Modelling infectious disease dynamics with probabilistic logic programming

Felix Weitkämper[a], Beatrice Sarbu[a] and Kailin Sun[b]

[a]Institut für Informatik der LMU München, Oettingenstr. 67, 80538 München, Germany
[b]Department Biologie I der LMU München, Menzinger Str. 67, 80638 München, Germany

Abstract

Motivated by the SARS-CoV-2 pandemic, we implemented a stochastic, network-based model of infectious disease transmission in the probabilistic logic programming language ProbLog. In this contribution, we show how probabilistic logic programming lends itself to very concise, transparent and adaptable modelling of infectious disease dynamics. We illustrate how some key features can contribute to succinct and expressive representation of epidemiological models. Our work makes full use of the compact relational representation, the support for stratified negation and flexible probabilities evaluated at run time, which are supported by the ProbLog 2 engine.

Keywords

ProbLog, SI-model, Disease Dynamics, Epidemiology Probabilistic programming

1. Introduction

Motivated by the SARS-CoV-2 pandemic and its societal impact, we implemented stochastic models of infectious disease dynamics in the declarative probabilistic logic programming language ProbLog [1]. In this contribution, we demonstrate how the features of ProbLog contribute to a concise and transparent implementation.

To the best of our knowledge, this is the first described use of probabilistic logic programming for epidemiological modelling. Its suitability for expressing network relationships is well-supported, however, and indeed link prediction in biological networks was used as a motivation in the original publication of the ProbLog language [1, 2, 3, 4, 5].

Epidemiological models fall into two broad categories: compartmental and network-based models. The paradigmatic example of a compartmental model is Kermack and McKendrik's classic susceptible-infected-recovery (SIR) model [6]. This model, illustrated in Figure 1, divides the population into three groups: susceptible (individuals who are not immune to the disease), infected (individuals who currently carry the disease and are able to spread it), and recovered (individuals who have recovered and are therefore immune). Differential equations model the transition of the proportion of the population moving from one group to another during each time step. There are certain assumptions of SIR models which may not be realistic in every

PLP 2021
felix.weitkaemper@lmu.de (F. Weitkämper); B.sarbu@campus.lmu.de (B. Sarbu); sun.kailin@campus.lmu.de (K. Sun)
0000-0002-3895-8279 (F. Weitkämper)

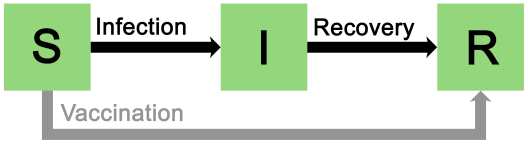

Figure 1: A typical SIR model, showing **S**usceptible, **I**nfected and **R**ecovered groups with their modes of transition

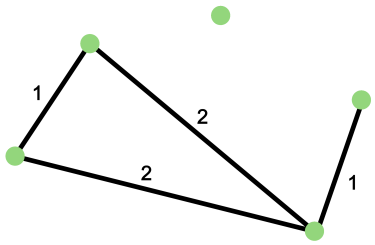

Figure 2: An example of a weighted network

situation; for example, SIR models assume a well-mixed, homogeneous, and large population [7]. Intermittent lockdowns, travel restrictions and changes in social behaviour over the course of the SARS-CoV-2 pandemic mean that populations are not always well-mixed. Individual differences in vulnerability to SARS-CoV-2 (and whether or not individuals are asymptomatic) negate the assumption of homogeneity. Indeed, we can assume that one or more of the above assumptions have been violated, since there is evidence that that the spread of SARS-CoV-2 is fat-tailed [8]. In addition, a model based on the assumption of a large population may not be helpful in modelling the spread of disease on a smaller scale, for example when combatting local outbreaks.

In response to those limitations, epidemiologists developed stochastic or network models, in which disease is transmitted along the edges of a contact graph. It is such a stochastic approach that we are implementing here. A network-based model incorporates the idea of the SIR groups into a local network-based structure. Since the model is probabilistic, random effects in the spread of disease are accounted for. Network-based modelling also removes the assumption that there is an infinite population size. Instead, individuals are modelled as nodes in an undirected graph, with edges between individuals that have contact. The disease then spreads by infection along the nodes of the graph in discrete time steps. To model different levels of contact, one can use a weighted graph, in which the infection probability depends on the weight. An example network is shown in Figure 2. For a thorough introduction to network-based models, see [9].

We have chosen to use ProbLog as a declarative probabilistic programming language since it grants immediate and transparent access to the parameters and underlying assumptions of the model. This is particularly important when modelling such a new disease as COVID-19, where medical and epidemiological knowledge changes rapidly, and there is considerable uncertainty about the appropriate values for the various modelling parameters. In addition, probabilistic

logic programming allows for a very compact representation of the underlying processes.

Among probabilistic logic programming languages, ProbLog stands out with its well-maintained ProbLog 2 engine [10], which also supports useful additional features such as flexible probabilities. Additional parameters and assumptions can be appended easily, so the model adapts to changing demands during the course of a pandemic. ProbLog 2 is also able to generate simulations as well as to calculate exact probabilities of infection, increasing the versatility of our approach [11].

As far as we know, current declarative approaches to modelling infectious disease dynamics are limited to compartmental rather than network-based models and are based on dedicated domain-specific languages. Those DSLs are themselves implemented imperatively as part of an integrated simulation suite. A state-of-the-art example of this is the EMULSION framework [12]. While such a DSL can be tailored to the application domain, it is more difficult to extend or verify its ad hoc implementation machinery. In contrast, our modelling can rely on the general purpose implementation ProbLog 2 of the ProbLog probabilistic logic programming language.

We present three examples of our modelling approach in the following sections. First, we explain the basic approach using a simple network model based on class enrolment data, with a certain probability of infection between classmates. The basic set-up already showcases the modularity and brevity of the code. Then, we introduce an extension to accommodate for resistance to the disease, which could be acquired through surviving the illness or through a vaccination programme. This uses stratified negation as implemented in ProbLog. In the Section 4, we discuss an alternative model based on distance metrics, which takes into account the precise location of the individuals. This makes use of flexible probabilities as a powerful and flexible modelling tool.

2. Epidemiological modelling based on SI-Model

In the basic model, we assume a closed population divided into two groups: susceptible individuals and infected individuals. Infected individuals contribute to the spread of the virus. Once a susceptible individual is infected, he or she belongs to the infected individuals, and once the infectivity period has lapsed, they move back to the susceptible group. We assume a closed world of individuals. Since our motivation comes from an academic setting, with its regular recurring intervals of lecturing weeks, discrete time steps are ideally suited to model our domain.

At the beginning of the period under consideration, most people are still susceptible. Through contact with infected individuals, they may become infected with a certain probability and may also spread the virus during the next week. In our ProbLog implementation, the epidemiological parameters are completely separated from the individuals and their mutual interaction. This makes our model very agile, as it can rapidly be adapted to different models of infectious disease transmission or different connections of individuals, and makes maintenance of the code much easier.

The entire infection logic of our implementation is contained in the following code:

Listing 1: SI infection logic

```
1  % In every given week, there is a 1 percent chance of infection outside of the setting.
```

```
0.01::ill(N,STUDENTx) :- student(STUDENTx), week(N).
0.01::ill(N,STUDENTx) :- student(STUDENTx), week(N), vulnerable(STUDENTx).

% 10 percent of students are particularly vulnerable to disease.
0.1::vulnerable(STUDENTx) :- student(STUDENTx).

% The average period of infectivity is two weeks.
0.5::ill(B, STUDENTx):- ill(A,STUDENTx), week(B), B is A+1.

% A student has a 5 percent chance of infection from a diseased classmate.
0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy, COURSE),  week(B),
B is A+1, vulnerable(STUDENTx).

0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy, COURSE),  week(B),
B is A+1.
```

The network of relations between different students was implemented by supplying enrolment data as a Prolog knowledge base and adding the following clauses to compute the network. The predicate `classmates/3` maps the relationship between persons and their lectures under the following clause:

Listing 2: Network of connections

```
classmates(PERSONx ,PERSONy, COURSE) :-
participate(COURSE, PERSONx), participate(COURSE, PERSONy), PERSONx \== PERSONy.
```

If two persons are attending the same course and the first person is different from the second person, then they are classmates. Note that this naturally supports a structure in which students attending more than one course together will have an independent probability in each class to infect each other. Those different probabilities of infection, together with the additional extrinsic infection risk, are internally processed with a *noisy-or* combination function; that means, they are treated as independent sources of infection, with illness occurring if any one of the sources causes it.

The predicate `vulnerable` is used to model natural variation in the likelihood of infection. This could come from their age, their medical history or from behavioural patterns. It could also be used to model the effects of possibly imperfect vaccination. While a perfect vaccination can also be modelled in this way, by removing the possibility of non-vulnerable individuals to contract the disease, it is better treated in the context of acquired resistance using the methods of Section 3. We again exploit the noisy-or combination to set an explicit additional likelihood of infection of vulnerable students. In Listing 1 above, vulnerability doubles effective exposure time by adding an additional and equal chance of infection to the baseline. On the other hand, various protective measures can affect not just an individual's own likelihood of contracting the illness, but also their chance of spreading it to others. This could be modelled in a similar fashion.

The chance of infection outside of the setting can be added or left out depending on the purpose of the model. Such a choice would have no impact on the remainder of the code.

The varying time of infectiousness is modelled by setting the chance to be infectious in a week following a previous week of infection to $1/l$, where l is the expected time of infectivity in

weeks.

The data provided consists firstly of a list of weeks matching the term length (or alternatively the time period under investigation), and then the enrolment data of the courses. While ProbLog provides a library that offers a simple interface to a CSV file, we found it more efficient to load the data into a Prolog knowledge base, since reading the data from CSV seems to add significant processing overhead. A small excerpt from such a knowledge base can be seen in Listing 3.

Listing 3: Courses and weeks

```
1  participate(course1,person1).
2  participate(course1,person2).
3  participate(course2,person3).
4  participate(course2,person4).
5  participate(course3,person2).
6  participate(course3,person4).
7  participate(course3,person5).
8
9
10 week(1).
11 week(2).
12 week(3).
13 week(4).
14 week(5).
```

ProbLog then allows for various modelling and simulation tasks. One could simply query the likelihood of illness across different weeks in different populations. One could also manually intervene by adding a fact of the form `ill(1,person1).` and observe how quickly a single case of infection can spread through the population. In principle, another interesting option would be using `evidence` to condition on the actual observations of infections at a given week. However, using `evidence` results in a significant increase in computation time when sampling from the distribution. This limits its usability in larger datasets.

The effectiveness of various policies to mitigate the disease can then be determined using straightforward extensions to the code. For instance, one could test the effectiveness of teaching in person classes only in even-numbered weeks by adding an appropriate test for evenness to the clause in lines 12-13 of Listing 1. If certain classes were to be moved online, they could simply be removed from the knowledge base.

3. Incorporating resistance

To incorporate resistance into the model, we use ProbLog's ability to work with stratified negation. ProbLog can process negative conditions in the body of a clause as long as there are no cycles involving such negative conditions. On the first glance, however, there do seem to be cycles involved in a naive formulation of an SIR contingency:

Infection is prevented by resistance, but resistance also arises from a prior infection. This logic could be naively implemented as in Listing 4.

Listing 4: Generic model of resistance

```
% Two different students are classmates if they attend the same class.
classmates(STUDENTx ,STUDENTy):- participate(COURSE, STUDENTx),
participate(COURSE, STUDENTy), STUDENTx\==STUDENTy.

student(STUDENTx) :- participate(COURSE, STUDENTx).

% In every given week, there is a 1 percent chance of infection outside the setting.
0.01::ill(B,STUDENTx) :- student(STUDENTx), week(B), \+ill(A, STUDENTx),
\+resistant(STUDENTx, A), B is A+1.

% 10 percent of students are particularly vulnerable to disease.
0.1::vulnerable(STUDENTx) :- student(STUDENTx).

% The average period of infectivity is two weeks.
0.5::ill(B, STUDENTx) :- ill(A,STUDENTx), week(B), B is A+1.

% A student has a 5 percent chance of infection from a diseased classmate.

0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, A), \+ill(A, STUDENTx), week(B), B is A+1.

0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, A), \+ill(A, STUDENTx),
week(B), B is A+1.

% A student is resistant as soon as he has recovered from an illness.

resistant(STUDENTx, B):- ill(A, STUDENTx), \+ill(B, STUDENTx), week(B), B is A+1.

% Resistance lasts for the following weeks.

resistant(STUDENTx,B):-resistant(STUDENTx, A), week(B), B is A+1.
```

Indeed, the ProbLog 2 engine throws an exception, having detected negative cycles.

On closer inspection, however, the apparent negative cycle disappears when grounding out the program with respect to the time in weeks. This is because illness causes *future* resistance, while resistance *in the past* prevents future illness.

We therefore partially grounded out the program with respect to weeks using a variant of the partial reducer from [13, Section 18.1]. The resulting unfolded program is passed to the ProbLog 2 engine. This causes no further errors, and is in fact significantly faster than the model of Section 2, since ProbLog no longer has to take into account multiple infections of the same individual.

The program unfolded with respect to the knowledge base of Listing 3 can be found in Appendix A.

Incorporating resistance unlocks modelling the full SIR cycle present in many infectious diseases. The same mechanism can also be expanded to model the effects of vaccinations. Vaccines take individuals from being susceptible direct to being resistant, essentially bypassing

the “infected” phase. This is represented by the grey arrow in Figure 1. This feature also allows modellers to quickly adapt the model depending on the rate of vaccination in the local population. Indeed, a constant rate of vaccination progress could be modelled by just a single additional clause. In cases where more than one dose of vaccine is required to reach full immunity, we can model this by reducing the vulnerability of individuals with one dose compared to unvaccinated individuals, then counting fully vaccinated individuals as `resistant`. The exact value of this reduction in vulnerability can be toggled easily depending on the latest data on vaccine efficacy, and can be adapted with the publication of new data.

As every part of our program, the treatment of resistance can also be flexibly adapted. Should it turn out, for instance, that resistance is obtained temporarily rather than permanently, this can be expressed simply by annotating Line 32 of Listing 4 with a non-1 probability.

4. Incorporating distance

While the previous two models implemented a network approach, one can also use ProbLog to implement a much more local model, which can explore different parameters of disease spread as a direct function of the distance between two individuals.

A major advantage of ProbLog2 is the support of flexible probabilities, which can be used for this implementation. This means that the probabilities are not set in the code, but are calculated by arithmetic expressions at run time, making it possible to specify more complex models. The probabilities of a possible infection are determined with the help of the Euclidean distance function. We assume a two-dimensional linear world, where the distance between two points is determined by the length of the straight line connecting them.

As the dispersal of the virus through droplets in the air is not fully understood, it is unclear how infectiousness depends on distance [14]. In our framework, the program can be rapidly adjusted for different models of airborne dispersal using the ProbLog 2 built-ins.

The position of the persons is determined by coordinates. An extract of a knowledge base is shown in Listing 5.

Listing 5: knowledge base of individuals and their coordinates

```
1  point(person0, 2, 3).
2  point(person1, 3, 4).
3  point(person2, 6, 9).
4  point(person3, 10, 24).
5  point(person4, 12, 16).
6  point(person5, 7, 8).
```

The calculation of the contagion probability and the modelling of the spread of the disease can be represented very compactly in a single clause, as shown in Listing 6. Additionally, the function calculating the distance can easily be altered or replaced entirely to simulate different models of spread.

Listing 6: Infection logic for the Euclidean distance model

```
1    P :: infects(PERSONx,PERSONy) :-  point(PERSONx, X, Y),
2    point(PERSONy, A, B),  PERSONx\==PERSONy,
```

```
3       D is sqrt((A-X)^2 + (B-Y)^2),
4       P is 0.1/(D^2), D < 10.
5       ill(PERSONx):-infects(PERSONx,PERSONy), is_ill(PERSONy).
6       ill(PERSONx):-is_ill(PERSONx).
```

Modelling disease likelihood based on relative location opens the door to further applications where the locations of (or the distances between) individuals are known. One use case would be as an epidemiological component of an agent-based modelling system. Agent-based modelling, which has developed into a promising alternative to equational and network-based models of infectious disease dynamics, simulates the behaviour of individuals according to given rules. An agent-based epidemiological model can be divided into several layers, one of them being the disease model itself [15]. Since agent-based models are often used to test the consequences of epidemiological assumptions, a flexible and rapidly specifiable system would be desirable there. Another opportunity would be to use the data from mobile tracking applications, such as the Corona-Warn-App used in Germany. This currently implements a much more basic model of disease spread, which has been criticised for being over-simplistic [16].

5. Experiments

To verify the feasibility of our approach, we tested the frameworks of Sections 3 and 4 for the SIR model and distance-dependent modelling respectively. All experiments were performed on a KVM virtual machine with 35 CPUs, 240 GB RAM and 17 GB swap running Ubuntu 21.04.

Sampling from a network-based SIR model We obtained student enrolment data from the Faculty for Mathematics, Computer Science and Statistics of the LMU University of Munich. This encompassed 14828 individual enrolment events of 4594 distinct students in 104 distinct courses. We then used the 'sampling' mode on the code of Appendix A to simulate the progress of the pandemic with respect to a randomly sampled subset of students from the institute. More precisely, we randomly sampled a given number of students and then included all enrolment events involving a sampled student in the knowledge base. We used the query `ill(_,_)` and the sampling mode of ProbLog 2 to simulate the spread of illness across five weeks.

The following run times were obtained:

Number of students sampled	Run time (seconds)
50	0.7
158	4.2
331	34.7
460	94
610	213
763	290
920	504
1078	670

This demonstrates that even for realistic connection graphs taken from real-life data and with several hundred nodes, simulations can be performed in reasonable time.

Calculating the risk of infection from a distance-based model To test the feasibility of the distance-based infection model of Listing 6, we implemented it with synthetic data of 1,000 people scattered randomly across a two-dimensional grid of 100*100 units. We also added a background infection rate of 1% and initialised a known case by adding a fact `is_ill(person1)`. We then queried the likelihood of infection for one particular individual. This exact inference is performed in 0,76 seconds; without the cutoff distance of 10 encoded in line 4 of Listing 6, the computation is performed in 2,23 seconds. This makes our approach viable for the implementation of more sophisticated risk assessments than is currently provided by exposure notification applications

6. Conclusion

We implement a stochastic model for infectious disease dynamics in the probabilistic logic programming language ProbLog. Our model differs from the classic SIR model by incorporating network structures. This is an advantage in scenarios with a fixed, small or heterogeneous population. Examples of these situations include students in university or school classes, patients in a ward, or employees in offices. We show that distances between individuals can also be accounted for, opening the door to further applications.

Our implementation also showcases how core features of ProbLog can be harnessed in the context of epidemiological modelling. The relational representation streamlines the specification process and leads to concise and transparent models. Meanwhile, support for stratified negation allows modelling immunity after infection, as we can specify that the lack of prior immunity is a prerequisite for infection. Flexible probabilities allow for the likelihood of infection to be computed at run time; this is essential for incorporating distance as a parameter.

The core and advanced features of ProbLog are key to making the model flexible, transparent and compact. The implementation is accessible for audiences outside of computer science and individuals who are not familiar with programming. For example, the model can be understood by academic researchers or administrative leadership. Modelling parameters and assumptions can be read off directly, making the predictions traceable and trustworthy. Adaptability is also important for rapid epidemiological modelling. In the case of a pandemic, decisions must be made quickly, and often at a local level without access to extensive guidance. Our implementation allows modelling with possibly incomplete knowledge, but is also flexible to incorporate new data and new scientific background knowledge as it arises. Although our work was originally motivated by the SARS-CoV-2 pandemic, the techniques can be used across infectious diseases. We show that probabilistic logic programming can provide a versatile tool for simulation and experimental evaluation of settings and parameters, potentially supporting local decision-making in exceptionally challenging circumstances.

Typically, the greatest challenge in harnessing probabilistic logic programming for modelling tasks such as this one is the high computational complexity of the task. To test the feasibility of our approach, we have implemented an SIR simulation with a network taken from real-life course enrolment data as well as a distance-based exact inference task predicting the risk of infection from synthetic data. In both cases run times appropriate to the respective setting have been achieved with as many as 1,000 distinct persons involved.

Acknowledgments

We would like to thank Ella Mayer and Yagiz Teker for many constructive conversations about the topic, and François Bry for suggesting the unfolding mechanism used for partially grounding the program. We would also like to thank Gregor Kleen and Michael Stübiger for providing and anonymising the data used in our experiments, and Philipp Wendler for assisting us in obtaining the requisite data protection clearance.

References

[1] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic Prolog and its application in link discovery, in: M. M. Veloso (Ed.), IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, pp. 2462–2467.

[2] A. Kimmig, F. Costa, Link and Node Prediction in Metabolic Networks with Probabilistic Logic, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 407–426. doi:10.1007/978-3-642-31830-6_29.

[3] D. De Maeyer, J. Renkens, L. Cloots, L. De Raedt, K. Marchal, Phenetic: network-based interpretation of unstructured gene lists in E. coli, Mol. BioSyst. 9 (2013) 1594–1603. doi:10.1039/C3MB25551D.

[4] D. De Maeyer, B. Weytjens, L. De Raedt, K. Marchal, Network-Based Analysis of eQTL Data to Prioritize Driver Mutations, Genome Biology and Evolution 8 (2016) 481–494. doi:10.1093/gbe/evw010.

[5] A. Groß, B. Kracher, J. M. Kraus, S. D. Kühlwein, A. S. Pfister, S. Wiese, K. Luckert, O. Pötz, T. Joos, D. Van Daele, L. De Raedt, M. Kühl, H. A. Kestler, Representing dynamic biological networks with multi-scale probabilistic models, Communications Biology 2 (2019) 21. doi:10.1038/s42003-018-0268-3.

[6] W. O. Kermack, A. G. McKendrick, Contribution to the mathematical modelling of diseases I, Proceedings of the Royal Society 115A (1927) 700–721.

[7] A. Huppert, G. Katriel, Mathematical modelling and prediction in infectious disease epidemiology, Clinical Microbiology and Infection 19 (2013) 999–1005. doi:10.1111/1469-0691.12308.

[8] F. Wong, J. J. Collins, Evidence that coronavirus superspreading is fat-tailed, Proceedings of the National Academy of Sciences 117 (2020) 29416. doi:10.1073/pnas.2018490117.

[9] F. Brauer, An Introduction to Networks in Epidemic Modeling, Springer, Berlin, 2008, pp. 133–146. doi:10.1007/978-3-540-78911-6_4.

[10] D. Fierens, G. V. den Broeck, J. Renkens, D. S. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. D. Raedt, Inference and learning in probabilistic logic programs using weighted boolean formulas, Theory Pract. Log. Program. 15 (2015) 358–401. doi:10.1017/S1471068414000076.

[11] A. Dries, Declarative data generation with ProbLog, in: Proceedings of the Sixth International Symposium on Information and Communication Technology, SoICT 2015, Association for Computing Machinery, New York, NY, USA, 2015, p. 17–24. doi:10.1145/2833258.2833267.

[12] S. Picault, Y.-L. Huang, V. Sicard, S. Arnoux, G. Beaunée, P. Ezanno, Emulsion: Transparent

and flexible multiscale stochastic models in human, animal and plant epidemiology, PLOS Computational Biology 15 (2019) 1–13. URL: https://doi.org/10.1371/journal.pcbi.1007342. doi:10.1371/journal.pcbi.1007342.

[13] L. Sterling, E. Shapiro, The Art of Prolog - Advanced Programming Techniques, 2nd Ed, MIT Press, 1994.

[14] M. E. Rosti, S. Olivieri, M. Cavaiola, A. Seminara, A. Mazzino, Fluid dynamics of COVID-19 airborne infection suggests urgent data for a scientific design of social distancing, Scientific Reports 10 (2020) 22426. doi:10.1038/s41598-020-80078-7.

[15] E. Hunter, B. Mac Namee, J. D. Kelleher, A taxonomy for agent-based models in human infectious disease epidemiology, Journal of Artificial Societies and Social Simulation 20 (2017) 2. doi:10.18564/jasss.3414.

[16] J. Braband, H. Schäbe, Analysis of the risk model of German Corona warning app, Reliability: Theory & Applications 16 (2021).

A. Code sample

Listing 7: Unfolded program incorporating resistance

```
1
2  0.01::ill(1,STUDENTx) :- student(STUDENTx), week(1), \+ill(0, STUDENTx), \+resistant(STUDENTx, 0).
3  0.01::ill(2,STUDENTx) :- student(STUDENTx), week(2), \+ill(1, STUDENTx), \+resistant(STUDENTx, 1).
4  0.01::ill(3,STUDENTx) :- student(STUDENTx), week(3), \+ill(2, STUDENTx), \+resistant(STUDENTx, 2).
5  0.01::ill(4,STUDENTx) :- student(STUDENTx), week(4), \+ill(3, STUDENTx), \+resistant(STUDENTx, 3).
6  0.01::ill(5,STUDENTx) :- student(STUDENTx), week(5), \+ill(4, STUDENTx), \+resistant(STUDENTx, 4).
7
8  % 0.01::ill(B,STUDENTx) :- student(STUDENTx), week(B), \+ill(A, STUDENTx), \+resistant(STUDENTx, A), B is A+1.
9
10 0.1::vulnerable(STUDENTx) :- student(STUDENTx).
11
12 % The average period of infectivity is two weeks.
13 0.5::ill(1, STUDENTx):- ill(0,STUDENTx).
14 0.5::ill(2, STUDENTx):- ill(1,STUDENTx).
15 0.5::ill(3, STUDENTx):- ill(2,STUDENTx).
16 0.5::ill(4, STUDENTx):- ill(3,STUDENTx).
17 0.5::ill(5, STUDENTx):- ill(4,STUDENTx).
18
19 % 0.5::ill(B, STUDENTx) :- ill(A,STUDENTx), week(B), B is A+1.
20
21 0.05::ill(1,STUDENTx) :- ill(0,STUDENTy), classmates(STUDENTx, STUDENTy),
22 vulnerable(STUDENTx), \+resistant(STUDENTx, 0), \+ill(0, STUDENTx).
23 0.05::ill(2,STUDENTx) :- ill(1,STUDENTy), classmates(STUDENTx, STUDENTy),
24 vulnerable(STUDENTx), \+resistant(STUDENTx, 1), \+ill(1, STUDENTx).
25 0.05::ill(3,STUDENTx) :- ill(2,STUDENTy), classmates(STUDENTx, STUDENTy),
26 vulnerable(STUDENTx), \+resistant(STUDENTx, 2), \+ill(2, STUDENTx).
27 0.05::ill(4,STUDENTx) :- ill(3,STUDENTy), classmates(STUDENTx, STUDENTy),
28 vulnerable(STUDENTx), \+resistant(STUDENTx, 3), \+ill(3, STUDENTx).
```

```
0.05::ill(5,STUDENTx) :- ill(4,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 4), \+ill(4, STUDENTx).

%0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, A), \+ill(A, STUDENTx), week(B), B is A+1.

0.05::ill(1,STUDENTx) :- ill(0,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 0), \+ill(0, STUDENTx).
0.05::ill(2,STUDENTx) :- ill(1,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 1), \+ill(1, STUDENTx).
0.05::ill(3,STUDENTx) :- ill(2,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 2), \+ill(2, STUDENTx).
0.05::ill(4,STUDENTx) :- ill(3,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 3), \+ill(3, STUDENTx).
0.05::ill(5,STUDENTx) :- ill(4,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 4), \+ill(4, STUDENTx).

%0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
%\+resistant(STUDENTx, A), \+ill(A, STUDENTx),
%week(B), B is A+1.

0.05::ill(1,STUDENTx) :- ill(0,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 0), \+ill(0, STUDENTx).
0.05::ill(2,STUDENTx) :- ill(1,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 1), \+ill(1, STUDENTx).
0.05::ill(3,STUDENTx) :- ill(2,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 2), \+ill(2, STUDENTx).
0.05::ill(4,STUDENTx) :- ill(3,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 3), \+ill(3, STUDENTx).
0.05::ill(5,STUDENTx) :- ill(4,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, 4), \+ill(4, STUDENTx).

%0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
vulnerable(STUDENTx), \+resistant(STUDENTx, A), \+ill(A, STUDENTx), week(B), B is A+1.

0.05::ill(1,STUDENTx) :- ill(0,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 0), \+ill(0, STUDENTx).
0.05::ill(2,STUDENTx) :- ill(1,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 1), \+ill(1, STUDENTx).
0.05::ill(3,STUDENTx) :- ill(2,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 2), \+ill(2, STUDENTx).
0.05::ill(4,STUDENTx) :- ill(3,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 3), \+ill(3, STUDENTx).
0.05::ill(5,STUDENTx) :- ill(4,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, 4), \+ill(4, STUDENTx).

%0.05::ill(B,STUDENTx) :- ill(A,STUDENTy), classmates(STUDENTx, STUDENTy),
\+resistant(STUDENTx, A), \+ill(A, STUDENTx),
%week(B), B is A+1.
```

```
78
79
80  resistant(STUDENTx, 1):- ill(0, STUDENTx), \+ill(1, STUDENTx).
81  resistant(STUDENTx, 2):- ill(1, STUDENTx), \+ill(2, STUDENTx).
82  resistant(STUDENTx, 3):- ill(2, STUDENTx), \+ill(3, STUDENTx).
83  resistant(STUDENTx, 4):- ill(3, STUDENTx), \+ill(4, STUDENTx).
84
85  %resistant(STUDENTx, B):- ill(A, STUDENTx), \+ill(B, STUDENTx), week(B), B is A+1.
86
87  resistant(STUDENTx,1):-resistant(STUDENTx, 0).
88  resistant(STUDENTx,2):-resistant(STUDENTx, 1).
89  resistant(STUDENTx,3):-resistant(STUDENTx, 2).
90  resistant(STUDENTx,4):-resistant(STUDENTx, 3).
91  resistant(STUDENTx,5):-resistant(STUDENTx, 4).
92
93  %resistant(STUDENTx,B):-resistant(STUDENTx, A), week(B), B is A+1.
```