to features of the data—like the beginning of the data or when the seed enters the data—could introduce biases. Since the disease enters from the outside by a process unrelated to the SIR dynamics, we should randomly choose the time. Of course, there is a chance that the outbreak will start toward the end of the data set and thus not have enough time to spread far. Therefore optionally, one could choose the starting time randomly in an early time interval. Nevertheless, there is no simple rule to chose that interval. If the research purpose is to investigate the largest possible outbreaks, one must find such a rule, even though it has undesired consequences. Otherwise, we recommend picking an infection time by uniform randomness in [0, T). For the rest of the paper, we will follow that principle.

Summary. Summarizing the above points, a precise formulation of the SIR model on contact sequences is as follows.

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 δ 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 β.

Algorithm

Now we describe the algorithm. The code, written in C and Python, is available at github.com/pholme/tsir/. This code is commented and written for clarity. Thus, we prioritize to describe the ideas rather than all the algorithmic details. We recommend the reader follow the actual code when reading this section.

Straightforward algorithm. The simplest way of simulating the SIR model on a temporal network is to:

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.

There are many tricks to speed up such a simulation. For example, one can use bisection search to find the first contact capable of spreading the disease (and thereby avoid scanning through all contacts before introducing the infection). Another trick is to note when individuals become inactive and stop the simulations when there are no active contacts. Still, the running time of this algorithm (above the epidemic threshold) will be linear in C. Now, consider the contacts between a pair of nodes. There could be thousands of these, but only one of them can spread the infection. So clearly, if we can identify that particular contact without having to scan through all contacts, that could make the algorithm much faster for denser data sets.

Event-based algorithm. The more elaborate algorithm that we will discuss is inspired by the *event-driven algorithm* for SIR on static networks by Kiss, Miller and Simon [2].

To understand our algorithm, first consider a pair of nodes (i, j) and assume one of them, say i, gets infected at time t_i. Now assume no other nodes can infect j other than i. The infection process between i and j is then a Bernoulli process of a finite number of binary random variables with probability β. (Note that the corresponding part of the event-based algorithm for SIR on static networks is a Poisson process.) The number of Bernoulli random variables is the number $n_{ij}(t_i)$ of contacts between i and j for $t > t_i$. The probability that the k'th such contact will transmit the disease is given by

$$\beta(1 - \beta)^{k-1}. \tag{2}$$

One can sample such a random number k by

$$\left\lfloor \frac{\log(1 - X)}{\log(1 - \beta)} \right\rfloor. \tag{3}$$

where X is a standard, uniformly distributed random variable X on the unit interval $[0, 1]$. Note that the above operations take O($\log c$) time (for a list of c contacts between two nodes), compared to linear time for just scanning through the contacts.

To conveniently handle the above type of computations we store the temporal network internally as follows. First, we represent the temporal network projected to a static network in the standard adjacency list format. (Each node is a C-struct that contains information about its number of neighbors and who those neighbors are.) Then, for every neighbor in the neighbor list there is a sorted list of contacts. See Fig 1 for an illustration and some further details.

Using the above strategy, when a node i gets infected, we can go through its neighbors $j \in \Gamma_i$, and calculate if j could be infected by i, then which one of the contacts between i and j would transmit the disease. In the C code, this happens by calling a subroutine, `conta-gious-contact`, that takes i's infection time t_i and the time-ordered list of contacts between i and j, $\mathbf{t}_{ij}$, as input. Then proceed as follows:

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 Eq 3 to k and call it k'.
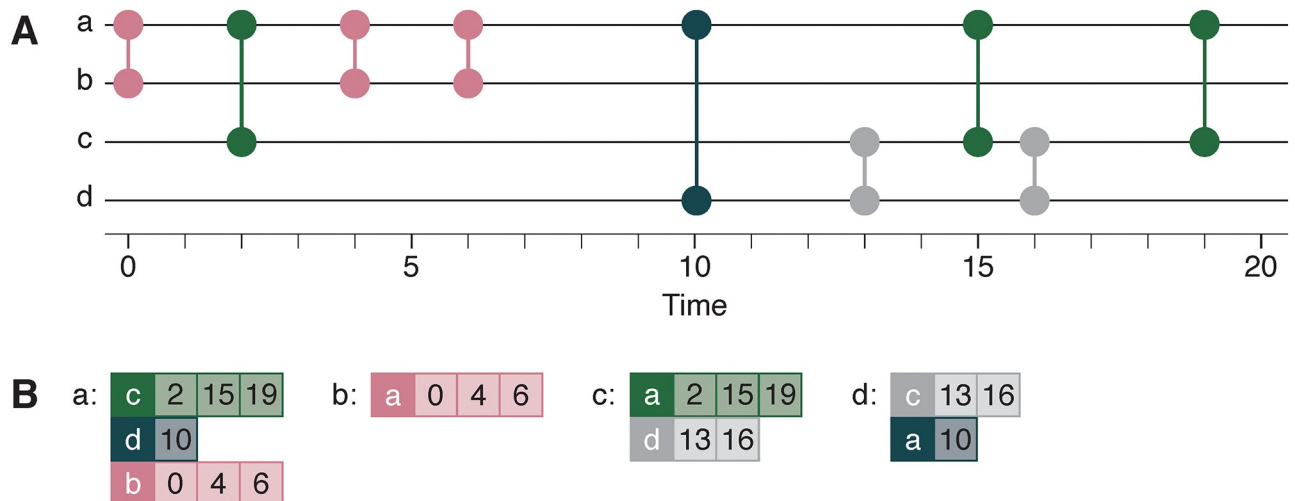
Fig 1. Internal representation of the temporal network. In this figure, panel B illustrates how the temporal network in panel A is represented internally. In A, we display a temporal network of four nodes (a, b, c, d), four edges ((a, b), (a, c), (a, d), (c, d)) and nine contacts ((a, b, 0), (a, c, 2), etc.). In panel B, the internal representation is organized by nodes. Every node has a list of neighbors (e.g., node a has the neighbor list (c, d, b)). For every neighbor, there is an ordered list of times of contacts with that neighbor (e.g. a has contact with c at times (2, 15, 19)). The neighbor lists are ordered in decreasing order of the last contact with that neighbor. This makes it possible to break iterations over neighbors if the infection time of a node is later than the last contact with the neighbor.

https://doi.org/10.1371/journal.pone.0246961.g001

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.

From the previous section, we can see that our code needs a priority queue—a data structure where one can quickly delete the smallest element and insert arbitrary elements. There are many ways to implement a priority queue. For our situation (where we have to delete, update, and add elements), operating a priority queue of length n has at least a complexity of O(log n). Among algorithms with this complexity, we use perhaps the simplest one—a *binary heap*. Apart from its simplicity, one appealing feature of using a binary heap for this problem is that updating the entry for an infected node already on the heap is very fast. Briefly speaking, updating a heap needs two types of sorting operations—`heap-up` and `heap-down`—where `heap-up` is much faster, and the only one needed to update elements already in the heap. We only use `heap-down` when we delete the smallest element.

The core of the code happens in a subroutine called `infect` that handles the infection of one node:

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 `contagious-contact`).

   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.

A trick to speed the code up is to sort the neighbor list in decreasing order of time of the last contact—see Fig 1. In that way, we can break the iterations over neighbors (step 2) whenever we encounter a neighbor with which *i* has no future contacts).

Then the final structure of the program is simply:

1. Read the network and initialize everything.

2. Infect the source node.

3. While there are any nodes left on the heap, call `infect`.

4. Reset the simulation.

5. Go to 2 until you have enough averages.

6. Evaluate the output.

Further notes about the implementation. Our implementation of the above algorithm, available at github.com/pholme/tsir/, uses a mix of C and Python. The idea is to exploit C's speed for the core routines and the many libraries of Python to simplify the pre- and post-processing. We have not made this into a full Python library because research building on this codebase would most likely need to add functionality on a low level. We have refrained from adding many imaginable measurements, both because it is hard to envision a sufficiently complete list of such, and it would slow down the program. We display an example output of the program in Fig 2.
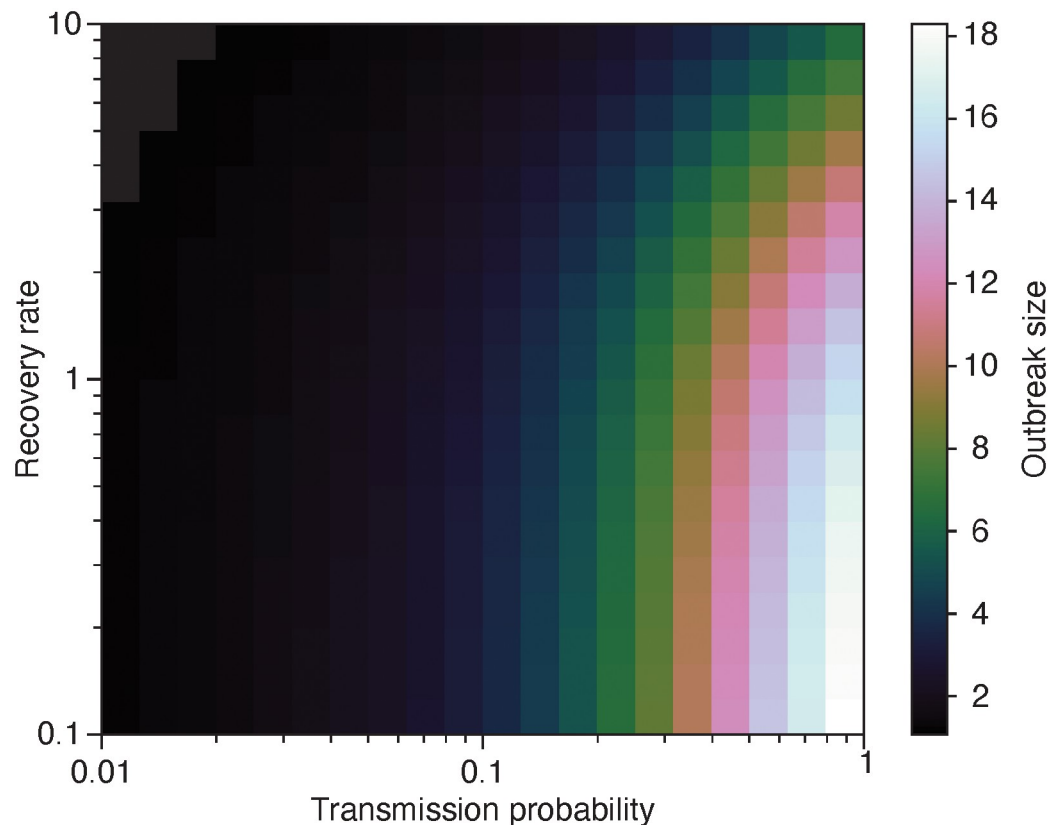
Fig 2. Example output. This heatmap shows the average outbreak size as a function of the model parameters. The raw data comes from the first day of sampling in Ref. [27]. It represents the proximity patterns of visitors to an art gallery.

https://doi.org/10.1371/journal.pone.0246961.g002

We use a 64-bit state version of the PCG (Permuted Congruential Generator) random number generator [28]. For this type of simulation, neither speed nor statistical quality of the random number generation is critical. For simplicity, we could just have used some lower-performance, library generator. Still, in the spirit of using state-of-the-art components, we opt for PCG. For some parameter values, it does save a few percent of computing time compared to popular random number generators of the previous generation (i.e., the Mersenne Twister).

Results

In this section, we go over some analysis of our temporal-network simulation program.

Validation

We validate our event-based program by consistency checks in several ways. In this section, we will discuss some such checks—validation against the straightforward implementation and the analytical solution of the standard Markovian SIR model on static networks. As mentioned above, if the contacts are generated by a Poisson process on the edges—i.e., if they have exponentially distributed times between the contacts—then SIR on a temporal network and static network simulations should give the same results.

For this test, we use the graph shown in Fig 3A. This graph has a complex behavior with respect to the SIR model. It is also small enough to solve exactly [29]—the expected outbreak
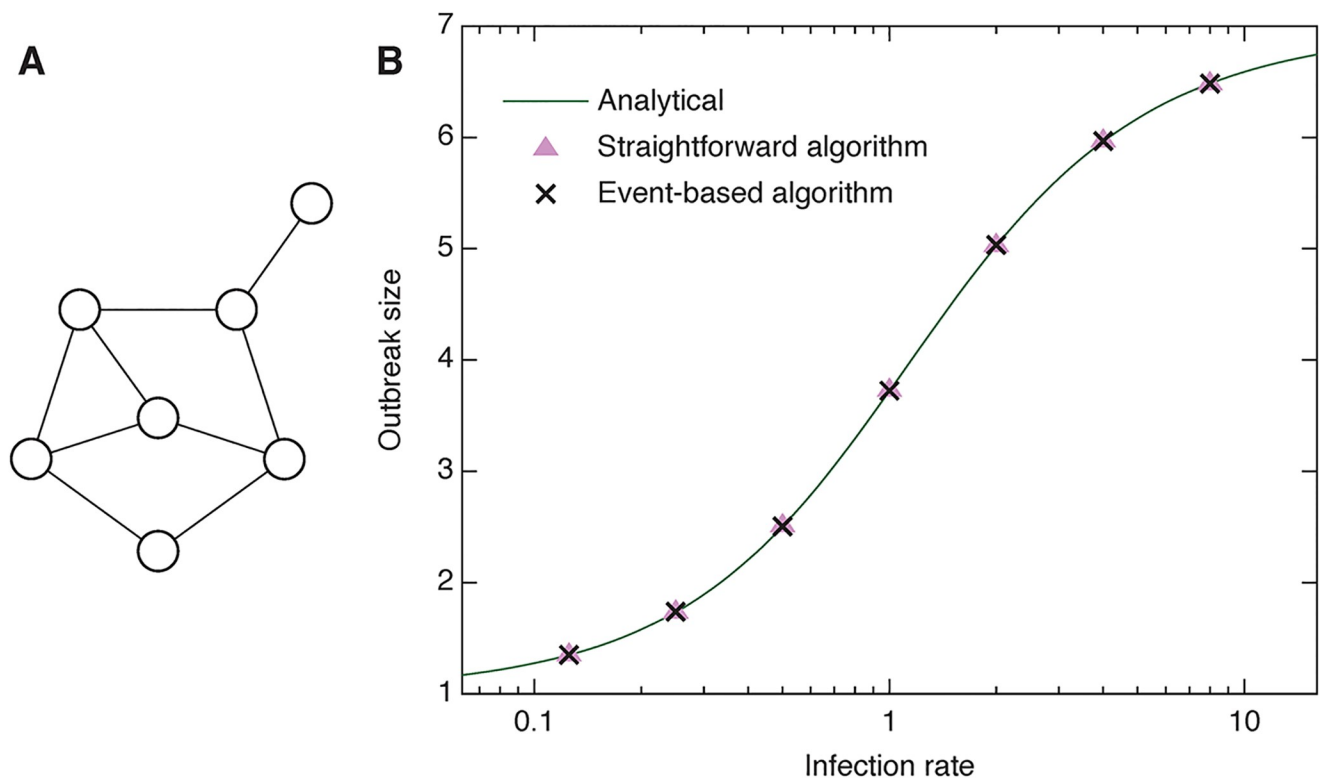
Fig 3. Validation of the program. Panel A shows a small graph with especially complex behavior with respect to the SIR model (and thus a good test case). Panel B shows the predicted outbreak size for the graph in A. The solid curve is the analytical solution. The symbols represent averages over 10^6 values for the straightforward and event-based algorithms, respectively.

https://doi.org/10.1371/journal.pone.0246961.g003

size Ω as a function of β is

$$\Omega(\beta) = \frac{1176\beta^{10} + 8540\beta^9 + \cdots + 123\beta + 7}{168\beta^{10} + 1316\beta^9 + \cdots + 105\beta + 7}. \qquad (4)$$

We add contacts to this graph's edges by drawing exponential random numbers with the rate parameter one. We break these time series when they are longer than $\tau = 1000$ (arbitrary units). This gives an expected number of 1000 contacts per edge. Since our code reads integer time stamps, and we want as high resolution as possible, we rescale the times so that $T = 2^{32} - 1$ which guarantees there is no 32-bit integer overflow. Note that there is a trade-off: If we have too many contacts per link, then the distribution of inter-event times gets further from exponential. If we have too few contacts per link, there is a higher chance the outbreak will not die by the end of the data set. We use $v = 1$ and average over 10^6 runs of the algorithms and 100 realizations of the generation of time-stamps. Even with these caveats, Ω from the temporal network simulation is statistically indistinguishable from the exact values from the static network version (standard scores between 0.5 and 2). Furthermore, the straightforward algorithm and the event-based algorithm are also indistinguishable.

Time complexity

The worst-case complexity of the event-driven algorithm is $O(n^2 \log n \log C)$ for a dense network (where C is the number of contacts). Each node has to enter and exit the priority queue (a factor $n \log n$). For an infected node, all the neighbors need to be scanned (another factor n). Then for each neighbor, the infecting contact needs to be identified (in a worst case this has a complexity $\log C$). Most real networks of interest are sparse—i.e. the degrees are bounded, giving the complexity $O(n \log n \log C)$—and has $C \gg n$.

The straightforward implementation is $O(C + n)$ in a worst case, and should be slower as long as C is sufficiently large. Conversely one could construct temporal networks where the straightforward algorithm is faster—make a list of one contact per node pair for all node pairs, then repeat the same list after the first. Is such a temporal network the number of links nodes is maximal and the number of contacts per link is low. Furthermore, every node is reachable from every other. This should be a case where the straightforward algorithm outperforms the event-based one. However, empirical temporal networks typically look very different with very large C values and relative low reachabilities—i.e. many node pairs are unreachable due to the constraint that paths have to follow increasing time stamps reduces the outbreak sizes, so that a large part of the network will never be reached even in a worst case. In the event-based simulations, the program does not need to evaluate contacts in these unreachable parts of the networks, which contributes to its speed in practice. Many other factors affect the running time. For example, the earlier the outbreaks die, or the smaller they get, the shorter are the execution times. For these reasons, it is challenging to make a complete theory of these algorithms' relative running times for practical parameter values.

Evaluation

To evaluate the speed of our event-based algorithm, we use artificial temporal networks. We generate these in a similar way to the ones used to check the limit to the static network SIR model described in the previous section. The difference is that we here use random graphs—the standard Erdős-Rényi or $G(n, p)$ model [30]—and Barabási-Albert models as the underlying structure. Then we put time series of contacts, with inter-event times drawn from an exponential distribution, on the edges.
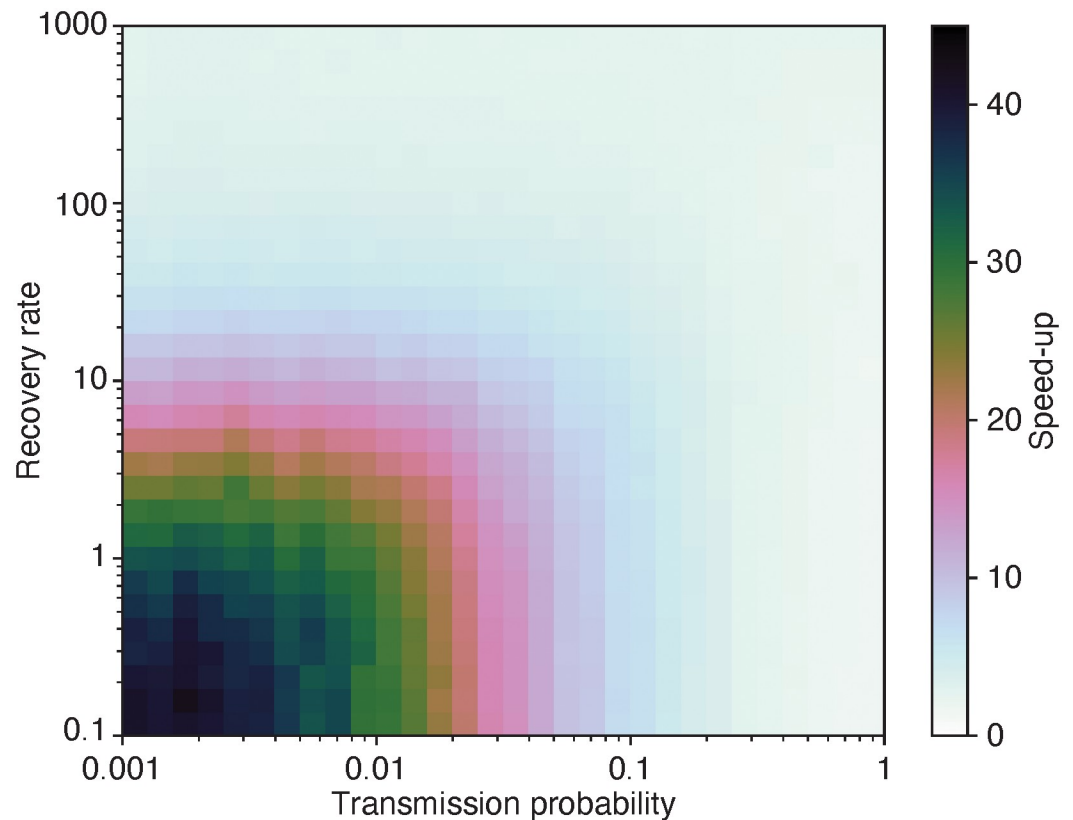
Fig 4. Speed-up relative to the straightforward algorithm for an artificial network as a function of the SIR model parameters. How many times faster the event-driven program is compared to the reference code for the same data set as in Fig 2. The minimum value of the speed-up in this figure is 2.8.

https://doi.org/10.1371/journal.pone.0246961.g004

We compare our event-driven algorithm to the straightforward method. For a fair comparison, we employ all simple optimizations that we can think of for both programs—such as bisection search to find the earliest contact after the beginning of the epidemics. We report the times of the disease simulation, i.e. excluding the time to read the data and fill up the data structures (which is not the bottleneck of the program).

In our first experiment—see Fig 4—we check the relative speed-up for the same data set as in Fig 2. We note that the event-based algorithm always outperforms the straightforward one, although the region of the parameter space where the speed-up is larger than around 3 is not that large. This region is at small transmission probabilities and small recover rates, i.e. the disease does not spread much, still it does not die out. In this case, the straightforward algorithm still has to go through all contacts, whereas the event-driven algorithm just has to go through the few ones that get infected.

In our second experiment, we use 10^3 temporal networks for averages, and 10^6 outbreak simulations per network. We chose the parameter values in such a way that the outbreak sizes should be intermediate. In Fig 5, we show the speed-up—the execution time of the straightforward implementation divided by the time of the event-driven simulation. We ran the simulations on a workstation with dual AMD EPYC 7552 CPUs, 256Gb RAM memory and 192 logical cores (at least half were idle during the experiment).

As predicted, more contacts per edge increases the advantage (Fig 5A). To keep the total number of contact the same, we let the average degree be $z = 1024/c$. This means that for the
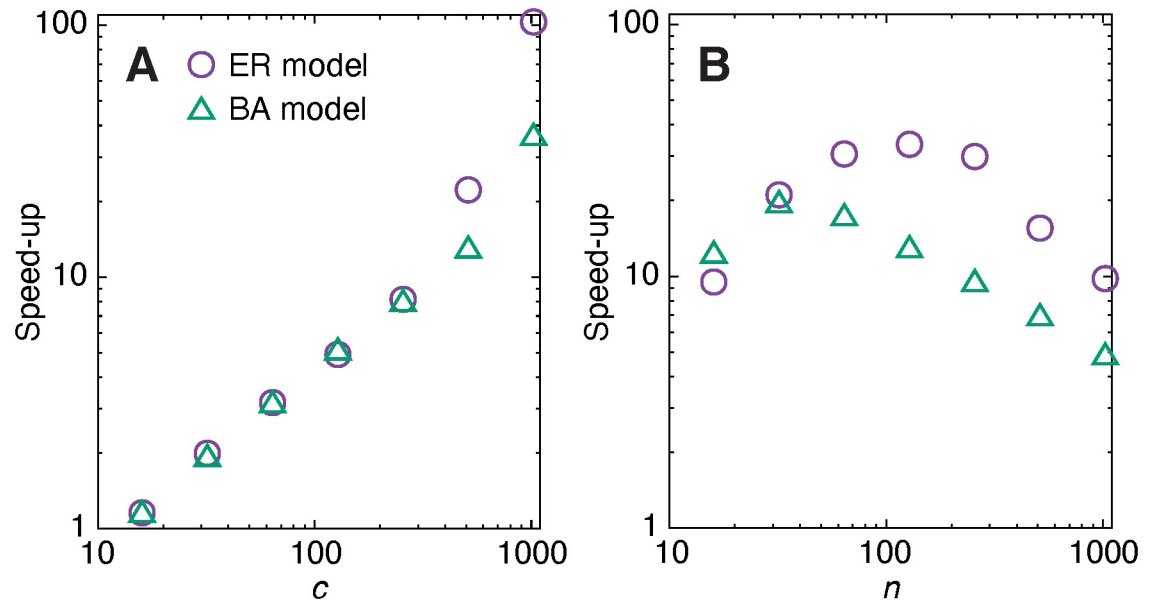
Fig 5. Speed-up relative to the straightforward algorithm of artificial networks. How many times faster the event-driven program is compared to the reference code. The underlying temporal networks are Erdős-Rényi—ER, binomial random graphs, $G(n, z/n)$ (where z is the average degree—for Barabási-Albert (BA) model networks [30]. We add exponential inter-event times to these static graphs ($\lambda = 1/c$) added until T is at least one. (Where λ is the usual, "rate parameter," of the exponential distribution.) We use 10^3 temporal networks and 10^6 outbreak runs per set of parameter values. The recovery rate is $v = 1$. Error bars (standard errors) would have been smaller than size of the symbols and thus not shown. Panel A shows the scaling of the speed-up as a function of the average number of contacts per link. Here $n = 128$ and $z = 1024/c$ (to keep the number of contacts constant). Panel B displays the speed-up as a function of n. Here $c = 512$ and $z = 2$.

largest values of c of the Erdős-Rényi model, the networks are fragmented, which explains the increase in the speed-up relative to the BA model (which never is fragmented). For fragmented networks, the event-based algorithm never needs to deal with connected components other than the one where the disease starts, which contributes to its speed.

In Fig 5B we see that the speed-up has intermediate peaks both for the Erdős-Rényi and the Barabási-Albert models. Why the relative speed-up decays for larger n is hard to say. It is important to notice that the practical run times cannot be explained by any single parameter, not even output quantities like the average outbreak size or time to extinction. Rather the practical run times depend both on the progression of the simulated outbreak and properties of the network that do not affect the disease spreading. The most important message this study is the event-driven algorithm can in theory be arbitrarily faster than the straightforward one, and in practical situations it is, at least for some parameter values and data sets, faster. If computational speed is of utmost importance, we recommend a probe of the running times with the two algorithms. Code for such a comparison is available here https://github.com/pholme/tsir_eval/.

Discussion and conclusion

We have derived a principled detailed formulation of the SIR model on temporal networks and presented an, open-source simulation code for an event-based simulation of this model. We also a give platform for comparing the speed of the event-based algorithm with a more straightforward version (running through all contacts in order of time).

The event-based algorithm that we proposed can be extended to many other compartmental models. As long as individuals do not reenter the susceptible state, it should be quite

straightforward to extend our code. This would cover e.g., the SEIR model [3]. The only significant difference would be that one needs to put different types of events in the priority queue. For models like the SIS, where individuals can become susceptible again, it might be hard to write efficient event-driven code. For example, in this case, one can no longer discard potential infection events because they happen later than others. Probably other ideas for fast epidemic simulations could work better in this case [31].

Another direction for future research would be to sacrifice some of the principles that we recommended to further increase the speed. One could, for example, use a fixed duration of the infectious stage [11]. In such a case, the gaps between the contacts determine whether an edge could transfer the infection. This, we speculate, could open for other types of fast algorithms. In general, temporal networks opens many intriguing problems for algorithm design. We recommend Ref. [32–34] for further inspiration.

Acknowledgments

Author Contributions

Conceptualization: Petter Holme.

Formal analysis: Petter Holme.

Funding acquisition: Petter Holme.

Investigation: Petter Holme.

Methodology: Petter Holme.

Project administration: Petter Holme.

Resources: Petter Holme.

Software: Petter Holme.

Validation: Petter Holme.

Visualization: Petter Holme.

Writing – original draft: Petter Holme.

Writing – review & editing: Petter Holme.

References

1. Hethcote HW. The mathematics of infectious diseases. SIAM Rev. 2000; 42(4):599–653. https://doi.org/10.1137/S0036144500371907

2. Kiss IZ, Miller JC, Simon PL. Mathematics of Epidemics on Networks. Cham: Springer; 2017.

3. Salathé M, Kazandjieva M, Lee JW, Levis P, Feldman MW, Jones JH. A high-resolution human contact network for infectious disease transmission. Proc Natl Acad Sci USA. 2010; 107(51):22020–22025. https://doi.org/10.1073/pnas.1009094108 PMID: 21149721

4. Masuda N, Holme P. Temporal Network Epidemiology. Springer: Singapore; 2017.

5. Masuda N, Holme P. Predicting and controlling infectious disease epidemics using temporal networks. F1000 Prime Rep. 2013; 5:6.

6. Rocha LE, Liljeros F, Holme P. Simulated epidemics in an empirical spatiotemporal network of 50,185 sexual contacts. PLoS Comput Biol. 2011; 7(3):e1001109. https://doi.org/10.1371/journal.pcbi.1001109

7. Barrat A, Cattuto C, Kivelä M, Lehmann S, Saramäki J. Effect of manual and digital contact tracing on COVID-19 outbreaks: a study on empirical contact data; 2020.

8. Schirdewahn F, Colizza V, Lentz HH, Koher A, Belik V, Hövel P. Surveillance for outbreak detection in livestock-trade networks. In: Temporal Network Epidemiology. Springer; 2017. p. 215–240.

9. Sapiezynski P, Stopczynski A, Lassen DD, Lehmann S. Interaction data from the Copenhagen networks study. Sci Dat. 2019; 6:315. https://doi.org/10.1038/s41597-019-0325-x

10. Holme P, Saramäki J. Temporal networks. Phys Rep. 2012; 519(3):97–125. https://doi.org/10.1016/j.physrep.2012.03.001

11. Lee S, Rocha LEC, Liljeros F, Holme P. Exploiting temporal network structures of human interaction to effectively immunize populations. PLOS One. 2012; 7(5):e36439. https://doi.org/10.1371/journal.pone.0036439

12. Starnini M, Machens A, Cattuto C, Barrat A, Pastor-Satorras R. Immunization strategies for epidemic processes in time-varying contact networks. J Theor Biol. 2013; 337:89–100. https://doi.org/10.1016/j.jtbi.2013.07.004

13. Takaguchi T, Sato N, Yano K, Masuda N. Importance of individual events in temporal networks. New J Phys. 2012; 14(9):093003. https://doi.org/10.1088/1367-2630/14/9/093003

14. Andersson H, Britton T. Stochastic Epidemic Models and Their Statistical Analysis. New York: Springer; 2012.

15. Cattuto C, Van den Broeck W, Barrat A, Colizza V, Pinton J, Vespignani A. Dynamics of Person-to-Person Interactions from Distributed RFID Sensor Networks. PLOS ONE. 2010; 5(7):e11596. https://doi.org/10.1371/journal.pone.0011596

16. Böttcher L, Antulov-Fantulin N. Unifying continuous, discrete, and hybrid susceptible-infected-recovered processes on networks. Phys Rev Research. 2020; 2:033121. https://doi.org/10.1103/PhysRevResearch.2.033121

17. VanderWaal KL, Ezenwa VO. Heterogeneity in pathogen transmission: mechanisms and methodology. Funct Ecol. 2016; 30(10):1606–1622. https://doi.org/10.1111/1365-2435.12645

18. Bass J, Lazar MA. Circadian time signatures of fitness and disease. Science. 2016; 354(6315):994–999. https://doi.org/10.1126/science.aah4965

19. Colman E, Spies K, Bansal S. The reachability of contagion in temporal contact networks: how disease latency can exploit the rhythm of human behavior. BMC Infect Dis. 2018; 18(1):219. https://doi.org/10.1186/s12879-018-3117-6

20. Holme P. Information content of contact-pattern representations and predictability of epidemic outbreaks. Sci Rep. 2015; 5:14462. https://doi.org/10.1038/srep14462

21. Vergu E, Busson H, Ezanno P. Impact of the infection period distribution on the epidemic spread in a metapopulation model. PLOS One. 2010; 5(2):e9371. https://doi.org/10.1371/journal.pone.0009371

22. Masuda N, Rocha LE. A Gillespie algorithm for non-Markovian stochastic processes. SIAM Review. 2018; 60(1):95–115. https://doi.org/10.1137/16M1055876

23. Krylova O, Earn DJD. Effects of the infectious period distribution on predicted transitions in childhood disease dynamics. J Roy Soc Interface. 2013; 10(84):20130098. https://doi.org/10.1098/rsif.2013.0098

24. Jankowski J, Szymanski BK, Kazienko P, Michalski R, Bródka P. Probing limits of information spread with sequential seeding. Sci Rep. 2018; 8:13996. https://doi.org/10.1038/s41598-018-32081-2

25. Janson S, Luczak M, Windridge P. Law of large numbers for the SIR epidemic on a random graph with given degrees. Random Struct Algor. 2014; 45(4):726–763. https://doi.org/10.1002/rsa.20575

26. Holme P, Masuda N. The basic reproduction number as a predictor for epidemic outbreaks in temporal networks. PLOS One. 2015; 10(3):e0120567. https://doi.org/10.1371/journal.pone.0120567

37. Van den Broeck W, Quaggiotto M, Isella L, Barrat A, Cattuto C. The making of sixty-nine days of close encounters at the Science Gallery. Leonardo. 2012; 45(3):285–285. https://doi.org/10.1162/LEON_a_00377

28. O'Neill ME. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. Claremont, CA: Harvey Mudd College; 2014. HMC-CS-2014-0905.

29. Holme P. Three faces of node importance in network epidemiology: Exact results for small graphs. Phys Rev E. 2017; 96(6):062305. https://doi.org/10.1103/PhysRevE.96.062305

30. Newman MEJ. Networks: An Introduction. Oxford UK: Oxford University Press; 2010.

31. St-Onge G, Young JG, Hébert-Dufresne L, Dubé LJ. Efficient sampling of spreading processes on complex networks using a composition and rejection algorithm. Comput Phys Commun. 2019; 240:30–37. https://doi.org/10.1016/j.cpc.2019.02.008

32. Badie-Modiri A, Karsai M, Kivelä M. Efficient limited-time reachability estimation in temporal networks. Phys Rev E. 2020; 101:052303. https://doi.org/10.1103/PhysRevE.101.052303

33. Himmel AS, Bentert M, Nichterlein A, Niedermeier R. Efficient Computation of Optimal Temporal Walks Under Waiting-Time Constraints. In: Cherifi H, Gaito S, Mendes JF, Moro E, Rocha LM, editors. Complex Networks and Their Applications VIII. Cham: Springer; 2020. p. 494–506.

34. Petrovic LV, Scholtes I. Counting causal paths in big times series data on networks; 2019.