Spread and defend infection in graphs

Arya Tanmay Gupta*

Computer Science and Engineering, Michigan State University

Abstract

The spread of an infection, a contagion, meme, emotion, message and various other spreadable objects have been discussed in several works. Burning and firefighting have been discussed in particular on static graphs. Graph burning simulates the notion of the spread of “fire” throughout a graph (plus, one unburned node burned at each time-step); graph firefighting simulates the defending of nodes by placing firefighters on the nodes which have not been already burned while the fire is being spread (started by only a single fire source).

This article studies a combination of firefighting and burning on a graph class which is a variation (generalization) of temporal graphs. Nodes can be infected from “outside” a network. We present a notion of both upgrading (of unburned nodes, similar to firefighting) and repairing (of infected nodes). The nodes which are burned, firefighted, or repaired are chosen probabilistically. So a variable amount of nodes are allowed to be infected, upgraded and repaired in each time step.

In the model presented in this article, both burning and firefighting proceed concurrently, we introduce such a system to enable the community to study the notion of spread of an infection and the notion of upgrade/repair against each other. The graph class that we study (on which, these processes are simulated) is a variation of temporal graph class in which at each time-step, probabilistically, a communication takes place (iff an edge exists in that time step). In addition, a node can be “worn out” and thus can be removed from the network, and a new healthy node can be added to the network as well. This class of graphs enables systems with high complexity to be able to be simulated and studied.

Keywords: variable burning, variable firefighting, temporal graphs, variable nodes, variable edges

1 Introduction

Several models based on discrete mathematics, probability and complex calculus have been used to demonstrate the spread of an infection or a contagion in a network of hosts, a human social network, or other biological network.

*atgupta@msu.edu
Graph burning is a process introduced on static graphs [6].

**Definition 1. Graph Burning.** Initially, all nodes are marked as “unburned”. Then in each time-step, (any) one unburned node is burned from “outside”, and then the fire spreads to the neighbouring nodes from the nodes which were burnt until the previous time-step. This process continues until all the nodes are burned.

Firefighting is another process which was introduced on static graphs [24].

**Definition 2. Graph Firefighting.** Fire is initiated from (any) one node in the first time-step. From the second time-step, at each time-step, a firefighter is placed on an unburned node, the fire spreads to all nodes neighbouring the nodes which were burned till the last time-step, except that a firefighted node cannot be burned. This process stops when fire cannot spread to any new nodes.

Both firefighting and graph burning have been verified as NP-Hard problems.

In this article, we extend our work in [22, 23], and study a model in which graph burning and firefighting are used against each other. We study more sophisticated versions of burning and firefighting in which we burn an arbitrary number of nodes from outside, and we firefight on an arbitrary number of nodes in each time-step, both of which are done probabilistically. The spread of an infection and the choice of upgrading/repair of nodes is done probabilistically. Further, both burning and firefighting is not permanent on the nodes. In addition, the graph that we study is not static. These modifications to the contemporary definitions of burning and firefighting on the model presented in this article have been done to be able to efficiently model several real-world systems.

In the literature, a temporal graph is defined as follows.

**Definition 3. Temporal Graph.** A temporal graph \( G = (V, E_1, E_2, \ldots, E_\ell) \) defined by a static set of nodes \( V(G) \), and a sequence \( G_1, G_2, \ldots, G_\ell \) of graphs which have the same node set as \( G \), but for any graph \( G_i \) (1 \( \leq \) i \( \leq \) \( \ell \)), \( G_i = (V, E_i) \).

As per the above definition, i corresponds to the \( i^{th} \) time-step, and as the name suggests, temporal graphs were initially introduced to simulate the graphs that change with time. A temporal graph \( G \) can be viewed as a graph which has constant number of nodes, but a sequence of (not necessarily) distinct sets of edges on \( V(G) \). We call \( G_i \) an instance of \( G \). We study a model which presents a fusion of probability based graph burning and firefighting on a variation of temporal graphs: in addition to how temporal graphs have been defined, we allow the number of nodes be modified in each time-step.

The structure of the article is as follows. The structure of the subject class of graphs that we study is discussed in Section 2. Section 3 contains the preliminaries. In Section 5, we study the variable burning, in Section 6 we study variable burning, in combination with variable repair and upgrade, and in Section 7 we study variable burning, repair and upgrade along with allowing the
nodes to be inserted and deleted. In Section 8, we introduce variable edge probability in nodes. In Section 9, we discuss some interesting modifications that can be done while simulating certain real-time systems. In Section 10, we discuss the related work in the literature, and we conclude ourselves in Section 11.

2 Structure of the subject class of networks

The model described in this article studies a network of nodes where we emphasize on the spreading of an infection and the defence against it. We reduce our discussion to the same in this article on an arbitrary graph $G$. In the model that we are going to present, the input is a graph $G$ with a defined set of nodes. Associated with the graph $G$, there are some variables that we define in Table 1 (page 3). Values to all the variables in Table 1 are provided as part of the input.

| Variable (associated with $G$) | What it represents |
|-------------------------------|-------------------|
| $\rho_{\text{del}}$ | probability of deletion of an infected node |
| $\rho_{\text{ins}}$ | probability of insertion of a new node to $G$, denotes the number of nodes being added in each time-step as the ratio to the contemporary number of nodes. |

Table 1: Variables associated with $G$

This is different than the traditional temporal graphs because we allow insertion and deletion of nodes as well.

Each node $v$ in the graph has some associated variables which we describe in Table 2 (page 3). Values to all the variables in Table 2 are not provided as part of the input, except for $v.\rho_c$ and $v.\text{type}$. The default initial values for the rest of the variables are discussed in Section 3.

| Variable | What it represents |
|----------|--------------------|
| $v.\rho_c$ | the probability that $v$ is connected to any other node at any time-step. |
| $v.i_s$ | true iff $v$ is infected (infection status). |
| $v.\text{type}$ | denotes the type of $v$, just for book-keeping. |
| $v.e_s$ | true if $v.i_s$ is true and the infection in $v$ is evident and has been reported (infection evidence status). |
| $v.t_e$ | denotes the time-step in which $v.e_s$ was last flipped to true (from false). |
| $v.t_r$ | the time-step when $v$ was repaired (after getting infected, and then getting reported). |
| $v.t_u$ | the time-step when $v$ was upgraded. |

Table 2: Variables associated with each node $v$ in $V(G)$
We also use some variables that are globally accessible, are commonly applicable to all the nodes, but are constant for each node in $G$, so we do not associate them with any node and we assume that a single copy of these variables will be used by all the nodes. We define these variables in Table 3 (page 4). Such variables define some statistical characteristics of all the nodes. These variables can also be defined as node-specific, depending on the type of nodes in the network, but we assume in our model that all the nodes are “probabilistically” similar. Values to all the variables in Table 3 are provided as part of the input.

| Variable | What it represents |
|----------|--------------------|
| $N_s$    | probability of a node getting infected from another node (spread), given that they communicate. |
| $N_e$    | denotes the average number of infected nodes in which the infection gets evident. |
| $N_r$    | denotes the average number of infected and “evident” which get repaired after their infection is reported. |
| $N_u$    | denotes the average number of nodes which are upgraded in each time-step, as the ratio to healthy nodes which were not “recently” repaired or upgraded. |
| $\tau_r$ | time(-steps) of immunity from infection after getting repaired. |
| $\tau_u$ | time(-steps) of immunity from infection after getting upgraded. |
| $N_o$    | denotes the fraction of healthy nodes that can be infected from outside, nodes (which were not “recently” repaired or upgraded). |

Table 3: Variables which are globally common for all nodes - applicable to all nodes, but are constant for each node in $G$.

Let $G'$ be the graph instance which is manifested at a time-step $t$. Based on the variables that we have defined, in each discrete time-step $t$ our model proceeds as follows.

1. In the edge set $E(G')$ of an instance $G'$ of $G$, the edge $\{u, v\}$ exists with a probability which is defined by $u.\rho_c$ and $v.\rho_c$ (we discuss this in detail as we describe the algorithm in Section 3). We can consider that a vertex $v$ is in $V(G')$ iff $v$ is a part of an edge in $E(G')$. All other nodes, since they are inactive for $G'$, they are not the part of $V(G')$.

2. Any healthy node is infected from “outside” the network with a probability of $N_o$.

3. From each node $u$ which was infected until time-step $t - 1$, any healthy node $v$ which is adjacent to $u$ in $G'$ gets infected with the probability $N_s$. This happens only when $v$ was repaired at least $\tau_r + 1$ steps before $t$ or $v$ was upgraded at least $\tau_u + 1$ steps before $t$. 


4. The infection gets reported with a probability $N_e$.
5. Each infected node for which an infection is reported is repaired with a probability $N_r$.
6. A healthy node which was repaired at least $\tau_r + 1$ time steps before $t$ and was upgraded at least $\tau_u + 1$ time steps before $t$ is upgraded with probability $N_u$.

Once the infection initiates in $G$, we start monitoring it. After that we terminate when:

1. all the nodes are infected, or
2. none of the nodes is infected.

If all the nodes get infected, we assume that the repair/upgrade strategy was not "good enough", and vice-versa.

3 Preliminaries

$V(G)$ is the set of nodes in a graph $G$. $E(G)$ is the set of edges in $G$. $E(G) = \{\{u, v\} \mid u, v \in V(G), u \neq v\} \implies |E(G)| = |V(G) \times (V(G) - 1)|$ but for each instance $G'$ of $G$, there is a specific probability $p_{uv}$ which decides the existence of an edge $\{u, v\}$ in $G'$. Before the algorithm starts to process $G$, each node is supposed to be initialized with the values $v.i_{\text{e}} = \text{false}$, $v.e_{\text{e}} = \text{false}$, $v.t_{\text{e}} = -1$, $v.t_{\text{r}} = -1$, and $v.t_{\text{u}} = -1$. $v.p_{\text{e}}$, as discussed in Section 2 for each node $v \in V(G)$, is provided with the input to the algorithm.

A state of a node at a particular time-step is defined by the value that each of its variable contains. The state of $G$, the global state, is the set of values of all variables of each node. A trace with respect to a node $v \in V(G)$ is defined by the sequence of states that $v$ goes through in each time-step, starting from time-step 0. The trace of $G$, the global trace, is the sequence of states of $G$. A fault is a contiguous subsequence of the trace of $v$ that is not desirable. In our model, we consider the invariant to be each node present in $G$ being uninfected, that is for each node $v$ we desire $\neg v.i_{\text{e}}$. Otherwise if $v.i_{\text{e}}$, then we consider that $v$ is in faulty state. If any node of $G$ is in faulty state, then we have that $G$ is outside the invariant. The transfer of infection can happen from within the network $G$ (which $v$ is a part of) or from outside of $G$.

From the perspective of the network as a whole, we define a state and trace as follows. While the algorithm proceeds, the global state is defined by the values in the variables $v.i_{\text{e}}$, $v.e_{\text{e}}$, $v.t_{\text{e}}$, $v.t_{\text{r}}$ and $v.t_{\text{u}}$ at a time-step for each node $v$ in $V(G)$; these are the only variables that are possibly modified throughout the execution of the algorithm. A trace is a sequence of such states, that is, a sequence of sets $\{v.i_{\text{e}}, v.e_{\text{e}}, v.t_{\text{e}}, v.t_{\text{r}}\}$ and $v.t_{\text{u}} \mid v \in V(G)$ at each time-step.

The infection spreads between nodes only as a result of a communication. A pair of nodes $u$ and $v$ communicate at a time step $G$ if and only if $\{u, v\} \in E(G')$, where $G'$ is the instance of $G$ at that time-step. Along with the original communication, a node may also transfer an infection to the destination node.
A node may or may not execute a fault if it is already infected. If it executes a fault, we assume that it is visible (throughout and outside the network) and is immediately reported, in which case it will be repaired and does not take part in any communication until repaired. In our model, it is preferred that the repairing and (random) upgrading strategy is able to eventually result in a state of the network $G$ where none of the nodes is infected, despite of the spread of the infection.

4 General firefight burning or burn firefighting

The graph class that we study allows vertices to be added or removed as required. Let $G$ be such a graph.

A general algorithm which simulates the spread and defend of infection in graphs can have the components as described in Table 4 (page 7) and Table 5 (page 7). The working of each function depends on the time-step number stored by the variable $time$. The variables that are discussed in a row are the only variables that are affected by the respective function, no other variable is modified. In this table, in most of the cases, we make copies of the vertex sets from the input graph instance $G'$ to the output graph instance $G''$. In these cases, we have that for a vertex $v$, if $G'.v$ stands for a vertex in the graph $G'$, and $G''.v$ stands for the same vertex (as copied) in the output graph $G''$. Table 4 describes the list of functions that an arbitrary spread-and-defend algorithm might use. It may not be necessary that such an algorithm uses all and only these methods explicitly, but the the underlying functionalities can be divided into these methods following the predicates as described. These functions with respect to their significance will be explained more in the following sections.

Each method takes the time-step number $time$ as an argument. An algorithm can choose to mark some changes by the time-step number. Some changes may depend on the occurrence time-step of certain events. For example, $Outside$-$Infect()$ and $Spread$-$Infection()$ may depend on the values of $\tau_r$ or $\tau_u$ in the vertices. Such dependencies are discussed in the following sections in this article; we are going to utilize the functions from Table 4.

4.1 Logic of algorithms: burning and firefighting

Any algorithm involving the simulation of the spread and defend of infection in graphs can be broken into the following modules, as demonstrated by the steps in Algorithm 1.

Algorithm 1. Given the input initial set of nodes $V$, perform the following steps.

$\text{GENERALIZED-BURNING}(G)$

Initialize $time = 0$. Run the following steps iteratively.

1. $time = time + 1$. 

Table 4: List of functions that an arbitrary spread-and-defend simulation algorithm might use. For each row, column 1: function name, column 2: return value symbol, column 3: predicates followed by the function.

| Function name          | Logical properties                  |
|------------------------|-------------------------------------|
| INSTANCE($G$, $time$) | $G' \subseteq V(G) \land G' = (V, E) \land E \subseteq V \times V.$ |
| OUTSIDE-INFECT($G'$, $time$) | $V' \subseteq V(G') \land \forall v \in V', \neg G'.v.i.s.$ |
| SPREAD-INFECTION($G'$, $time$) | $V' \subseteq V(G') \land \forall v \in V',$ |
| | $(G'.v.i.s \lor (\exists u \in V(G') : \{u, v\} \in E(G') \land G'.u.i.s))$. |
| REPORT-INFECTION($G'$, $time$) | $V'' = V(G') \land E(G'') = E(G') \land \forall v \in V(G'),$ |
| | $(G''.v.e.s \implies (G'.v.i.s \lor G'.v.e.s)) \land (G'.v.e.s \implies G''.v.e.s))$. |
| REPAIR-INSTANCE($G'$, $time$) | $V'' = V(G') \land E(G'') = E(G') \land \forall v \in V(G'),$ |
| | $((-G'.v.i.s \land G''.v.t_r = time) \implies G'.v.i.s)$. |
| UPGRADE-INSTANCE($G'$, $time$) | $V'' = V(G') \land E(G'') = E(G') \land \forall v \in V(G'),$ |
| | $((-G'.v.i.s \land G''.v.t_u = time) \implies \neg G'.v.i.s)$. |
| DELETE-INFECTED($G'$, $time$) | $V'' \subseteq V(G') \land \forall v \in V(G') \setminus V(G''), G'.v.i.s.$ |
| INSERT-NEW($G'$, $time$) | $V'' \subseteq V(G'') \land \forall v \in V(G'') \setminus V(G'), \neg G''.v.i.s.$ |

Table 5: List of functions that may be invoked by the functions in Table 4.

| Function name | Functionality |
|---------------|---------------|
| INFECTION($v$) | infect $v$ |
| REPAIR($v$)   | repair $v$ |
| UPGRADE($v$)  | upgrade $v$ |
2. \( G' = \text{Instance}(G, time) \).
3. \( I_{\text{out}} = \text{Outside-Infect}(G', time) \).
4. \( S_{\text{in}} = \text{Spread-Infection}(G', time) \).
5. \( \forall v : v \in S_{\text{in}} \cup I_{\text{out}}, \text{Infect}(v) \).
6. \( G' = \text{Report-Infection}(G', time) \).
7. \( G' = \text{Repair-Instance}(G', time) \).
8. \( G' = \text{Upgrade-Instance}(G', time) \).
9. \( G' = \text{Delete-Infected}(G', time) \).
10. \( G' = \text{Insert-New}(G', time) \).

5 Only burning

In this section, we are going to study the spread of contagion through a network. The functions that we are going to utilize are as follows. \( \epsilon \) stands for a null character.

A. \text{Instance}(G)

1. \( V' \leftarrow V(G) \). \( E' \leftarrow \phi \).
2. \text{for} every set \( \{u, v\} : u, v \in V' \land u \neq v \),
3. \( e_{uv} \leftarrow \epsilon \). \( e_{vu} \leftarrow \epsilon \).
4. With probability \( u.\rho_c \), execute: \( e_{uv} \leftarrow (u, v) \).
5. With probability \( v.\rho_c \), execute: \( e_{vu} \leftarrow (v, u) \).
6. \text{if} \( e_{uv} = (u, v) \land e_{vu} = (v, u) \), then
7. \( E' \leftarrow E' \cup \{u, v\} \).
8. Return \( G' = (V', E') \).

B. \text{Outside-Infect}(G, time)

1. \( I_{\text{out}} = \phi \).
2. \( \forall v \in V(G) \):
3. \( \text{if} \ \neg \text{Is-Infected}(v) \),
4. \text{if} \ \{v, t_r = -1 \land time - v.t_r \geq \tau_r \} \land \{v, t_u = -1 \land time - v.t_u \geq \tau_u \}
5. \text{With probability} \ N_o \text{, execute:}
6. \( I_{\text{out}} \leftarrow I_{\text{out}} \cup \{v\} \).
7. Return \( I_{\text{out}} \).

C. \text{Spread-Infection}(G, time)

1. \text{for} each set \( u, v : u, v \in V(G) \):
2. \( \text{if} \ \{u, v\} \in E(G) \)
3. \( \text{if} \ XOR(u.i_s, v.i_s) \):
Algorithm 2. Given the input graph \( G = (V,E) \), where essentially the edge set \( E(G) \) is empty, along with the variables discussed in Table 1, Table 2 and Table 3 provided as part of the input, perform the following steps.

**Variable-Burning** \((G)\)

1. Initialize \( time = 0 \).
2. Repeat the following steps until the algorithm stops.
   1. \( G' = \text{Instance}(G) \).
   2. \( \text{if} \ \forall \ v \in V(G'), \text{Is-Infected}(v), \text{then Stop.} \)
   3. \( time \leftarrow time + 1 \)
   4. \( I_{\text{out}} = \text{Outside-Infect}(G', \text{time}). \)
   5. \( S_{\text{in}} = \text{Spread-Infection}(G', \text{time}). \)
   6. \( \forall v : v \in S_{\text{in}} \cup I_{\text{out}}, \text{Infect}(v). \)
   7. \( V(G) \leftarrow V(G'). \)

We describe Algorithm 2 in the following few paragraphs. We initiate with an instance \( G' \) of \( G \) (line 1). \( G' \) has the same node set as that of \( G \). The edges in \( G' \) are decided based on the values in \( v.\rho_c \) in every node \( v \), such that a node \( v \) may decide an arc \((v,u)\) to exist based on \( v.\rho_c \), but the edge \( e = \{u,v\} \) will be inserted in \( G'.E \) only if \( u \) also decides the arc \((u,v)\) to exist based on \( u.\rho_c \).

We stop if every node is infected (line 2).

Now we simulate the infection that nodes get from outside of the network \( G \). We first compute the set of nodes \( I_{\text{out}} \) which can be infected from outside (line 4); each node gets infection from outside the network with probability \( N_o \). Then we determine the set of nodes \( S_{\text{in}} \) which are infected as a result of the spread of infection from within the network (line 5). Both \( I_{\text{out}} \) and \( S_{\text{in}} \) are computed independent on each other. At any time step, both of them depend on the status of the nodes in the beginning of that time-step. Then we actually infect the nodes in \( I_{\text{out}} \) and \( S_{\text{in}} \) (line 6). This is similar to the notion of the graph burning procedure [6]. We are only spreading infection to the healthy nodes in line 5, so this would help us simulate the notion that a node can get
infection from both inside and outside of its local network only if it is healthy. In line 5, we spread the infection from within the network such an infected node \( u \) can infect an uninfected node \( v \) with the probability \( N_s \) if \( \{u,v\} \in E(G) \). When a node \( v \) gets infected, \( v.i \) is set to \( \text{true} \).

In the above set of functions, (1) the \textbf{if} condition at Line 4 of \textsc{Outside-Infect()}, and (2) the \textbf{if} conditions at line 4, 6 and 7 of \textsc{Spread-Infection()} are not useful for our purposes right now, but they will become useful later (in Section 6). For now, they can be safely ignored, as they are will always remain true according to Algorithm 2.

Algorithm 2 follows Algorithm 1 as per the constraints listed in Table 4. The functions which are used explicitly follow respective functions. Algorithm 2 also follows all the constraints of Algorithm 1 where the functions of Algorithm 1 are not used in Algorithm 2.

**Observation 1.** If at the beginning of some time-step, the fraction of healthy nodes is \( h \), then at the end of that time-step the fraction of healthy nodes will be \( h(1 - N_o) \).

**Lemma 1.** If at the beginning of some time-step, the fraction of healthy nodes is \( h \), and the edge probability for each vertex is \( \rho_c \), then the number of nodes that remain healthy at the end of that time-step is \( h - N_s nh(1 - h)(\rho_c)^2 \).

**Proof.** In the beginning, the fraction of infected nodes is \( 1 - h \). Let the total number of nodes in the subject graph be \( n \). If each healthy node is connected to all the unhealthy nodes, then the number of communications that any healthy node will do is \( n(1 - h) \).

The edge probability of each vertex is \( \rho_c \), so the number of communications that a healthy node will do with the unhealthy nodes is \( (\rho_c)^2 n(1 - h) \).

Now if each healthy node is connected to one unhealthy node, then any healthy node will get infected with the probability \( N_s \).

Since the edge probability of each node is \( \rho_c \), so the probability for a pair of nodes to agree to communicate is \( (\rho_c)^2 \). Each healthy node is connected to \( (\rho_c)^2 n(1 - h) \) unhealthy nodes, so the probability with which any healthy node will be infected is \( N_s (\rho_c)^2 n(1 - h) \).

There are \( nh \) healthy nodes. The fraction of nodes which get infected in this step is \( \frac{N_s n(1 - h)(\rho_c)^2 \times nh}{n} = N_s nh(1 - h)(\rho_c)^2 \). The fraction of nodes remaining healthy after one time step is \( h - N_s nh(1 - h)(\rho_c)^2 \).

**Theorem 1.** If at the beginning of some time-step, the fraction of healthy nodes is \( h \), then by the end of that time-step, the fraction of nodes that are healthy is \( h - hN_o + hN_s n(1 - h)(\rho_c)^2 - h^2 N_o N_s n(1 - h)(\rho_c)^2 \).

**Proof.** According to the description of the algorithm, first (1) the nodes are “chosen” to be infected from outside, then (2) the nodes are chosen which get infected from within the network, and then (3) the nodes chosen at (1) and (2) are “declared” infected.
So the infection from outside and spread of infection within the network happen independently from each other. This infection only depends on the vertices that were already infected at the end of the previous time-step. The number of infected nodes is the union of the fraction of nodes infected from outside and the nodes which are infected due to the spread of infection from within the network. This number is $hN_o + hN_s n(1 - h)(\rho_c)^2 - hN_o \times hN_s n(1 - h)(\rho_c)^2$. The number of healthy vertices that remain after one time step is $h - hN_o + hN_s n(1 - h)(\rho_c)^2 - hN_o \times hN_s n(1 - h)(\rho_c)^2 = h - hN_o + hN_s n(1 - h)(\rho_c)^2 - h^2 N_o N_s n(1 - h)(\rho_c)^2$.

The experimental results are as follows. We took average over 10 runs. We took the following values.

| n    | 100 |
|------|-----|
| $N_s$ | .2  |
| $N_o$ | .05 |
| $\rho_c$ | 0.05 |

Table 6: Initial values of the variables in the experiment.

Figure 1: The experimental results agree with theoretical results (of Theorem 1). This graph is a plot of the time-step $i$ number against the mean number of healthy vertices at $i^{th}$ time-step (over all the runs of the algorithm). The longest run took 84 time-steps.

From Theorem 1 it can be observed that the number that we have come up with depends on $n$, the number of vertices in the graph as well. It is proportional to the number of vertices in the graph, and it depends on the edge probability.
as well. Recall that in this section, we have assumed that the probability of communication for all the vertices is same.

6 Introducing repair and upgrade on nodes

In this section, we will introduce the notion of repair and upgrade of the nodes. We also have that after repair or upgrade of a node, there is a certain amount of time-steps until which that node remains immune to infection, that is, until a certain amount of time, it will not catch infection even after an infected communication.

The additional functions that we utilize are as follows.

A. Report-Infection\((G, time)\)
1. \(\forall v \in V(G)\)
2. if \(v.i_s \land \neg v.e_s\)
3. With probability \(\mathcal{N}_c\) execute
4. \(v.e_s \leftarrow true, v.t_e \leftarrow time\).

B. Repair-Instance\((G, time)\)
1. \(\forall v \in V(G)\)
2. if \(v.i_s \land v.e_s\), then
3. With probability \(\mathcal{N}_r\), execute: Repair-node\((v, time)\).

C. Upgrade-Instance\((G, time)\)
1. \(\forall v \in V(G)\)
2. if \(\neg v.i_s\), then
3. With probability \(\mathcal{N}_u\), execute: Upgrade-node\((v, time)\).

D. Repair-node\((v, time)\)
1. \(v.i_s \leftarrow false\)
2. \(v.t_r \leftarrow time\)
3. \(v.t_e \leftarrow -1\)
4. \(v.e_s \leftarrow false\)

E. Upgrade-node\((v, time)\)
1. \(v.t_u \leftarrow time\)
2. \(v.t_e \leftarrow -1\)
3. \(v.e_s \leftarrow false\)
4. \(v.t_r \leftarrow -1\)

We study the behaviour of a random temporal graph when we introduce repair of infected vertices and upgrade of healthy vertices. The upgradation of healthy nodes that we introduce here is similar to firefighting, with a difference that we impose a minimum time \(\tau_u\) (only) until which the upgraded node remains immune to the infection.

The algorithm that we use here is as follows.
Algorithm 3. Given the input graph $G = (V, E)$, where essentially the edge set $E(G)$ is empty, along with the variables discussed in Table 1, Table 2 and Table 3 provided as part of the input, perform the following steps.

**Variable-Burning($G$)**

Initialize $time = 0$ and $infection\_started = false$. Repeat the following steps until the algorithm stops.

1. $G' = \text{Instance}(G)$.
2. if not $infection\_started$
3. if $\exists v \in G' : \text{Is-Infected}(v)$, then $infection\_started = true$
4. if $infection\_started$:
5. if $\forall v \in V(G')$, $\text{Is-Infected}(v)$, then Stop.
6. if $\forall v \in V(G')$, $\neg \text{Is-Infected}(v)$, then Stop.
7. $time \leftarrow time + 1$
8. $I_{out} = \text{Outside-Infect}(G'$, $time)$.
9. $S_{in} = \text{Spread-Infection}(G'$, $time)$.
10. $\forall v : v \in S_{in} \cup I_{out}$, Infect($v$).
11. Report-Infection($G'$, $time$).
12. Repair-Instance($G'$, $time$).
13. Upgrade-Instance($G'$, $time$).
14. $V(G) \leftarrow V(G')$.

Algorithm 3 is more complex than Algorithm 2. We are going to discuss the differences between Algorithm 3 and Algorithm 2 and the new insertions in Algorithm 3. In this section, we will have to use the lines that we insisted to ignore after we described some functions in Section 5. After initializing a functional instance of $G$ (line 1), we determine if the infection has started (lines 2, 3).

After when the infection has started in $G$, then we stop if every node is infected (lines 4, 5). Similarly, after when the infection has started in $G$, then we stop if every node is not infected (lines 4, 6), which will imply that the all the nodes in the system have been “cured”. The current time-step number is stored in the variable $time$ (line 7).

We compute the set $I_{out}$ of vertices which are infected as a result of outside infection (line 8) and the set $S_{in}$ of vertices which are infected as a result of the spread of infection from within the network (line 9). Then burn the vertices in $I_{out}$ and $S_{in}$ (line 10). A healthy node is infected from outside with a probability of $N_o$, and is infected as a result of the spread of infection from with the network with a probability of $N_f$. In addition to these constraints, a node can only be infected if it was repaired at least $\tau_r$ steps before $time$, also, it should be upgraded at least $\tau_u$ time-steps before $time$. That is, whether a node $v$ is infected from outside or from within the network, it is can be infected only if
\( (v.t_r = -1 \lor time - v.t_r \geq \tau_r) \land (v.t_r = -1 \lor time - v.t_r \geq \tau_r) \) holds true. In addition to that if the infection of a node has become evident, that is, if \( v.e_s \) is set to true, then it cannot be infected, as we assume that that node is under scrutiny and will not take part in communications or will take part in screened communications only. When a node \( v \) is declared infected, \( v.i_s \) is set to true.

The nodes which are infected may be reported as their infection gets evident with a probability of \( N_e \) (line 11), and the time of their being reported is recorded, that is, for each node \( v \in G \), \( v.e_s \) is set to true and \( v.t_e \) is set to \( time \) based on \( N_e \). This simulates that the infection in a node may not get reported immediately as they get infected. It is not necessary for a node to execute the fault state as soon as it gets infected. As the node executes a fault state, we assume that its infection is evident throughout and outside the network, its infection gets reported. A node whose infection gets reported shall not take part in any communication or will take part in screened communications only in \( G \) until it is repaired, so it does not spread infection to any other nodes.

Any node which has been infected will be repaired (line 12) with probability \( N_r \); for each node \( v \in G \) for which \( v.i_s \) is true, \( v.i_s \) is set to false and \( v.t_r \) is set to \( time \) based on \( N_r \). This simulates the notion that a node may take time to get repaired and cannot be repaired immediately in the same time-step in which its infection was reported. The notion of probability is inserted here to create a delay for a node to be finally repaired. Next, any uninfected node will be upgraded (line 13) with probability \( N_u \); for each node \( v \in G \) for which \( v.i_s \) is false, \( v.t_u \) is set to \( time \) based on \( N_u \). This simulates the notion that each node may not be set to upgraded at each time-step, a node may be chosen randomly by the system administrator to be upgraded. The notion of probability is inserted here to denote randomness of choice to upgrade a node.

After a node is repaired (respectively, upgraded), it is immune to the infection for next \( \tau_r \) (respectively, \( \tau_u \)) time-steps and thus cannot be infected. Also, note that we are upgrading a node irrespective of when it was upgraded latest; this may be less coherent with real-time human social networks in the sense that if a healthy human has been treated against some disease, then she may get a dose of the vaccine in less than \( \tau_r \); she may not get two doses of vaccines within more or less than a certain period of time. But this is conforming with a network of computers.

We now study the behaviour of the instructions that we have inserted at line 10 - line 12, that is we study the behaviour of the lines 10 - line 12 when we have that the spread and outside infection of the nodes have already taken place.

Let that at the beginning of some time-step, let

1. \( h \) be the fraction of healthy nodes with not repaired or upgraded status,
2. \( u \) be the fraction of healthy nodes with upgraded status,
3. \( r \) be the fraction of healthy nodes with repaired status,
4. \( e_y \) be the fraction of nodes that have already shown evidence of infection,
5. $\epsilon_n$ be the fraction of nodes that have not shown evidence of infection.

We have that $h + u + r + \epsilon_n + e_y = 1$. Now (by Theorem 1) by the end of line 9 (Algorithm 3), the fraction of infected nodes will increase by $h\mathcal{N}_o + \mathcal{N}_s ne_n(\rho_c)^2 - h\mathcal{N}_o\mathcal{N}_s ne_n(\rho_c)^2$ of the nodes.

The following values will be affected.

1. The final value of $h$ will be
   \[ h = h - (h\mathcal{N}_o + \mathcal{N}_s ne_n(\rho_c)^2 - h\mathcal{N}_o\mathcal{N}_s ne_n(\rho_c)^2). \]

2. The final value of $\epsilon_n$ will be
   \[ \epsilon_n = \epsilon_n + h\mathcal{N}_o + \mathcal{N}_s ne_n(\rho_c)^2 - h\mathcal{N}_o\mathcal{N}_s ne_n(\rho_c)^2. \]

Lemma 2. $\epsilon_n(e_y + \mathcal{N}_e)$ nodes in total show infection evidence in this step.

Proof. The fraction of unhealthy nodes which have not shown evidence of infection yet is $\epsilon_n$. $\mathcal{N}_e$ (cf. Table 3) nodes of them show evidence of infection. So $\epsilon_y + \epsilon_n\mathcal{N}_e$ vertices in total are the fraction of vertices which are infected and show evidence of infection by this time-step. \[ \square \]

At the end of line 10,

1. The final value of $e_y$ will be
   \[ e_y = e_y + \epsilon_n\mathcal{N}_e. \]

2. The final value of $\epsilon_n$ will be
   \[ \epsilon_n = \epsilon_n - \epsilon_n\mathcal{N}_e. \]

Lemma 3. The vertices repaired are $e_y\mathcal{N}_r$ and the vertices upgraded are $h\mathcal{N}_u$.

Proof. The fraction of nodes which have shown infection evidence is $e_y$. So it trivially follows that the fraction of nodes that are newly repaired is $e_y\mathcal{N}_r$.

Only those nodes are considered upgraded which do not have infection, and were upgraded more than $\tau_u$ time-steps ago or were repaired $\tau_r$ time-steps ago. Then the total number of nodes which are upgraded are $h\mathcal{N}_u$. \[ \square \]

At the end of line 11,

1. The final value of $r$ will be
   \[ r = r + e_y\mathcal{N}_r. \]

2. The final value of $e_y$ will be
   \[ e_y = e_y - e_y\mathcal{N}_r. \]
At the end of line 12,

1. The final value of $h$, and $u$ will be

$$h, u = h - h \ast N_u, u + hN_u.$$ 

**Observation 2.** Let that during an iteration of the algorithm, $r_{\text{time}-\tau_r}$ be the fraction of infected vertices with evident status that were repaired at the time-step $(\text{time} - \tau_r)$, then if $\text{time} \geq \tau_r + 1$, then the final fraction of nodes at the end of that time step with repair status now is $r - r_{\text{time}-\tau_r}$.

At the end of a time step,

1. The final value of $h$ will be

$$h = h + r_{\text{time}-\tau_r}.$$ 

2. The final value of $r$ will be

$$r = r - r_{\text{time}-\tau_r}.$$ 

**Observation 3.** Let that during an iteration of the algorithm, $u_{\text{time}-\tau_u}$ be the fraction of infected vertices with evident status that were upgraded at the time-step $(\text{time} - \tau_u)$, then if $\text{time} \geq \tau_u + 1$, then the final fraction of nodes at the end of that time step with repair status now is $u - u_{\text{time}-\tau_u}$.

At the end of a time step,

1. The final value of $h$ will be

$$h = h + u_{\text{time}-\tau_u}.$$ 

2. The final value of $u$ will be

$$u = u - u_{\text{time}-\tau_u}.$$ 

For Observation 2 and Observation 3, while computing for the statistical values, instead of deriving a complex formula to predict the values of $u$ or $r$ $\tau_u$ or $\tau_r$ time-steps ago, we use the standard dynamic programming trick to retrieve those values.

The experimental results are as follows (Figure 2). We took average over 10 runs. We initiated the experiment with the following values (Table 7). In all the runs, all the nodes were cured.
Adding and removing nodes

In this section, we are going to add more complexity to Algorithm 3 where we allow addition and removal of nodes.

The additional functions that we utilize are as follows.

A. **Delete-Infected(G)**

1. **for** each node $v$ in $V(G)$,
2. if $v.t_s$, then
3. With probability $\rho_{del}$, execute $\text{Delete-node}(G', v)$.

B. **Insert-New(G)**

1. **for** each node $v \in V(G)$,
2. With probability $\rho_{ins}$, execute:
3. $v \leftarrow$ a new node.
4. $v.i_s \leftarrow false$. $v.e_s \leftarrow false$. $v.t_e \leftarrow -1$. $v.t_r \leftarrow -1$. $v.t_u \leftarrow -1$.
5. $v.\rho_c \leftarrow$ a random or fixed number between 0 and 1 (say, some number between $\min_{u \in V(G)} \{u.\rho_c\}$ and $\max_{u \in V(G)} \{u.\rho_c\}$).
6. $V(G) = V(G) \cup \{v\}$.

C. **Delete-node(G, v)**

1. $V(G) = V(G) \setminus \{v\}$

We have $G$ as initial graph with the properties as described in Table 1 and Table 2. We have a list of 15 functions described as follows, which we utilize in the main algorithm.

**Algorithm 4.** *Given the input graph $G = (V,E)$, where essentially the edge set $E(G)$ is empty, along with the variables discussed in Table 1, Table 2 and Table 3 provided as part of the input, perform the following steps.*

| $n$  | 100 |
|------|-----|
| $\mathcal{N}_s$ | .2 |
| $\mathcal{N}_c$ | .3 |
| $\tau_q$ | 25 |
| $\tau_u$ | 25 |
| $\mathcal{N}_p$ | .05 |
| $\mathcal{N}_r$ | .5 |
| $\mathcal{N}_o$ | .15 |
| $\rho_c$ | 0.05 |

Table 7: Initial values with which the experiment started: burn with repair and upgrade.
Variable-Burning($G$)

Initialize $time = 0$ and $infection\_started = false$. Repeat the following steps until the algorithm stops.

1. $G' = \text{INSTANCE}(G)$.
2. if not $infection\_started$
3. if $\exists v \in G' : \text{IS-INFECTED}(v)$, then $infection\_started = true$
4. if $infection\_started$:
5. if $\forall v \in V(G')$, is-INFECTED(v), then Stop.
6. $time \leftarrow time + 1$
7. $I_{out} = \text{OUTSIDE-INFECT}(G', time)$.
8. $S_{in} = \text{SPREAD-INFECTION}(G', time)$.
9. $\forall v : v \in S_{in} \cup I_{out}$,\text{ INFECT}(v)$.
10. $\text{REPORT-INFECTION}(G', time)$.
11. $\text{REPAIR-INSTANCE}(G', time)$.
12. $\text{UPGRADE-INSTANCE}(G', time)$.
13. $\text{DELETE-INFECTED}(G')$.
14. $\text{INSERT-NEW}(G')$.
15. $V(G) \leftarrow V(G')$. 

Figure 2: Experimental versus theoretical results: burning with repair and upgrade.
We explain Algorithm 4 as follows. Most of the functionality is similar to Algorithm 3 except for lines 13 and 14.

An infected node is deleted from $G$ (lines 14, 16) with a probability $\rho_{del}$ such that for each node $v$ in $V(G)$, $v$ is deleted with probability $\rho_{del}$ only if $\text{v.i.s}$ is true. This is based on the notion that a node can get unusable, and once a fault makes a node unusable, it can be discarded completely. About $\rho_{ins} \times |V(G)|$ new nodes can be inserted to $G$ (lines 15, 16) such that for each $v$ in $V(G)$, a new node can be inserted with probability $\rho_{ins}$. This denotes the potential efforts made by the system administrators to maintain the usability and efficiency of the network which is also based on the number of the nodes in it. Mark that $V(G)$ itself is variable.

For the following two lemmas, we are going to assume that at the beginning of line 13,

1. $h$ is the fraction of healthy nodes with not repaired or upgraded status,
2. $u$ be the fraction of healthy nodes with upgraded status,
3. $r$ be the fraction of healthy nodes with repaired status,
4. $e_y$ be the fraction of nodes that have already shown evidence of infection, and
5. $e_n$ be the fraction of nodes that have already shown evidence of infection.

**Lemma 4.** Let that at the end of line 13, the final fraction of healthy vertices is

\[
h + u + r \frac{1}{1 - (e_y + e_n)\rho_{del}}.
\]

**Proof.** The number of vertices removed are $n(e_y + e_n)\rho_{del}$. The number of vertices remaining now is $n - n(e_y + e_n)\rho_{del}$.

The final fraction of healthy vertices is

\[
\frac{n(h + u + r)}{n - n(e_y + e_n)\rho_{del}} = \frac{h + u + r}{1 - (e_y + e_n)\rho_{del}}.
\]

**Lemma 5.** Let that at the end of line 14, the final fraction of healthy vertices is

\[
h + u + r \frac{1 + \rho_{ins}}{1 - (1 - e_y - e_n)\rho_{del}}.
\]

**Proof.** The number of nodes remaining at the end of line 13 is $n - n(1 - e_y - e_n)\rho_{del}$. The number of vertices now is $(n - n(1 - e_y - e_n)\rho_{del}) + \rho_{ins}(n - n(1 - e_y - e_n)\rho_{del})$.

The final fraction of healthy vertices is

\[
\frac{h + u + r}{(1 - (1 - e_y - e_n)\rho_{del}) + \rho_{ins}(1 - (1 - e_y - e_n)\rho_{del})}.
\]

The experimental results are as follows. We took average over 10 runs. We took the following values.

In all the runs, all the nodes were cured.
Table 8: Initial values with which the experiment started: burn with repair, upgrade, add and remove.

|   |   |
|---|---|
| $n$ | 100 |
| $N_s$ | .2 |
| $N_c$ | .3 |
| $\tau_r$ | 25 |
| $\tau_u$ | 25 |
| $N_o$ | .05 |
| $N_r$ | .5 |
| $N_u$ | .15 |
| $\rho_c$ | 0.05 |
| $\rho_{de}$ | 0.002 |
| $\rho_{ins}$ | 0.005 |

8 Variable edge probability

Let that at some time-step time,

1. $f_1, f_2, ..., f_k$ be the fraction of nodes
2. $\rho_1, \rho_2, ..., \rho_k$ respectively be the edge probabilities of the nodes,
3. $h_1, h_2, ..., h_k$ be the fraction of healthy nodes not having the upgrade or repair status,
4. $r_1, r_2, ..., r_k$ be the fraction of healthy nodes having the repair status,
5. $u_1, u_2, ..., u_k$ be the fraction of healthy nodes having the upgrade status,
6. $e_{n_1}, e_{n_1}, ... , e_{n_1}$ be the fraction of infected nodes which have not shown infection evidence,
7. $e_{y_1}, e_{y_1}, ... , e_{y_1}$ be the fraction of infected nodes which have shown infection evidence,

The probability of the edges that they make with the unhealthy and unreported nodes will be

$$E_j = \sum_{i=0}^{k} (h_j e_{n_i}) (\rho_j \rho_i).$$

The fraction of nodes infected by spread and outside infection will be

$$I_j = nE_jN_s + h_jN_o - (nE_jN_s) \times (h_jN_o).$$

At the end of line 9, the following values are modified.

1. The final fraction of infected nodes will be (we show this by reassigning the value to $e_{n_j}$ so that we can reuse it later)

$$e_{n_j} = e_{n_j} + I_j.$$
Figure 3: Experimental versus theoretical results: burning with repair, upgrade, add and remove.

2. The final fraction of healthy nodes not having the upgrade or repair status is

\[ h_j = h_j = I_j. \]

At the end of line 10, \( N_e e_{n_j} \) of the total nodes, which were not evident earlier, become evident. At the end of line 10,

1. The final fraction of infected nodes not having evident status will be

\[ e_{n_j} = e_{n_j} - N_e e_{n_j}. \]

2. The final fraction of infected nodes evident status will be

\[ e_{n_j} = e_{y_j} + N_e e_{n_j}. \]

At the end of line 11, \( \sum_j N_r e_{y_j} \) of the nodes are repaired. At the end of line 11,

1. The final fraction of repaired nodes will be

\[ r_j = r_j + N_r e_{y_j}. \]

2. The final fraction of infected nodes with evident status will be

\[ e_{y_j} = e_{y_j} - N_r e_{y_j}. \]
At the end of line 12, \( \sum_j (h_j + r_j + u_j)N_u \) more of the vertices are upgraded. At the end of line 12,

1. The final fraction of repaired nodes will be
   \[ r_j = r_j - N_u r_j. \]

2. The final fraction of healthy nodes not having repair or upgrade status will be
   \[ h_j = h_j - N_u h_j. \]

3. The final fraction of upgraded nodes will be
   \[ u_j = (h_j + r_j + u_j)N_u. \]

At the end of line 13, \( \sum_j \rho_{del}(e_{n_j} + e_{y_j}) \) of the nodes are removed. At the end of line 13,

1. The total number of nodes is
   \[ n = n - \rho_{del}(e_{n_j} + e_{y_j}) \]

2. The final fraction of fraction of nodes will be
   \[ f_j = \frac{n f_j}{n - \rho_{del}(e_{n_j} + e_{y_j})} = \frac{f_j}{1 - \rho_{del}(e_{n_j} + e_{y_j})} \]

3. The final fraction of healthy nodes not having the upgrade or repair status will be
   \[ h_j = \frac{n h_j}{n - \rho_{del}(e_{n_j} + e_{y_j})} = \frac{h_j}{1 - \rho_{del}(e_{n_j} + e_{y_j})} \]

4. The final fraction of healthy nodes having the repair status will be
   \[ r_j = \frac{n r_j}{n - \rho_{del}(e_{n_j} + e_{y_j})} = \frac{r_j}{1 - \rho_{del}(e_{n_j} + e_{y_j})} \]

5. The final fraction of healthy nodes having the upgrade status will be
   \[ u_j = \frac{n u_j}{n - \rho_{del}(e_{n_j} + e_{y_j})} = \frac{u_j}{1 - \rho_{del}(e_{n_j} + e_{y_j})} \]

6. The final fraction of infected nodes which have not shown infection evidence will be
   \[ e_{n_j} = \frac{n e_{n_j} - n \rho_{del} e_{n_j}}{n - \rho_{del}(e_{n_j} + e_{y_j})} = \frac{e_{n_j} - \rho_{del} e_{n_j}}{1 - \rho_{del}(e_{n_j} + e_{y_j})}, \text{ and} \]
7. The final fraction of infected nodes which have shown infection evidence will be
\[ e_{y_j} = \frac{n e_{y_j} - n \rho_{del} e_{y_j}}{n - \rho_{del} n(e_{n_j} + e_{y_j})} = \frac{e_{y_j} - \rho_{del} e_{y_j}}{1 - \rho_{del} (e_{n_j} + e_{y_j})}. \]

At the end of line 14, \( n \rho_{ins} \) nodes are added. At the end of line 14,

1. The total number of nodes is
\[ n = n + n \rho_{ins}. \]

2. The final fraction of fraction of nodes will be
\[ f_j = \frac{n f_j}{n + n \rho_{ins}} = \frac{f_j}{1 + \rho_{ins}}. \]

3. The final fraction of healthy nodes not having the upgrade or repair status will be
\[ h_j = \frac{n h_j}{n + n \rho_{ins}} = \frac{h_j}{1 + \rho_{ins}}. \]

4. The final fraction of healthy nodes having the repair status will be
\[ r_j = \frac{n r_j}{n + n \rho_{ins}} = \frac{r_j}{1 + \rho_{ins}}. \]

5. The final fraction of healthy nodes having the upgrade status will be
\[ u_j = \frac{n u_j}{n + n \rho_{ins}} = \frac{u_j}{1 + \rho_{ins}}. \]

6. The final fraction of infected nodes which have not shown infection evidence will be
\[ e_{n_j} = \frac{n e_{n_j}}{n + n \rho_{ins}} = \frac{e_{n_j}}{1 + \rho_{ins}}, \]
and

7. The final fraction of infected nodes which have shown infection evidence will be
\[ e_{y_j} = \frac{n e_{y_j}}{n + n \rho_{ins}} = \frac{e_{y_j}}{1 + \rho_{ins}}. \]

Now we discuss what happens when the nodes leave their upgrade or repair status. If \( time \geq \tau_r + 1 \), then

1. The final fraction of healthy nodes not having repair or upgrade status will be
\[ h_j = h_j + r_{time - \tau_r}. \]

2. The final fraction of nodes with repair status will be
\[ r_j = r_j - r_{time - \tau_r}. \]
If $\text{time} \geq \tau_u + 1$, then

1. The final fraction of healthy nodes not having repair or upgrade status will be
   $$h_j = h_j + u_{\text{time} - \tau_j}.$$  

2. The final fraction of nodes with repair status will be
   $$u_j = u_j - u_{\text{time} - \tau_j}.$$ 

### 8.1 Test case

We demonstrate the working of Algorithm 4 on a small network of 10 initial nodes. After that, in Section 9, we discuss the possible variations by which this model can be used to study the several networks, with more focus on the human and biological networks. We initialized the global variables to the following values, as described in Table 9.

| $N_s$ | .2 |
|-------|----|
| $N_e$ | .1 |
| $\tau_r$ | 60 |
| $\tau_u$ | 60 |
| $N_c$ | .05 |
| $N_i$ | .08 |
| $N_u$ | .15 |
| $\rho_{del}$ | 0.02 |
| $\rho_{ins}$ | 0.005 |

**Table 9:** Input values of the variables of sample.

In the graph $G$ of order 10 such that for 4 nodes, $v, \rho_c$ was 0.1 and for 6 nodes, $v, \rho_c$ was 0.2, we ran the algorithm 10,000 times. Each iteration was run until all the nodes were reported not infected after the onset of the initial infection in the network. We received the following output, described in Table 10. Here, we only focus on the average time to disinfect (for one iteration, time to disinfect).

**Definition 4. Time to Disinfect.** Given an input graph and the infection and disinfection processes running on it, the time to disinfect is the difference between the time-step number in which infection started was set to true (line 3, Algorithm 4), and the time-step number in which for each node $v$, IsInfected($v$) is false, that is, when all nodes in the network are disinfected after the onset of infection (line 6, Algorithm 4).

### 9 Discussion

Several or all the proceedings of this algorithm can be transformed from probabilistic to definite algorithmic procedures (for example, changing how we spread
infection, or how we upgrade a node) and used to study the systems under those modified constraints.

In Table 1, we discussed some variables which decide the removal of “worn-out” nodes or insertion of new nodes to a network. This may represent the removal of an infected node from the network, or insertion of a new node to a network. This may apply to more general and real-time systems as nodes can be removed from a network or new nodes can be inserted to the network for several administrative or cost-related reasons. If no modifications to the number of nodes is desired, then both $\rho_{\text{ins}}$ and $\rho_{\text{del}}$ can be set to zero. Clearly, temporal graphs are a subclass of this graph class where both $\rho_{\text{ins}}$ and $\rho_{\text{del}}$ are zero.

As discussed in Table 3, the notion associated with $\tau_r$ (respectively, $\tau_u$) can be changed to a probability of a node being immune to infection after a repair (respectively, upgrade). We have that when a node is repaired (respectively, upgraded), it is vulnerable to infection which is spread from within the network or introduced from outside the network after $\tau_r$ (respectively, $\tau_u$) time steps. The notion of the probability of infection in a node getting reported is based on the fact that a node reaches a state that is a fault is not necessary immediately when a fault is inserted. The notion of probability of repair denotes the average of time-steps that nodes takes to get repaired. The notion of a non-infected node is upgraded with a probability denotes the fraction of nodes that are upgraded on an average in a single time-step.

Several modifications of this model can be studied. For modelling human social networks, we have that a group of, say, $k$ people meet frequently, that is the edge probability per node can be high for that cluster of $k$ nodes, on the other hand, if two nodes belong to different clusters, this probability reduces. If the cluster are populated far apart, then this probability can further reduce with the edge probability being in some inverse proportion to some exponent of the distance. This is similar for social networks in other biological ecosystems. In the current proposed model, the edge probability of a node $v$ has uniform impact on all possible edges containing $v$ in the network $G$. Further, the probability of repair may be increased in a supercluster of nodes (a cluster of clusters residing locally with respect to each other) where the percentage of infection is greater. We have, on the other hand, we have used the value over all the nodes. Another very obvious and desirable modification is studying several algorithmic strategies of a combination of burning and firefighting on several graph classes, instead to choosing nodes based on a probability. Such variations can be used to study the nature of the spread of infection along with an optimal vaccination strategy.

| Average total time-steps | 34.9148 |
|--------------------------|--------|
| Average time-steps to disinfect | 28.403 |
| New nodes added (average) | 1.9348 |
| Infected nodes removed (average) | 1.0633 |
| Infection start time (average) | 6.5118 |

Table 10: Output of sample.
from the perspective of a human social network, or other complex biological or artificial networks.

More generally, the class of graphs that we study is an implementation of the graph class defined as follows.

Definition 5. Altered Temporal Graphs with Generalization. To formally define the class of graphs that we study in this article, we first define a set of nodes $V$ and a set of edges $E$ such that $\forall u,v \in V, \{u,v\} \in E$. A graph $G$ falling in this graph class is defined as follows: $G = (V_1, V_2, ..., V_\ell, E_1, E_2, ..., E_\ell)$ such that $\forall i: 1 \leq i \leq \ell, V_i \subseteq V \wedge E_i \subseteq E \wedge G_i = (V_i, E_i)$ is an instance of $G$.

The class of graphs as defined in Definition 5 is an extension to the class of temporal graphs as defined in the literature. We can better understand this class of graphs as follows. Let $G$ be a graph falling in this class. $G$ may or may not have well defined set of vertices $V(G)$ where $V(G) = V_1 \cup V_2 \cup ... \cup V_\ell$. But by the time an algorithm such as Algorithm terminates, we will obtain a well defined set of vertices $V(G)$ such that for each $v \in V(G)$, $v$ is also an element of some $V_i$ such that $G_i : G_i = (V_i, E_i)$ is an instance of $G$. For further formal considerations, we assume that we have a defined set $V(G)$. The sets $V_i : 1 \leq i \leq \ell$ may be decided by an algorithm (Algorithm 4 in our case), or may be provided as part of the input. Similarly, the sets $E_i : 1 \leq i \leq \ell$ may be decided by an algorithm (in our case, $\text{Instance}(G)$ is assigning edges to $V_i$ for Algorithm 4), or may be provided as part of the input. At a time-step $i$, any set $E_i : E_i \subseteq V(G) \times V(G)$ is defined such that no vertex in $V(G) \setminus V_i$ takes part in forming an edge at that time-step. Such a definition has practical applications as it allows the flexibility to the system that it can restrict some of none of the nodes in $V(G)$ to not to take part in any communication at some time step; we have already demonstrated a few related examples earlier in this section.

10 Related work

In the literature, the trend of the spread of a virus through biological systems is studied using several efficient models. In addition to this, spread of a virus through hosts (computational systems), spread of a meme, or other contagion in networks is studied or opined [4].

Graph burning was first studied in [6] with a notion that only one new fire source is initiated in each time step, and in each time-step the fire spreads as well. Graph burning has been shown to be NP-Complete in [4, 23, 22] and has been studied several times in [6, 7, 5, 17, 21, 26, 27, 29, 35]. Graph burning where more than one (but a constant number of) nodes has been studied in [33].

The notion of the firefighter problem was first described in [24]. This problem was also discussed on static graphs and with a model in which 1 firefighter is to be placed in each time-step. The firefighting problem is NP-Complete [18, 20, 30] and has been studied several times in the literature [3, 2, 8, 11, 12, 13, 14, 14, 25].
In addition, temporal graphs is a concept which is significant for this article, and has been studied intensively. Temporal graphs were first discussed in [28]. Since then, several works have been done on temporal graphs [9, 10, 15, 16, 31, 32, 34].

This article introduces the notion of firefighting with multiple firefighters bring placed on nodes in a single time-step. This article introduces the notion of variable firefighting and variable burning. Also, this article introduces a model in which burning and firefighting are analyzed together, them being used against each other. We allow firefighting to save infected nodes, which represents “repair” of an infected nodes, along with firefighting of uninfected nodes, which represents “upgrade” of healthy nodes. This makes graph burning [6] and $u$-burning [33] special cases of this model where a constant number of nodes are burned from outside. This article also introduces a graph class as defined in Definition 5, which an extension of temporal graphs in which number of nodes are also variable: nodes can be both added or removed. The model that we presented in this article is an implementation of this graph class. Temporal graphs are a special case of this model, where (a) $\rho_{\text{del}} = 0$, and (b) $\rho_{\text{ins}} = 0$. This also makes static graphs a special case of this model, where (a) $\rho_{\text{del}} = 0$, (b) $\rho_{\text{ins}} = 0$, and (c) the probability of existence of edges is either 1 or 0. From the perspective of the class of graphs defined in Definition 5, we can obtain temporal graphs by setting $\forall i, j: 1 \leq i, j \leq \ell, V_i = V_j$, and we can obtain static graphs by setting $\forall i, j: 1 \leq i, j \leq \ell, V_i = V_j \land E_i = E_j$.

11 Conclusion

The model the we present in this article may be viewed as a complex fusion of graph burning (introduced in [6]) and firefighting (introduced in [24]) on a variation of temporal graphs (temporal graphs introduced in [28]) where each instance $G'$ of the underlying graph $G$ is a probabilistic graph such that we introduce probability on the insertion of new nodes to the underlying graph $G$ as well as on the deletion of nodes once they get infected, along with only having probabilities on every edge. This is one of the variations of the graph class as defined in Definition 5.

The enforcement of the probabilities in our model, however, remains simple and is based on uniform probability distribution. Several other complex probability distributions can be enforced based on the nature of system being simulated.

This model can be further used to study for different (heuristic) algorithmic strategies of a quantification of repairs and upgrades (firefighting) required, as well as to test strategies for spread of infection among the nodes (burning), independently, or (burning and firefighting) against each other. In this article we presented a system in which the notions of burning and firefighting have been modelled against each other. In the application system that we have demonstrated our model on, we prefer firefighting to “win” over burning, that is, the desired trace property in this example is “eventually, for each node $v$ in
the network $G, \neg v.i.$ holds true”. Such preferences may also change based on the system being studied.

This model can also be studied on larger systems such as cellular networks, molecular networks, human social networks or other biological or ecological systems and study several aspects of this model including its reliability and robustness on general networks.

We have also established an introductory theory of a more diverse class of graphs. The class of graphs as defined in Definition 5 is an advancement to the existing definition of the class of temporal graphs; it allows a larger set of systems to be represented formally and modelled theoretically. The model that we have presented in this article works on these graphs: static graphs or temporal graphs would be insufficient for this model. We call this model bvfa, which stands for variable burning versus firefighting on Altered Temporal Graphs with Generalization.

References

[1] Bowen Alpern and Fred B. Schneider. Defining liveness. Information Processing Letters, 21(4):181 – 185, 1985.

[2] Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. Journal of Computer and System Sciences, 80(7):1285 – 1297, 2014.

[3] Cristina Bazgan, Morgan Chopin, and Michael R. Fellows. Parameterized complexity of the firefighter problem. In Takao Asano, Shin-ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, Algorithms and Computation, pages 643–652, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] Stéphane Bessy, Anthony Bonato, Jeannette Janssen, Dieter Rautenbach, and Elham Roshanbin. Burning a graph is hard. Discrete Applied Mathematics, 232:73 – 87, 2017.

[5] Anthony Bonato. A survey of graph burning, 2020.

[6] Anthony Bonato, Jeannette Janssen, and Elham Roshanbin. Burning a graph as a model of social contagion. In In: Bonato A., Graham F., Pralat P. (eds) Algorithms and Models for the Web Graph. WAW 2014. Lecture Notes in Computer Science., pages 13–22. Springer International Publishing, 2014.

[7] Anthony Bonato and Thomas Lidbetter. Bounds on the burning numbers of spiders and path-forests. Theoretical Computer Science, 794:12 – 19, 2019. Special Issue on Theory and Applications of Graph Searching.
Leizhen Cai, Yongxi Cheng, Elad Verbin, and Yuan Zhou. Surviving rates of graphs with bounded treewidth for the firefighter problem. *SIAM Journal on Discrete Mathematics*, 24(4):1322–1335, 2010.

Arnaud Casteigts, Ralf Klasing, Yessin M Neggaz, and Joseph Peters. Computing Parameters of Sequence-Based Dynamic Graphs. *Theory of Computing Systems*, 63(3):394–417, April 2019.

Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster Editing in Multi-Layer and Temporal Graphs. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Xujin Chen, Xiaodong Hu, Changjun Wang, and Ying Zhang. Continuous firefighting on infinite square grids. In T.V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Theory and Applications of Models of Computation*, pages 158–171, Cham, 2017. Springer International Publishing.

Pierre Coupechoux, Marc Demange, David Ellison, and Bertrand Jouve. Firefighting on trees. *Theoretical Computer Science*, 794:69 – 84, 2019. Special Issue on Theory and Applications of Graph Searching.

Marek Cygan, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting revisited. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation*, pages 13–26, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Bireswar Das, Murali Krishna Enduri, Masashi Kiyomi, Neeldhara Misra, Yota Otachi, I. Vinod Reddy, and Shunya Yoshimura. On structural parameterizations of firefighting. *Theoretical Computer Science*, 782:79 – 90, 2019.

Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9810–9817, Apr. 2020.

Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 444–455, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

Zahra Rezai Farokh, Maryam Tahmasbi, Zahra Haj Rajab Ali Tehrani, and Yousof Buali. New heuristics for burning graphs, 2020.

Stephen Finbow, Andrew King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discrete Mathematics*, 307(16):2094 – 2105, 2007. EuroComb ’03 - Graphs and Algorithms.
[19] Stephen Finbow and Gary Macgillivray. The firefighter problem: A survey of results, directions and questions. *The Australasian Journal of Combinatorics [electronic only]*, 43, 02 2009.

[20] Fedor V. Fomin, Pinar Heggernes, and Erik Jan van Leeuwen. The firefighter problem on graph classes. *Theoretical Computer Science*, 613:38 – 50, 2016.

[21] Rahul Kumar Gautam, Anjeneya Swami Kare, and S. Durga Bhavani. Faster heuristics for graph burning, 2020.

[22] Arya Tanmay Gupta. Burning geometric graphs. mathesis, Indian Institute of Information Technology Vadodara, India, July 2020.

[23] Arya Tanmay Gupta, Swapnil A. Lokhande, and Kaushik Mondal. Burning grids and intervals. In Apurva Mudgal and C. R. Subramanian, editors, *Algorithms and Discrete Applied Mathematics*, pages 66–79, Cham, 2021. Springer International Publishing.

[24] B. Hartnell. Firefighter! an application of domination. *the 24th Manitoba Conference on Combinatorial Mathematics and Computing, University of Manitoba, Winnipeg, Canada, 1995*, 1995.

[25] Bert Hartnell and Qiyan Li. Firefighting on trees: How bad is the greedy algorithm? *Congressus Numerantium*, 145, 01 2000.

[26] Shahin Kamali, Avery Miller, and Kenny Zhang. Burning two worlds. In Alexander Chatzigeorgiou, Riccardo Doudi, Herodotos Herodotou, Christos Kapoutsis, Yannis Manolopoulos, George A. Papadopoulos, and Florian Sikora, editors, *SOFSEM 2020: Theory and Practice of Computer Science*, pages 113–124, Cham, 2020. Springer International Publishing.

[27] Anjeneya Swami Kare and I. Vinod Reddy. Parameterized algorithms for graph burning problem. In Charles J. Colbourn, Roberto Grossi, and Nadia Pisanti, editors, *Combinatorial Algorithms*, pages 304–314, Cham, 2019. Springer International Publishing.

[28] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820 – 842, 2002.

[29] Huiqing Liu, Ruiting Zhang, and Xiaolan Hu. Burning number of theta graphs. *Applied Mathematics and Computation*, 361:246 – 257, 2019.

[30] Gary MacGillivray and Ping Wang. On the firefighter problem. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 47, 01 2003.

[31] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.
[32] Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1 – 23, 2016.

[33] Debajyoti Mondal, N. Parthiabn, V. Kavitha, and Indra Rajasingh. Apx-hardness and approximation for the k-burning number problem, 2020.

[34] Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The Complexity of Finding Small Separators in Temporal Graphs. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[35] Marek Šimon, Ladislav Huraj, Iveta Dirgová Luptáková, and Jiří Pospíchal. Heuristics for spreading alarm throughout a network. *Applied Sciences*, 9(16), 2019.