Definitions:

Different types of temporal graphs

we classify the graphs not based on their mathematical characteristics but based on the type of data set we are dealing with.

1. the snapshot type: in this model, the temporal graph is actually described as a set of static graphs, which appear at different times (usually with a constant time step).

As stated above, it isn't important if the static graphs are correlated or not, the property we currently care about is the format that the temporal graph is implemented.

Example(each edge is described as to nodes and two dashes between them.):

t=1:
1- -2
1- -3
t=2:
1 - - 3
2 - - 4

* at t = 1, node 1 is connected to nodes 2 & 3; at t = 1, node 1 is connected to 3 and 2 is connected to 4.

2. interval link: in this model, we have 2 time stamps for respectively appearance and disappearance of each links.

Example: (the plus character, describes the appearance and the minus means disappearance)

t=1:
+ 1 - - 2
t=2:
+ 1 - - 3
t=3:
- 1 - - 3
+ 2 - - 4
…

3. link sequence: in this type, every node is indicated by only 1 timestamp.

The link is "on" only in the specified time, and is "off" both before and after that time. (the node can be turned on several times.)

the link sequence can be described as a limit of interval link type, which the lifetime of every node is equal with the minimum time step of the system.

Different types of dynamics

in this report we define two kinds of dynamics, the dynamic "on" the system which describes the transition of the nodes between the states of SIR model and the "dynamic "of" the system which describes the temporal behavior of the graph structure itself, we call these dynamics respectively "dynamic of" and "dynamic on".

Algorithms:

the abstract model is described in the previous reports so we only discuss the structure of the two different algorithms that we have introduced.

1. Rejection based model

this model operates on time steps with equal length and is best implemented for the snapshot temporal graph.
Every agent has two properties, the “current health” and the “future health”. “current health” is responsible for the diseases that the agent can supply for (transfer to) its neighbors, and the future health is the responsible for the interactions that the agent demands (the diseases that it is vulnerable to).
When the “update” function is called, the current health is set equal to the future health.

Note: the reason for the distinction between these properties, is that the time step (whether it’s actually a day or an hour or ...) is the shortest possible time interval of our model. And so it cannot be cut into smaller pieces. Therefore a node cannot get infected by the noon and become an spreader by the evening, because this type of mindset would introduce two parts of the time step, the time before the infection of the described node, and the part after it. Hence, all the updates in a single time step should be simultaneous.

In every time step these set of steps are taken:
1. all the possible interactions ($S \rightarrow I$ transitions) for each node are done based on the probability (p or q), updating their future health.

2. the past health of each node is set equal to future health.

3. all the previously active nodes (the spreader nodes of the finished time step.) go from I to R by 100% chance.

1. Gillespie Algorithm

Action List

we define an action list consisting of all the possible transitions. Every action has a relative probability to occur, for the “first-time-infections”, the “second-time-infections” and the “recoveries” the relative probabilities are respectively p, q & r.

Static graphs

Suppose that we only have one possible action in our system, in the rejection based model we would have to check in every time step, if that transition happens our not, so we were supposed to run the program for many time steps (specially if the probability of that action is low.)
but based on the theory, the waiting time for an event with uniform probability is described by the Poisson distribution function (k=0) so instead of the previous method we can directly predict the time of occurrence of the next action. (the probability which after time t the mentioned node hasn’t taken place.)
Note: the probability doesn’t need to be globally uniform, being locally uniform (until the next action happens) would suffice.

1. $S_m(\tau) = e^{-\lambda_m \times \tau}$

which λ_m is called the transition rate.
By doing so, we can calculate the desired time, obtained by the random exponential distribution (Poisson) and directly jump to that time. (applying the action on that time.)

but what happens if we have not only one possible action, but many of them?
In this situation we need another waiting time which is the “cumulative waiting time”.
We calculate the probability which after time t, none of the actions have taken place.

$$2. \qquad S(\tau)=\prod_m S_m(\tau)=e^{-\sum_m \lambda_m \times \tau}=e^{-\Lambda \times \tau}$$

This new distribution is the waiting time for any "action" which means how much should we wait until "something" happens, and Λ is called the cumulative transition rate.
Now that we have the time for the next action, we should decide "which" action it is. This decision is made by using another random distribution based on the weighted probability distribution of the current possible actions (using the relative probability of each action).

"Dynamic of" the graph

1. In the previous section our list of possible actions only changed when an action happened. For example when a node is turned I→R then the action corresponding to this transition is removed, and the node couldn't infect any of it's S state neighbors anymore so their corresponding actions would be deleted too.
therefore we were able to calculate the time for the next action based on the current list, and only update the list after an action happened.

2. if our temporal graph were adaptive and therefore the "dynamic of" only depended on the "dynamic on" there would be no problem with the previous algorithm, the action list was still stable until an action took place, where we would update the action list and "cumulative waiting time" anyways.

3. in the case of a non-adaptive temporal network the action list has to be updated both after an action (dynamic-on) and after a "dynamic-of" the graph.
Which means that for our current purpose we have a new kind of update for action list which happens when edges get add or removed, because it may add or remove actions to our list.
The problem introduced here is not that a particular transition which we plan to apply, may get deleted from the action list and become obsolete, because *we don't decide which action is going to take place, until we get to the "action time" obtained by equation 2.* and the action which we are going to apply, will be chosen from the list of possible actions at that particular time.
But the problem is that after a "dynamic-of" *the probability of occurrence of the actions inside our action list may cease to be locally uniform.* for example, if a particular edge gets removed, the corresponding infection's probability rate will change to zero, therefore we cannot derive equations 1 and 2.

to solve this issue, we introduce the following steps:
1. we draw the next action time τ based on the cumulative waiting time distribution $e^{-\Lambda_1 \times \tau}$.

2. we call the time step for the next "dynamic-of" t_1 .

3. if $\tau < t_1$ then we move to the time τ and set $t_1 = t_1 - \tau$ and then we go to line 1 again.

4. else if $\tau > t_1$ (the root of our problem) we know that after t_1 the cumulative transition rate changes from Λ_1 to Λ_2 . now the elapsed part of the waiting time is t_1 and the "remaining part" of it is $\tau - t_1$. now there are two factors we want to respect in order to find the actual time for the next action.
First: the new cumulative rate Λ_2 .
Second: the remaining part of the waiting time.

For considering these two factors, we take into account that the numbers drawn by the $e^{-\Lambda_2\times\tau}$ are by average $\frac{\Lambda_1}{\Lambda_2}$ times the numbers drawn by $e^{-\Lambda_1\times\tau}$. This proposition can be verified by considering the formula which the random exponential numbers are constructed from the uniform random distribution by the computer.
If r is a number produced with a random uniform distribution, then for creating the exponential distribution with parameter Λ_2 we calculate $1/\Lambda_2 \ln(1/r)$.
now when our cumulative transition rate changes from Λ_1 to Λ_2 we just take the remaining part of the waiting time $\tau - t_1$ and multiply it by $\frac{\Lambda_1}{\Lambda_2}$ and set $\tau = \frac{\Lambda_1}{\Lambda_2}\times(\tau - t_1)$ now we have used the remaining fraction of the waiting time, but this remaining fraction is elapsed by a new rate Λ_2.
For example we consider $\Lambda_1 = 1$ *and* $\Lambda_2 = 2$ *. Also we assume that the initial random* τ *is drawn equal to 1, and that the time for the next "dynamic-of" is* $t_1 = 0.6$ *so after 0.6 we have to enter step 4 of the algorithm, the new* τ *is set to* $\tau = \frac{\Lambda_1}{\Lambda_2}\times(\tau - t_1)$ *which means* $\tau = \frac{1}{2}\times(0.4) = 0.2$ *so the remaining part is divided by the factor of 2.*

After these sub-steps, we then calculate the time for the next "dynamic of" and set t_1 equal to that time.
This is the end of step 4, we now go back to line 3.

in the supplementary we have compared the different results obtained by the two algorithms.

Advantages of the Gillespie algorithm:

1. the algorithm is more efficient because it's time complexity is of order n, but the complexity of the rejection based algorithm is of order n^2.
2. the rejection based algorithm is correct only for the the limit of very small time steps, but the Gillespie algorithm is "stochastically exact".
3. Gillespie algorithm can be modified to use for the all three kind of temporal graphs introduced in section 1.