Evaluating Worst-Case Epidemic Spread Using Network-Based Modeling

Quintin Pope

June 12, 2020

Abstract

Typical epidemiological modeling aims to produce a probability distribution over the extent of possible disease spread, often by simulating multiple epidemics and recording the distribution over the total number infected. Such an approach runs the risk of missing low probability, high impact scenarios in which "just the wrong people" are infected, and the disease spreads more quickly than expected. I provide a tool to investigate such "worst case" epidemics using a network-based approach that identifies subsets of people with the potential to spread infection rapidly through the network. I additionally try to explore preventing such worst case outcomes through selective vaccination. I provide a codebase for simulating epidemics on graphs and generating SMT reductions of the above problems.

1 Introduction

The simplest epidemiological models predict disease spread using a geometric relation between the number of infected and disease spread rate [KMW27]. However, real-world disease spread depends strongly on the social contacts of the infected. As a result, epidemics can deviate greatly from the geometric progression predicted by simpler models [LSSKG05]. I provide a tool to evaluate the possible magnitude of such deviations evaluating contact networks and identifying subsets of people that, if infected, maximally spread disease. Specifically, the tool accepts (or generates) a contact network, *limit* as the number of initial infected, and *target* as the target number of final infected. The tool then determines whether it's possible to initially infect no more than *limit* nodes and eventually infect at least *target* nodes. It is thereby possible to use multiple calls to find the minimum *limit* that infects at least *target* nodes.

I investigate the algorithm's running time as a function of the numbers of nodes and edges in the graph, *limit*, and *target*. I find that for *limit* $<<$ the number of nodes, worst case run time (meaning there's no solution and the tool has to check every possible solution) increases approximately polynomially with the number of edges and the number of nodes, decreases exponentially (or faster) with *target*, and increases exponentially with *limit*. However, run time also depends on the structure of the graph. In particular, if the graph allows for many solutions, evaluation may stop well short of the worst case.

Additionally, I attempt to program the tool to solve the vaccination problem, where we seek to prevent worst case outbreaks by vaccinating a small set of key individuals. I think I am able to generate correct quantified SMT reductions for this problem, but evaluations of the resulting formula do not produce the expected results.

2 Implementation

To simulate plausible worst case epidemics, we need plausible contact networks. Differences in employment, family size, travel patterns and other relevant parameters mean that contact networks differ significantly between regions [PCJ17]. Ideally, I'd use real-life contact networks as a starting point and then generate similarly structured contact networks using a method such as Replication of Complex Networks (ReCoN) [SHS$^+$16].

Unfortunately, researchers appear hesitant to publicly provide real-life contact networks (or I have no idea where to find them). Instead, I use the NetworkX [HSSC08] implementation of the Watts–Strogatz algorithm [WS98]. Given parameters integer $|V|$, even integer m, and probability

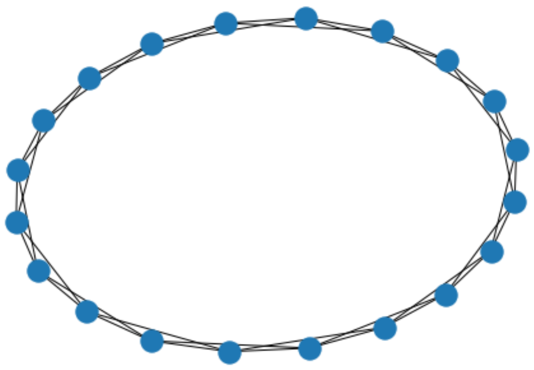


Figure 1: A Watts–Strogatz during the ring construction phase (n = 20, m = 4).

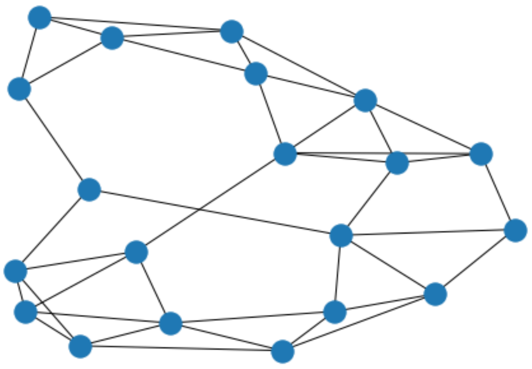


Figure 2: A Watts–Strogatz after connections are randomly re-assigned (n = 20, m = 4, p = 0.1).

p, it first creates a ring of $|V|$ nodes and connects each node to m of its nearest neighbors. Then for each connection, with probability p it replaces that connection with a connection to a randomly chosen node. See figures 1 and 2.

Although there is no guarantee that Watts–Strogatz graphs will closely model the population distribution of any particular region, they do have properties that make them better suited to modeling epidemics than random networks, such as a high degree of local clustering and relatively short path lengths [WS98]. Additionally, Watts–Strogatz graphs have $|V| \times m$ edges, so the number of edges scales linearly with $|V|$.

Let G = (V, E) be a graph representing the social connections present in a population. Then, we can predict how a disease might spread from any initial infectee to the rest of the network. Given a starting point s, total timesteps T, and probability of spread per timestep p, we sequentially simulate each timestep. See algorithm 1.

Using this method, we can construct a directed graph, G' = (V, E') that has an edge from node i to node j if we predict that an infection started at node i would reach node j within T timesteps. Note that for $p < 1$, this approach is stochastic and will produce different distributions of infection with each run.

Once we've generated G', we can try to find some S ⊆ V such that:

- $|S| \leq limit$
- if all of S start infected, total infected after T timesteps $\geq target$

This is equivalent to a directed partial vertex cover problem on G', which is an NP-complete problem. I solve this problem by reducing it to another NP-complete problem: satisfiability modulo theories (SMT) and using pySMT [GM15] to solve the reduced problem. In the directed partial vertex cover problem, we're given a directed graph G' = (V, E') and integers limit and target. We want to know if it's possible to find some subset S of V such that:

$$|S| \leq limit$$

Algorithm 1: Simulate spread on graph

```
Currently_Infected = [s]
G' = (V, E')
// E' starts empty
for each node i in $Currently_Infected$ do
    for each uninfected neighbor n of i in G do
        if Random_Real(0, 1) > p then
            Add n to Currently_Infected
            Create an edge from s to n
        end
    end
end
```

$$|\bigcup_{i \in S} Outneighbors_i| \geq target$$

Given this problem definition, we can construct an SMT formula that is satisfiable if and only if the partial vertex cover problem is satisfiable. We start by defining two indicator variables for each node i: init_i and final_i. init_i will be set to one if node i is chosen as an element of S. final_i will be set to one if node i is in the vertex cover of S. To enforce this behavior, I define the following clauses for each node:

- If init_i = 1, then final_i = 1 and for each node j in outneighbors$_i$, final_j = 1
- If final_i = 1, then init_i = 1 or at least one node j in inneighbors$_i$ has init_j = 1
- Note: each of the above represents a single clause per variable, so two clauses per node.

Combined, these clauses ensure that final_i = 1 if and only if init_j = 1 for some j, where node j is a predecessor of node i. Additionally, I enforce the constraints implied by limit and target using the following clauses:

- $\sum_{i=0}^{|V|} init\_i \leq limit$
- $\sum_{i=0}^{|V|} final\_i \geq target$
- Note: these represent a total of two additional clauses.

Then, the full SMT formula is formed by requiring simultaneous satisfaction of all clauses.
Proof of reduction:

- Suppose $\exists$ some assignment of values to $\{init\_i_i\}$ and $\{final\_i_i\}$ such that they satisfy the full SMT formula.
  - Then, this assignment corresponds to a covering with the following properties.
  - $\forall i$,init_i= 1 $\implies$ node i and all its outneighbors are infected.
  - $\forall i$,final_i= 1 $\implies$ node i or at least one of its inneighbors are infected.
  - The total number of initial infected is less than or equal to limit.
  - The total number of final infected is greater than or equal to target.
  - Therefore, the assignment satisfies all constraints of a valid covering.
  - Therefore, $\exists$ some solution to the directed partial vertex cover problem.
- Suppose $\exists$ some S $\subseteq$ V such that S is a solution to the partial directed vertex cover problem.
  - Then, we can construct a solution to the SMT problem from S.
  - Set $\{init\_i_i\}$ = 1 if node i is initially infected

  - Set $\{final\_i_i\} = 1$ if node i is ever infected.
  - Then, the SMT assignment will satisfy the following constraints.
  - If init_i = 1, then final_i = 1 and for each node j in outneighbors$_i$, final_j = 1
  - If final_i = 1, then init_i = 1 or at least one node j in inneighbors$_i$ has init_j = 1
  - $\sum_{i=0}^{|V|} init\_i \leq limit$
  - $\sum_{i=0}^{|V|} final\_i \geq target$
  - Therefore, this assignment satisfies all the constraints of the SMT problem.
  - Therefore, $\exists$ some solution to the SMT problem.

- Therefore, iff $\exists$ a solution to the directed partial vertex cover problem, then $\exists$ a solution to the SMT reduction.
- Therefore, the reduction is valid.

To simplify the proof, I pretended that the variables were booleans. However, pySMT requires any summed variable to be an integer or a real number. As a result, init_i and final_i are integers, and each have two additional clauses that constrain their values to be either 0 or 1. The total number of clauses in a formula is thus $6 \times |V| + 2$.

I additionally attempted to explore whether it's possible to prevent such worst case spread with targeted vaccinations. Given a directed graph G', we can ask whether there exists some bounded size subset of V, R, such that:

- $|R| \leq vaccine\_limit$
- $\forall$ S (|S| $\leq limit$), total infected after t timesteps $< target$

Finding such an R in G' is an example of the quantified partial directed vertex cover problem, which is in Σ_2^P. To solve this problem, I reduce it to the Σ_2^P extension of SMT: quantified SMT. Suppose spread_formula an SMT formula for the unquantified partial directed vertex cover problem. We can construct a formula for the quantified version of the problem by defining an additional variable, vacc_i for each node i, then constructing the following set of clauses:

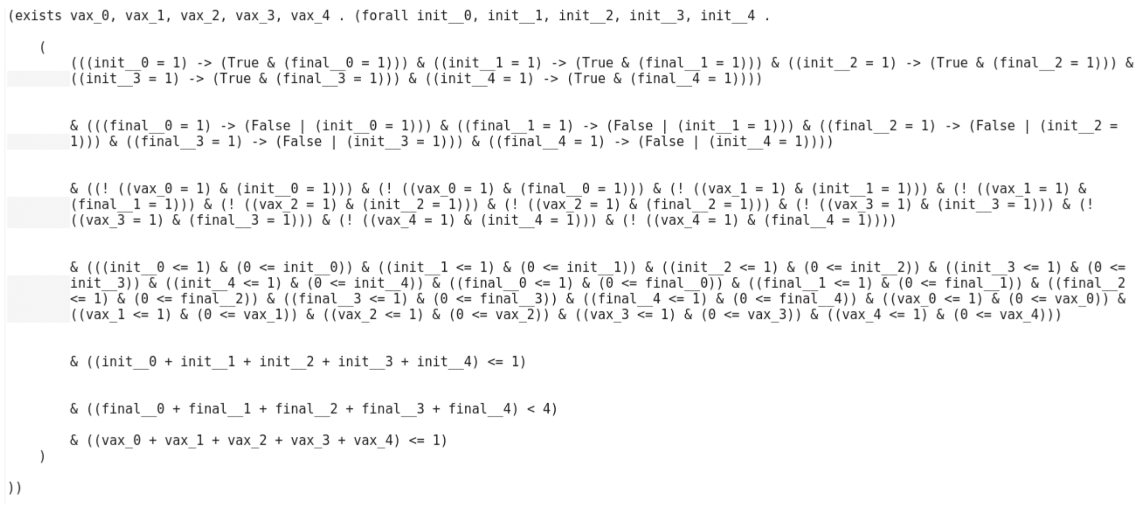
- For each node i, $\neg$ (vacc_i = 1 and init_i = 0)
- For each node i, $\neg$ (vacc_i = 1 and final_i = 1)
- $\sum_{i=0}^{|V|} vacc\_i \leq vaccine\_limit$

I then construct the SMT formula for the quantified SMT problem requiring satisfaction of all the clauses described above. Then, I use pySMT's Exists() and ForAll() methods to evaluate whether there exists an assignment of values to $\{vacc\_i\}_i$ such that for all assignments of values to $\{init\_i\}_i$:

$$\sum_{i=0}^{|V|} final\_i < target$$

Because vacc_i is also an integer, I use another two clauses per node. The total number of clauses is thus $10 \times |V| + 3$.

Although the generated formula are correct as far as I can tell, pySMT always evaluates the reduction as unsatisfiable, even when there are trivial solutions. For example, 3 is a quantified SMT formula for a graph with 5 nodes, no edges (meaning no disease spread), one vaccination, a target of 4 and a limit of 1. Satisfying the formula should be easy because it's impossible for there to be more than one final infection regardless of which node is vaccinated, but pySMT reports the formula is not satisfiable.



```
(exists vax_0, vax_1, vax_2, vax_3, vax_4 . (forall init__0, init__1, init__2, init__3, init__4 .

    (
        (((init__0 = 1) -> (True & (final__0 = 1))) & ((init__1 = 1) -> (True & (final__1 = 1))) & ((init__2 = 1) -> (True & (final__2 = 1))) &
        ((init__3 = 1) -> (True & (final__3 = 1))) & ((init__4 = 1) -> (True & (final__4 = 1))))


        & (((final__0 = 1) -> (False | (init__0 = 1))) & ((final__1 = 1) -> (False | (init__1 = 1))) & ((final__2 = 1) -> (False | (init__2 =
        1))) & ((final__3 = 1) -> (False | (init__3 = 1))) & ((final__4 = 1) -> (False | (init__4 = 1))))


        & ((! ((vax_0 = 1) & (init__0 = 1))) & (! ((vax_0 = 1) & (final__0 = 1))) & (! ((vax_1 = 1) & (init__1 = 1))) & (! ((vax_1 = 1) &
        (final__1 = 1))) & (! ((vax_2 = 1) & (init__2 = 1))) & (! ((vax_2 = 1) & (final__2 = 1))) & (! ((vax_3 = 1) & (init__3 = 1))) & (!
        ((vax_3 = 1) & (final__3 = 1))) & (! ((vax_4 = 1) & (init__4 = 1))) & (! ((vax_4 = 1) & (final__4 = 1))))


        & (((init__0 <= 1) & (0 <= init__0)) & ((init__1 <= 1) & (0 <= init__1)) & ((init__2 <= 1) & (0 <= init__2)) & ((init__3 <= 1) & (0 <=
        init__3)) & ((init__4 <= 1) & (0 <= init__4)) & ((final__0 <= 1) & (0 <= final__0)) & ((final__1 <= 1) & (0 <= final__1)) & ((final__2
        <= 1) & (0 <= final__2)) & ((final__3 <= 1) & (0 <= final__3)) & ((final__4 <= 1) & (0 <= final__4)) & ((vax_0 <= 1) & (0 <= vax_0)) &
        ((vax_1 <= 1) & (0 <= vax_1)) & ((vax_2 <= 1) & (0 <= vax_2)) & ((vax_3 <= 1) & (0 <= vax_3)) & ((vax_4 <= 1) & (0 <= vax_4)))


        & ((init__0 + init__1 + init__2 + init__3 + init__4) <= 1)


        & ((final__0 + final__1 + final__2 + final__3 + final__4) < 4)

        & ((vax_0 + vax_1 + vax_2 + vax_3 + vax_4) <= 1)
    )

))
```

Figure 3: A quantified SMT formula with |V| = 5, no edges, target = 4, limit = 1, and one vaccination.

2.1 Interface

The tool is provided as either a github repository or a Google colab notebook. If used from the repository, the tool is called using the decide_epidemic() and decide_quant_epidemic() method. decide_epidemic() accepts the arguments 'source', 'size', 'm', 'p', 'limit', 'target', 'timesteps', and 'p_infect'.

'source' can either be a path to a NetworkX graph adjlist file for an undirected graph or the string 'generate'. If it's the string 'generate', then the tool generate a Watts-Strogatz graph using the provided size, m and p parameters. Otherwise, it ignores size, m and p. 'limit' and 'target' decide the limit for initial infected and the target final infected, respectively. 'timesteps', and 'p_infect' control how many jumps the infection is simulated to perform, in accordance with algorithm 1. Calls to decide_epidemic() return a tuple containing a boolean indicating the satisfiability of the associated SMT problem and an assignment of variables for the SMT problem. decide_quant_epidemic() has the same arguments as decide_epidemic(), with the addition of 'n_vaccinate', which determines the number of allowed vaccines. decide_quant_epidemic() returns a tuple containing a boolean indicating the satisfiability of the associated quantified SMT problem and an assignment of variables for the problem.

3 Performance

I evaluate run time for the worst-case performance, meaning problem instances where there is no solution, and the program must check every possible assignment to return its answer. Run time is a function of |V|, |E|, limit, target and the graph structure together. Because it is not feasible to evaluate run time across all possible variations of these parameters, I'll show plots where one of these parameters varies (typically for small values of limit and |E| = 0) and extract an approximate asymptotic trend from that. Note: all plots displayed use logarithmic scaling on their y-axes.

I generate random graphs without any edges, meaning epidemics can't spread. I use limits less then their target numbers (thus ensuring no solution). Figures 4 and 5 show the impact that varying the target number has on run time. Figure 4 uses a limit value of 1 while 5 uses a limit of 2. Their x-axes show the "target fraction", meaning the target value for total infected is set to that fraction × the graph size. These graphs are approximately linear on a log plot, indicating that the run time decreases at least exponentially as the target number increases, with potentially faster than exponential decrease for larger graphs with two initial infected. In future plots, I use a fraction value of 0.9 to speed up experiments.

I repeat the same process, but this time include edges in the graphs. Specifically, I use algorithm 1 to simulate epidemic spread for 0, 1, 2, or 3 steps, with a probability of spread per step of 0.5 and two initial infected. The roughly linear plots suggest that run time increases exponentially with simulation steps. However, |E| increases exponentially with simulation steps because each

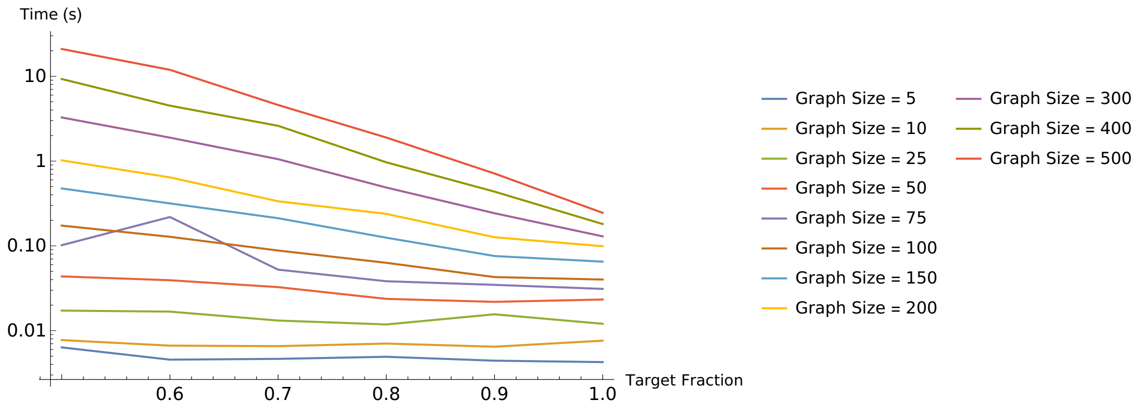


Figure 4: Plot of run time vs fraction with single initial infection (target total infections = fraction × |V|).

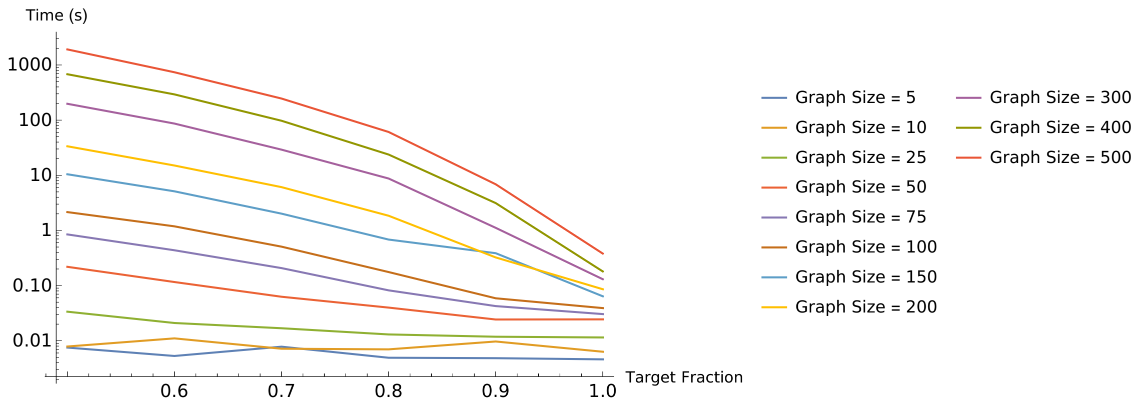


Figure 5: Plot of run time vs fraction with two initial infections (target total infections = fraction × |V|).

step of algorithm 1 increases the final edge count geometrically. This suggests run time is roughly polynomially proportional to |E|.

Figure 7 plots run time vs the limit with |E| = 0. This plot suggests run time increases exponentially with limit. Figure 8 uses the same data as Figure 7 to show the impact of |V| on run time. The sub-linear curves suggest run time increases polynomially with graph size. Note that increasing |V| also linearly increases the number of clauses (in fact, it's the only way to change the number of clauses). Therefore, this plot also shows that run time increases polynomially with clause count.

Overall, the feasibility of a given problem strongly depends on the parameters described above. In particular, increasing the limit value past 4 or 5 on a large graph will quickly lead to an infeasible worst-case run time. The graph structure also impacts run time. These worst case trends don't necessarily hold for graphs whose structure allows for an easy solution. As a result, it may be feasible, for example, to run the tool for a predetermined time and treat the problem as unsatisfiable if no solution emerges in that time.

4 Conclusion

I provide a tool for estimating the maximum severity of an epidemic on a given contact network by reducing the problem to SMT. I show general trends in worst case run time as a function of |V|, |E|, initial infected limit and final infected target. Although run time considerations prevent exact solutions for larger graphs, useful approximations are likely possible.

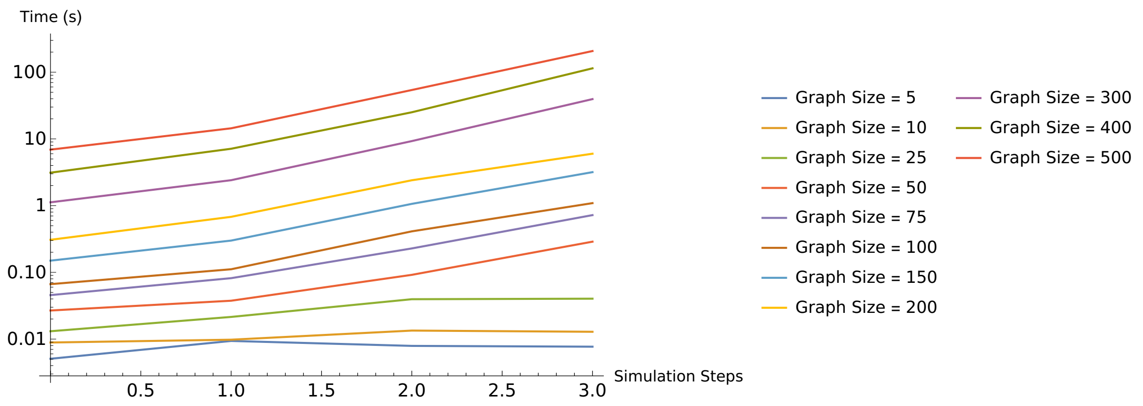


Figure 6: Plot of run time vs simulation steps with two initial infections (probability of infection per step = 0.5).

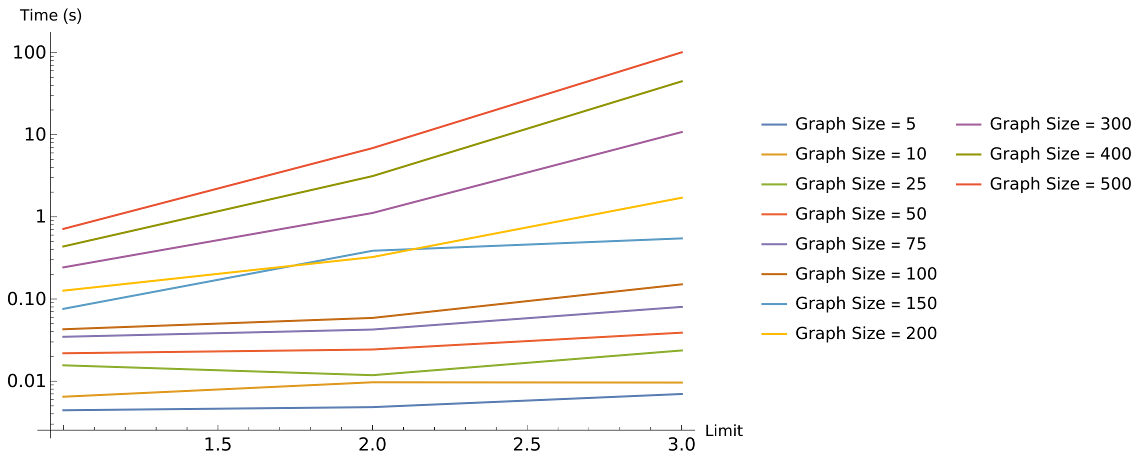


Figure 7: Plot of run time vs limit on initial infection number with no spread.

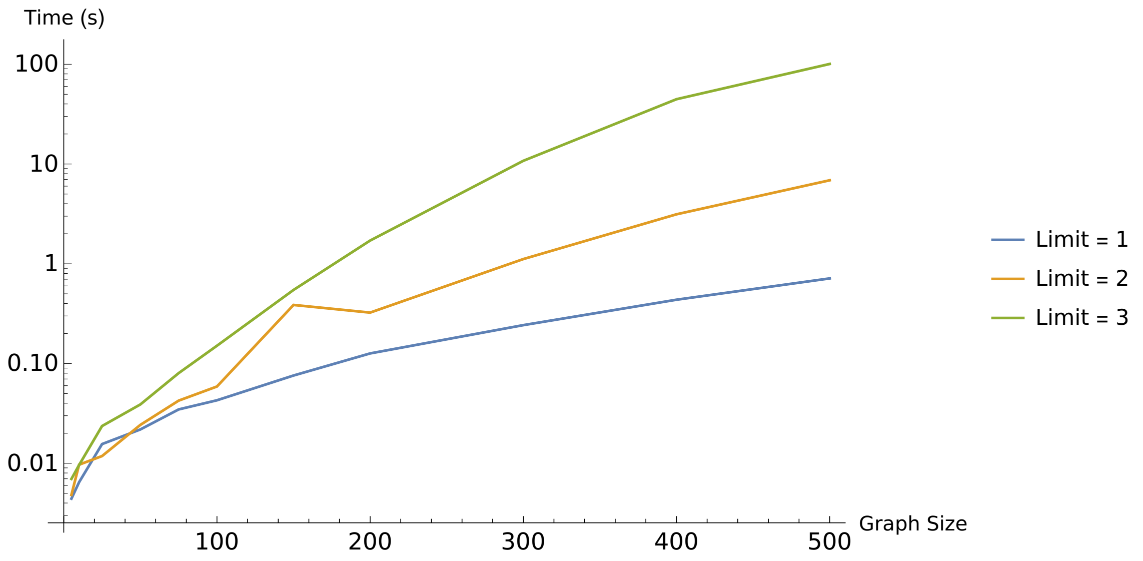


Figure 8: Plot of run time vs graph size with no spread.

References

[GM15] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In *SMT Workshop 2015*, 2015.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[KMW27] William Kermack, A. G. McKendrick, and Gilbert Walker. A contribution to the mathematical theory of epidemics. *Proc. R. Soc. Lond. A*, 115, 8 1927.

[LSSKG05] James Lloyd-Smith, Sebastian Schreiber, P.E. Kopp, and Wayne Getz. Superspreading and the effect of individual variation on disease emergence. *Nature*, 438:355–9, 12 2005.

[PCJ17] Kiesha Prem, Alex R. Cook, and Mark Jit. Projecting social contact matrices in 152 countries using contact surveys and demographic data. *PLOS Computational Biology*, 13(9):1–21, 09 2017.

[SHS$^+$16] Christian Staudt, Michael Hamann, Ilya Safro, Alexander Gutfraind, and Henning Meyerhenke. Generating scaled replicas of real-world complex networks. 09 2016.

[WS98] Duncan Watts and Steven Strogatz. Collective dynamics of 'small-world' networks. *Nature*, page 440–442, 6 1998.