SimInf: An R package for Data-driven Stochastic Disease Spread Simulations

Stefan Widgren
National Veterinary Institute
and Uppsala University
Sweden

Pavol Bauer
Uppsala University
Sweden

Stefan Engblom
Uppsala University
Sweden

Abstract

Livestock movements are critical for the spread of many infectious diseases in animal populations. The use of real livestock data allows for disease spread modeling that incorporates the time-varying contact network and the population demographic. This paper introduces **SimInf**, an efficient and general framework for stochastic spatio-temporal disease-spread modeling over a temporal network of connected nodes. It integrates within-node infection dynamics as continuous-time Markov chains and livestock data as scheduled events. The core simulation solver is implemented in C and uses OpenMP to divide work over multiple processors. We provide a technical description of the framework, how to use a predefined model in **SimInf**, demonstrate a case study, and finally show how to extend the framework with a user defined model.

Keywords: computational epidemiology, discrete-event simulation, multicore implementation, stochastic modeling.

1. Introduction

Livestock movements are an important transmission route for many infectious diseases and can transfer infectious individuals between holdings over large distances (Danon, Ford, House, Jewell, Keeling, Roberts, Ross, and Vernon 2011). The livestock movements between holdings can be represented as a temporal network with nodes coupled by directed edges and the time when each edge was active (Holme and Saramäki 2012). The temporal network of livestock movements forms a complex dynamic system in the topology, connectivity and intensity (Bajardi, Barrat, Natale, Savini, and Colizza 2011; Dutta, Ezanno, and Vergu 2014), and these properties affect how various diseases spread in a network (Shirley and Rushton 2005; Büttner, Krieter, Traulsen, and Traulsen 2016). European Union legislation requires member states to keep a registry of all bovine animals in national databases (Anonymous 2000, 2004), which makes it possible to use real data for the time-varying contact network and the population demographics for large scale disease spread simulations, to better understand transmission and explore control strategies. However, incorporating large amount of network data in simulations is computationally challenging and require efficient algorithms. Furthermore, specifying realistic models for disease transmission is a complicated process since various diseases have fundamentally different transmission routes. Compare, for example, fecal-oral pathogens with indirect transmission routes via the environment with infections that are transferred by direct contact between individuals (Brooks-Pollock, de Jong, Keeling, Klinkenberg, and Wood 2015).

In this work, we present an efficient and flexible framework for data-driven spatio-temporal disease spread modeling, designed to efficiently incorporate large volumes of population demographics- and temporal network data. This allows the user to asses the spread of, for example, endemic diseases in the cattle population, in national scale simulations (Bauer, Engblom, and Widgren 2016). We have developed the R (R Core Team 2017) package **SimInf**, a discrete-event simulator that divides work among multiple processors available in standard computers. The model integrates infection dynamics as continuous-time Markov chains and available data such as animal movements, births, slaughter or aging are incorporated as scheduled events. One of our design goal was to make **SimInf** completely extendable and allow for growth in available models and specialized solvers through contributions from the community. Using compiled C code, rather than interpreted R code, for the transition-rate functions ensures maximum efficiency when simulating the model. To simplify the process of generating the required C code for models, the **SimInf** package include functionality to automatically create C code from a model specification.

There exists several related R packages for epidemiological modeling on the Comprehensive R Archive Network (CRAN) that we would like to mention here. The package **amei** (Merl, Johnson, Gramacy, and Mangel 2010) is designed for parameter estimation of epidemic models and finding optimal intervention strategies, for example, vaccination, to control disease spread. Another package is **surveillance**, a framework for monitoring, modeling, and regression analysis of infectious diseases, (Meyer, Held, and Höhle 2017). **EpiModel** (Samuel M. Jenness, Goodreau, and Morris 2017) is an R package that, similar to **SimInf**, includes a framework for modeling spread of diseases on networks. However, they differ in how the data is generated for the modeling: the network data in **EpiModel** is simulated using **statnet** (Handcock, Hunter, Butts, Goodreau, and Morris 2008), while **SimInf** was designed to incorporate available network data. The are also similarities between the R package **GillespieSSA** (Pineda-Krch 2008) and **SimInf**, in that both uses the Gillespie stochastic simulation algorithm (SSA) (Gillespie 1977). However, for computational efficiency, the algorithm is implemented using C code in **SimInf**.

The paper is organized as follows. In §2 we summarize the mathematical foundation for our framework. Section §3 gives a technical description of the simulation framework. In §4 we illustrate the use of the package by some worked examples. Finally, in §5 we demonstrate how to extend **SimInf** with user-defined models.

2. Epidemiological modeling

In the following section we give a brief overview of the epidemiological modeling framework employed in **SimInf**. The overall approach consists of continuous-time Markov chains as a general model of the dynamics of the epidemiological state. Importantly, we also allow for a coupling with concentration variables obeying ordinary differential equations (ODEs), as well as to externally defined events. We draw much of the material here from (Bauer *et al.* 2016; Engblom and Widgren 2017).

In §2.1–2.2 below we distinguish between the *local* dynamics that describes the evolution of the epidemiological state at a single node, and the *global* dynamics, which describes the system at the network level. The overall numerical approach underlying **SimInf** is described in §2.3.

2.1. Local dynamics

We describe the state of a single node with a *state vector* $X(t) \in \mathbf{Z}_+^{N_\text{comp}}$, which counts the number of individuals at each of N_comp compartments at time t. The transitions between these compartments are stochastic and are described by the transition matrix $\mathbb{S} \in \mathbf{Z}^{N_\text{comp} \times N_\text{trans}}$ and the transition intensities $R : \mathbf{Z}_+^{N_\text{comp}} \to \mathbf{R}_+^{N_\text{trans}}$, assuming N_trans different transitions. We then form a *random counting measure* $\mu_k(dt) = \mu(R_k(X(t-)); dt)$ that is associated with a Poisson process for the kth intensity $R_k(X(t-)$, which in turn depends on the state prior to any transition at time t, that is, $X(t-)$.

The local dynamics can then compactly be described by a pure jump stochastic differential equation (SDE),

$$dX(t) = \mathbb{S}\boldsymbol{\mu}(dt), \tag{1}$$

where $\boldsymbol{\mu}(dt)$ is a vector measure built up from the scalar counting measures $\boldsymbol{\mu}(dt) = [\mu_1(dt), \ldots, \mu_{N_\text{trans}}(dt)]^\top$. If at time t, transition k occurs, then the state vector is updated according to

$$X(t) = X(t-) + \mathbb{S}_k, \tag{2}$$

with $\mathbb{S}_k$ the kth column of $\mathbb{S}$. In (1) the N_trans different epidemiological state transitions are competing in the sense of independent Poisson processes. The 'winning' process decides what event happens and changes the state according to (2). The simulation then proceeds under the Markov assumption where previous events are remembered via the state variable X only.

To make this abstract notation a bit more concrete we consider a traditional example as follows. In an SIS-model the transitions between a susceptible and an infected compartment can be written as

$$\left.\begin{array}{rl} S + I & \xrightarrow{\beta} 2I \\ I & \xrightarrow{\gamma} S \end{array}\right\}. \tag{3}$$

With a state vector consisting of two compartments $X = [\#S, \#I]$, i.e., the number of susceptible and infected individuals, respectively, we can then write the transition matrix and intensity vector as

$$\mathbb{S} = \left[\begin{array}{rr} -1 & 1 \\ 1 & -1 \end{array}\right], \tag{4}$$

$$R(x) = [\beta x_1 x_2, \gamma x_2]^\top. \tag{5}$$

To connect this with traditional ODE-based models, note that, replacing the random measure in (1) with its mean drift, we arrive at

$$\frac{dx(t)}{dt} = \mathbb{S}R(x), \tag{6}$$

where now the state variable $x \in \mathbf{R}^{N_\text{comp}}$. The differences between (1) and (6) are that the randomness and discreteness of the state variable are not present in the latter formulation. If these features are thought to be important, then (1) is an accurate stochastic alternative to (6), relying only on the Markovian "memoryless" assumption.

There are, however, situations where we would like to mix the discrete stochastic model with a concentration-type ODE model. In a multi-scale description there are typically variables for which a continuous description is more natural: a typical example is the concentration of bacteria in an infectious environment for which individual counting would clearly not be feasible.

Assuming an additional concentration state vector $Y \in \mathbf{R}^{N_{\text{conc}}}$ a general model which augments (1) is

$$\left.\begin{array}{rcl} dX(t) & = & \mathbb{S}\boldsymbol{\mu}(dt) \\ Y'(t) & = & f(X(t-), Y(t)) \end{array}\right\}, \tag{7}$$

where now the random measure depends also on the concentration variable,

$$\boldsymbol{\mu}(dt) = \boldsymbol{\mu}(R(X(t-), Y(t)), dt). \tag{8}$$

The overall combined state vector is then $[X;\ Y] \in [\mathbf{Z}^{N_{\text{comp}}};\ \mathbf{R}^{N_{\text{conc}}}]$.

2.2. Global dynamics

To extend the local dynamics to a network model consisting of N_{nodes} nodes we first define the overall state matrices $\mathbb{X} \in \mathbf{Z}_{+}^{N_{\text{comp}} \times N_{\text{nodes}}}$ and $\mathbb{Y} \in \mathbf{R}^{N_{\text{conc}} \times N_{\text{nodes}}}$ and then extend (7) to

$$d\mathbb{X}^{(i)}(t) = \mathbb{S}\boldsymbol{\mu}^{(i)}(dt), \tag{9}$$

$$\frac{d\mathbb{Y}^{(i)}(t)}{dt} = f(\mathbb{X}^{(i)}, \mathbb{Y}^{(i)}), \tag{10}$$

where $i \in \{1, ..., N_{\text{nodes}}\}$ is the node index.

We then consider the N_{nodes} nodes being the vertices of an undirected graph $\mathcal{G}$ with interactions defined in terms of the counting measures $\boldsymbol{\nu}^{(i,j)}$ and $\boldsymbol{\nu}^{(j,i)}$. Here $\boldsymbol{\nu}^{(i,j)}$ represents the state changes due to an inflow of individuals from node i to node j, and $\boldsymbol{\nu}^{(j,i)}$ represents an inflow of individuals from node j to node i, assuming node j being in the connected component $C(i)$ of node i, and vice versa.

The network dynamics is then written as

$$d\mathbb{X}_t^{(i)} = -\sum_{j \in C(i)} \mathbb{C}\boldsymbol{\nu}^{(i,j)}(dt) + \sum_{j;\, i \in C(j)} \mathbb{C}\boldsymbol{\nu}^{(j,i)}(dt), \tag{11}$$

$$\frac{d\mathbb{Y}^{(i)}(t)}{dt} = -\sum_{j \in C(i)} g(\mathbb{X}^{(i)}, \mathbb{Y}^{(i)}) + \sum_{j;\, i \in C(j)} g(\mathbb{X}^{(j)}, \mathbb{Y}^{(j)}). \tag{12}$$

In (12), g is similarly the "flow" of the concentration variable $\mathbb{Y}$ between the nodes in the network. For example, this could be the natural modeling target for concentration variables $\mathbb{Y}$ which are transported via surface water or air.

Combining this with (9)–(10) we obtain the overall dynamics

$$d\mathbb{X}^{(i)}(t) = \mathbb{S}\boldsymbol{\mu}^{(i)}(dt) - \sum_{j \in C(i)} \mathbb{C}\boldsymbol{\nu}^{(i,j)}(dt) + \sum_{j;\, i \in C(j)} \mathbb{C}\boldsymbol{\nu}^{(j,i)}(dt), \tag{13}$$

$$\frac{d\mathbb{Y}^{(i)}(t)}{dt} = f(\mathbb{X}^{(i)}, \mathbb{Y}^{(i)}) - \sum_{j \in C(i)} g(\mathbb{X}^{(i)}, \mathbb{Y}^{(i)}) + \sum_{j;\, i \in C(j)} g(\mathbb{X}^{(j)}, \mathbb{Y}^{(j)}). \tag{14}$$

Note that $\boldsymbol{\nu}^{(i,j)}$ and $\boldsymbol{\nu}^{(j,i)}$ may be equivalently employed for externally scheduled events given by data using an equivalent construction in terms of Dirac measures. This is the case, for example, when intra-nodal transport data of individuals are available.

2.3. Numerical method

In **SimInf**, we solve (13)–(14) by splitting the local update scheme (9)–(10) from the global update scheme (11)–(12). We discretize time as $0 = t_0 < t_1 < t_2 < \cdots$, which is partially required as external data has to be incorporated at some finitely resolved time stamps. The numerical method of **SimInf** can then be written per node i as

$$\tilde{\mathbb{X}}^{(i)}_{n+1} = \mathbb{X}^{(i)}_n + \int_{t_n}^{t_{n+1}} \mathbb{S}\boldsymbol{\mu}^{(i)}(ds), \tag{15}$$

$$\mathbb{X}^{(i)}_{n+1} = \tilde{\mathbb{X}}^{(i)}_{n+1} - \int_{t_n}^{t_{n+1}} \sum_{j \in C(i)} \mathbb{C}\boldsymbol{\nu}^{(i,j)}(ds) + \int_{t_n}^{t_{n+1}} \sum_{j;\, i \in C(j)} \mathbb{C}\boldsymbol{\nu}^{(j,i)}(ds), \tag{16}$$

$$\begin{aligned} \mathbb{Y}^{(i)}_{n+1} = \mathbb{Y}^{(i)}_n &+ f(\tilde{\mathbb{X}}^{(i)}_{n+1}, \mathbb{Y}^{(i)}_n)\,\Delta t_n \\ &- \sum_{j \in C(i)} g(\tilde{\mathbb{X}}^{(i)}_{n+1}, \mathbb{Y}^{(i)}_n)\Delta t_n + \sum_{j;\, i \in C(j)} g(\tilde{\mathbb{X}}^{(j)}_{n+1}, \mathbb{Y}^{(j)}_n)\Delta t_n. \end{aligned} \tag{17}$$

In this scheme, (15) forms the local stochastic step, that is in practice simulated by the Gillespie method (Gillespie 1977). Eq. (16) is the data step, where externally scheduled events are incorporated. Note that the stochastic step evolves at continuous time increments in the interval $[t_n, t_{n+1}]$, and the data step operates only on the final state $\tilde{\mathbb{X}}$ at t_{n+1}. The final step (17) is just the Euler forward method in time with time-step $\Delta t_n = t_{n+1} - t_n$ for the concentration variable $\mathbb{Y}$.

3. Technical description of the simulation framework

The overall design of **SimInf** was inspired and partly adapted from the Unstructured Mesh Reaction-Diffusion Master Equation (URDME) framework (Engblom, Ferm, Hellander, and Lötstedt 2009; Drawert, Engblom, and Hellander 2012). **SimInf** is a general software framework for data-driven modeling and simulation of stochastic disease-spread in a temporal network of connected nodes. In particular, the transition rates and the transition topology are specified using a compiled language C and R matrices, respectively. The overall modular design makes extensions easy to handle and since the simulation engine is written in a compiled language, the efficiency of the simulator is very high.

The **SimInf** package is available via CRAN at `https://CRAN.R-project.org/package=SimInf` and is loaded in R with the following command

```
R> library("SimInf")
```

3.1. Overview

The **SimInf** R package uses object oriented programming with S4 classes (Chambers 2008) to define objects with logical layers connected by well-defined interfaces for different modeling scenarios. The two S4 classes `SimInf_model` and `SimInf_events` are central and provide the basis for the flexible framework to perform efficient simulations in compiled C code of the predefined models in **SimInf** or user-defined models. The `SimInf_model` class contains slots (Table 1) to data structures that define the disease-spread model and one slot `events` associated to a `SimInf_events` object, which contains slots (Table 2) to incorporate and process scheduled events. Furthermore, the `SimInf_model` object also contains the output trajectory after running the core simulation solver on the disease-spread model.

All disease-spread models in **SimInf** contains the S4 class `SimInf_model`. Moreover, the predefined disease-spread models in **SimInf** have a generating function (Chambers 2008), with the same name as the model, which check parameters and initialize the necessary data structures for that specific model. After the model is initialized, the simulation is started with a call to the `run` method. To facilitate post-processing and visualization of output data, the predefined models in **SimInf** provide several utility functions, for example, `susceptible`, `infected`, `prevalence` and `plot`.

3.2. Specification of an epidemiological model

The within-node disease spread model in **SimInf** is specified as a compartment model with the individuals divided into compartments defined by discrete disease statuses. The model is defined by the slots in the S4 class `SimInf_model` (Table 1). The compartments contains the number of individuals in each of the N_{comp} disease states in every N_{nodes} nodes.

Eq. (17), the stochastic step, contains N_{trans} state transitions and is processed using the two slots, `S` and `G`. The `S` slot is the state-change matrix ($N_{\text{comp}} \times N_{\text{trans}}$) that determines how to change the number of individuals in the compartments of a node when the j^{th} state transition occurs, where $1 \leq j \leq N_{\text{trans}}$. Each row corresponds to one compartment and each column to a state transition. Let `u[, i]` be the number of individuals in each compartment in node i at time t_i. To move simulation time forward in node i to $t_i = t_i + \tau_i$, the vector `u[, i]` is updated according to the j^{th} transition by adding the state-change vector `S[, j]` to `u[, i]`. After updating `u[, i]`, the transition rates must be recalculated to obtain the time to the next event. However, a state transition might not need all transition rates to be recalculated. The dependency graph `G` is a matrix ($N_{\text{trans}} \times N_{\text{trans}}$) that determines which transition rates that need to be recalculated. A non-zero entry in element `G[k, j]` indicates that transition rate `k` needs to be recalculated if the j^{th} state transition occurs, where $1 \leq$ `k` $\leq N_{\text{trans}}$. Furthermore, the final step (17) is incorporated using a model specific post time step callback to allow update of the concentration variable $\mathbb{Y}$.

Model-specific data that is passed to the transition-rate functions and the post time-step function are stored in the two slots: `ldata` and `gdata` in the `SimInf_model` object. The `ldata` matrix holds local data for each node where `ldata[, i]` is the data vector for node i. Data that is global, i.e., shared between nodes, is stored in the `gdata` vector.

Slot	Description
`S`	Each column corresponds to a state transition, and execution of state transition j amounts to adding the `S[, j]` column to the state vector `u[, i]` of node i where the transition occurred. Sparse matrix ($N_{\text{comp}} \times N_{\text{trans}}$) of object class `dgCMatrix`.
`G`	Dependency graph that indicates the transition rates that need to be updated after a given state transition has occurred. A non-zero entry in element `G[i, j]` indicates that transition rate i needs to be recalculated if the state transition j occurs. Sparse matrix ($N_{\text{trans}} \times N_{\text{trans}}$) of object class `dgCMatrix`.
`tspan`	A vector of increasing time points where the state of each node is to be returned.
`U`	The result matrix with the number of individuals in each compartment in every node. `U[, j]` contains the number of individuals in each compartment at `tspan[j]`. `U[1:`N_{comp}`, j]` contains the number of individuals in each compartment in node 1 at `tspan[j]`. `U[(`N_{comp}` + 1):(2 * `N_{comp}`), j]` contains the number of individuals in each compartment in node 2 at `tspan[j]` etc. Integer matrix ($N_{\text{nodes}} N_{\text{comp}} \times \text{length(tspan)}$).
`U_sparse`	It is possible to run the simulator and write the number of individuals in each compartment to the `U_sparse` sparse matrix (`dgCMatrix`), which can save a lot of memory if the model contains many nodes and time-points, but where only a few of the data points are of interest. If `U_sparse` is non-empty when `run` is called, the non-zero entries in `U_sparse` indicates where the number of individuals should be written to `U_sparse`. The layout of the data in `U_sparse` is identical to `U`. Please note that the data in `U_sparse` is numeric and that the data in `U` is integer.
`u0`	The initial number of individuals in each compartment in every node. Integer matrix ($N_{\text{comp}} \times N_{\text{nodes}}$).
`V`	The result matrix for the real-valued continuous state. `V[, j]` contains the real-valued state of the system at `tspan[j]`. Numeric matrix ($N_{\text{nodes}} N_{ld} \times \text{length(tspan)}$).
`V_sparse`	It is possible to run the simulator and write the real-valued continuous state to the `V_sparse` sparse matrix (`dgCMatrix`), which can save a lot of memory if the model contains many nodes and time-points, but where only a few of the data points are of interest. If `V_sparse` is non-empty when `run` is called, the non-zero entries in `V_sparse` indicates where the real-valued continuous state should be written to `V_sparse`. The layout of the data in `V_sparse` is identical to `U`.
`v0`	The initial value for the real-valued continuous state. Numeric matrix ($N_{ld} \times N_{\text{nodes}}$).
`ldata`	A numeric matrix with local data specific to each node. The column `ldata[, j]` contains the local data vector for node j. The local data vector is passed as an argument to the transition rate functions and the post time step function.
`gdata`	A numeric vector with global data that is common to all nodes. The global data vector is passed as an argument to the transition rate functions and the post time step function.
`events`	Scheduled events to modify the discrete state of individuals in a node at a pre-defined time t. S4 class `SimInf_events`, see §3.3 and Table 2.
`C_code`	Character vector with optional model `C` code, see §5. If non-empty, the `C` code is written to a temporary file when the `run` method is called. The temporary file is compiled and the resulting DLL is dynamically loaded. The DLL is unloaded and the temporary files are removed after running the model.

Table 1: Description of the slots in the S4 class `SimInf_model` that defines the epidemiological model. N_{trans} is the number of state transitions in the model. N_{comp} is the number of compartments in the model. N_{nodes} is the number of nodes in the model. N_{ld} is the number of local data specific to each node and equals `dim(ldata)[1]`.

Slot	Description
`E`	Each row corresponds to one compartment in the model. The non-zero entries in a column indicates the compartments to include in an event. Sparse matrix of object class `dgCMatrix`.
`N`	Each row represents one compartment in the model and the values determine how to move sampled individuals in *internal transfer* and *external transfer* events. Integer matrix.
`event`	Type of event: 0) *exit*, 1) *enter*, 2) *internal transfer*, and 3) *external transfer*. Other values are reserved for future event types and not supported by the current default core solver. Integer vector.
`time`	Time of the event. Integer vector.
`node`	The node that the event operates on. Also the source node for an *external transfer* event. Integer vector. $1 \leq$ `node[i]` $\leq N_{\text{nodes}}$.
`dest`	The destination node for an *external transfer* event, equals 0 for the other event types. Integer vector.
`n`	The number of individuals affected by the event. Integer vector. `n[i]` ≥ 0.
`proportion`	If `n[i]` equals zero, the number of individuals affected by `event[i]` is calculated by summing the number of individuals in the compartments determined by `select[i]` and multiplying with `proportion[i]`. Numeric vector. $0 \leq$ `proportion[i]` ≤ 1.
`select`	Column j in the event matrix E that determines the compartments that the event operates on. Integer vector.
`shift`	Column k in the shift matrix N that determines how individuals in *internal transfer* and *external transfer* events are shifted to enter another compartment. Integer vector.

Table 2: Description of the slots in the S4 class `SimInf_events` that holds data to process events. Each index, `i`, of the vectors represent one event. N_{nodes} is the number of nodes in the model.

The `events` slot in the `SimInf_model` holds data to process the scheduled events, further described in §3.3.

During simulation of one trajectory, the state of the system is written to the two matrices `U` and `V`. This happens at each occasion the simulation time passes a time point in `tspan`, a vector of increasing time points. The first and last element in `tspan` determines the start- and end-point of the simulation. The column `U[, m]` contains the number of individuals in each compartment in every node at `tspan[m]`, where $1 \leq$ `m` $\leq$ `length(tspan)`. The first N_{comp} rows in `U` contains the compartments of the first node. The next N_{comp} rows contains the compartments of the second node etc. The `V` matrix contains output from continuous state variables. The column `V[, m]` contains the values at `tspan[m]`. The rows are grouped per node and the number of rows per node is determined by the number of continuous state variables in that specific model. It is also possible to configure the simulator to write the state of the system to the sparse matrices `U_sparse` and `V_sparse`, which can save a lot of memory if the model contains many nodes and time-points, but where only a few of the data points are of interest. In order to use this feature, call the `U` and `V` methods (before running a trajectory) with a sparse matrix with non-zero entries where the simulator should write the state of the system. The initial state in each node is specified by the two matrices `u0` and `v0` where `u0[, i]` is the initial number of individuals in each compartment at node i and `v0[, i]` is the initial continuous state in node i.

Argument	Description
`v_new`	The continuous state vector in the node after the post time step. Exists only in `PTSFun`.
`u`	The compartment state vector in the node.
`v`	The current continuous state vector in the node.
`ldata`	The local data vector for the node.
`gdata`	The global data vector.
`node`	The node index. Note the node index is zero-based, i.e., the first node is 0.
`t`	Current time in the simulation.

Table 3: Description of the arguments to the transition rate functions (`TRFun`) and the post time step function (`PTSFun`).

3.3. Specification of scheduled events

The scheduled events are used to modify the discrete state of individuals in a node at a pre-defined time t. There are four different types of events; *enter*, *internal transfer*, *external transfer* and *exit*. The *enter* event adds individuals to a node, for example, due to births. The *internal transfer* event moves individuals between compartments within one node, for example, ageing of individuals in a model with several age compartments or vaccination to move individuals to a vaccinated compartment. The *external transfer* event moves individuals from compartments in one node to compartments in a destination node. Finally, the *exit* event removes individuals from a node, for example, due to slaughter. The event types are classified into those that operate on the compartments of a single node $E_1 = \{$*enter*, *internal transfer*, *exit*$\}$ and those that operate on the compartments of two nodes $E_2 = \{$*external transfer*$\}$. The parallel algorithm processes these two classes of events differently, see Algorithm 1 in the appendix with pseudo-code for the core simulation solver. The scheduled events are processed when simulation time reaches the time for any of the events. Events that are scheduled at the same time are processed in the following order: *exit*, *enter*, *internal transfer* and *external transfer*.

The S4 class `SimInf_events` contains slots with data structures to process events (Table 2). The slots `event`, `time`, `node`, `dest`, `n`, `proportion`, `select` and `shift`, are vectors of equal length. These vectors hold data to process one event: `e`, where $1 <$ `e` $\leq$ `length(event)`. The event type and the time of the event are determined by `event[e]` and `time[e]`, respectively. The compartments that `event[e]` operates on, are specified by `select[e]` together with the slot `E`. Each row $\{1, 2, ..., N_{\text{comp}}\}$ in the sparse matrix `E`, represents one compartment in the model. Let `s <- select[e]`, then each non-zero entry in the column `E[, s]` includes that compartment in the `event[e]` operation. The definitions of all of these operations are a bit involved and to quickly get an overview, schematic pictures illustrating all of them have been prepared, we refer to Figures 12,13,14,15,16 in the appendix.

Processing of an enter event

The *enter* event adds `n[e]` individuals to one compartment at `node[e]`, where the compartment is specified by a non-zero entry in the row for the compartment in column `E[, s]`. Please note that, if the column `E[, s]` contains several non-zero entries, the individuals are added to the compartment represented by the first non-zero row in column `E[, s]`. The values of `dest[e]`, `proportion[e]` and `shift[e]`, described below, are not used when processing an *enter* event. See Figure 13 in the appendix for an illustration of a scheduled *enter* event.

Processing of an internal transfer event

The *internal transfer* event moves `n[e]` individuals into new compartments within `node[e]`. However, if `n[e]` equals zero, the number of individuals to move is calculated by multiplying the `proportion[e]` with the total number of individuals in the compartments represented by the non-zero entries in column `E[, s]`. The individuals are then proportionally sampled and removed from the compartments specified by `E[, s]`. The next step is to move the sampled individuals to their new compartment using the matrix `N` and `shift[e]`, where `shift[e]` specifies which column in `N` to use. Each row $\{1, 2, ..., N_{\text{comp}}\}$ in `N`, represents one compartment in the model and the values determine how to move sampled individuals before adding them to `node[e]` again. Let `q <- shift[e]`, then each non-zero entry in `N[, q]` defines the number of rows to move sampled individuals from that compartment i.e., sampled individuals from compartment `p` are moved to compartment `N[p, q] + p`, where $1 \leq$ `N[p, q] + p` $\leq N_{\text{comp}}$. The value of `dest[e]`, described below, is not used when processing an *internal transfer* event. See Figure 15 in the appendix for an illustration of a scheduled *internal transfer* event.

Processing of an external transfer event

The *external transfer* event moves individuals from `node[e]` to `dest[e]`. The sampling of individuals from `node[e]` is performed in the same way as for an *internal transfer* event. The compartments at `node[e]` are updated by subtracting the sampled individuals while adding them to the compartments at `dest[e]`. The sampled individuals are added to the same compartments in `dest[e]` as in `node[e]`, unless `shift[e]` > 0. In that case, the sampled individuals change compartments according to `N` as described in processing an *internal transfer* event before adding them to `dest[e]`. See Figures 14 and 16 in the appendix for illustrations of scheduled *external transfer* events.

Processing of an exit event

The *exit* event removes individuals from `node[e]`. The sampling of individuals from `node[e]` is performed in the same way as for an *internal transfer* event. The compartments at `node[e]` are updated by subtracting the sampled individuals. The values of `dest[e]` and `shift[e]` are not used when processing an *exit* event. See Figure 12 in the appendix for an illustration of a scheduled *exit* event.

3.4. Core simulation solver

The **SimInf** package uses the ability to interface compiled code from R Chambers (2008). The solver is implemented in the compiled language C (Kernighan and Ritchie 1988) and is called from R using the `.Call()` interface (Chambers 2008). Using compiled rather than interpreted code ensures high performance when running the model. To improve performance further, the solver uses OpenMP (OpenMP Architecture Review Board 2008) to divide work over multiple processors and perform computations in parallel. The solver uses the GNU Scientific Library (GSL) (Galassi, Davies, Theiler, Gough, Jungman, Alken, Booth, and Rossi 2009) to generate random numbers for the simulation.

Function pointers

The flexibility of the solver is partly achieved by using function pointers (Kernighan and Ritchie 1988). A function pointer is a variable that stores the address of a function that can be used to invoke the function. This provides a simple way to incorporate model specific functionality into the solver. A model must define one transition rate function for each state transition in the model. These functions are called by the solver to calculate the transition rate for each state transition in each node. The output from the transition rate function depends only on the state of the system at the current time. However, the output is unique to a model and data are for that reason passed on to the function for the calculation. Furthermore, a model must define the post time step function. This function is called once for each node each time the simulation of the continuous-time Markov chain reaches the next day (or, more generally, the next unit of time) and after the E_1 and E_2 events have been processed. The main purpose of the post time step function is to allow for a model to update continuous state variables in each node.

The transition rate function is defined by the data type `TRFun` and the post time step function by the data type `PTSFun`. These data types are defined in the header file `'src/SimInf.h'` and shown below. The arguments `v_new`, `u`, `v`, `ldata`, `gdata`, `node`, and `t` of the functions are described in Table 3.

```
typedef double (*TRFun)(const int *u, const double *v, const double *ldata,
                        const double *gdata, double t);

typedef int (*PTSFun)(double *v_new, const int *u, const double *v,
                      const double *ldata, const double *gdata,
                      int node, double t);
```

Overview of the solver

Here follows an overview of the steps in the solver to run one trajectory, see Algorithm 1 for pseudo-code and `'src/core/SimInf_solver.c'` for the source code. The simulation starts with a call to the `run` method with the model as the first argument and optionally the number of threads to use and a seed for the random number generator. This method will first call the validity method on the model to perform error-checking and then call a model specific C function to initialize the function pointers to the transition rate functions and the post time step function of the model. Subsequently, the simulation solver is called to run one trajectory using the model specific data, the transition rate functions, and the post time step function. If the `C_code` slot is non-empty, the C code is written to a temporary file when the `run` method is called. The temporary file is compiled using `'R CMD SHLIB'` and the resulting DLL is dynamically loaded. The DLL is unloaded and the temporary files are removed after running the model. This is further described in §5.

The solver simulates the trajectory in parallel if OpenMP is available. The default is to use all available threads. However, the user can specify the number of threads to use. The solver divides data for the N_{nodes} nodes and the E_1 events over the number threads. All E_1 events that affect node i is processed in the same thread as node i is simulated in. The E_2 events are processed in the main thread.

The solver runs the continuous-time Markov chain for each node i. For every time step τ_i, the count in the compartments at node i is updated according to the state transition that occurred (§3.2). The time to the next event is computed, after recalculating affected transition

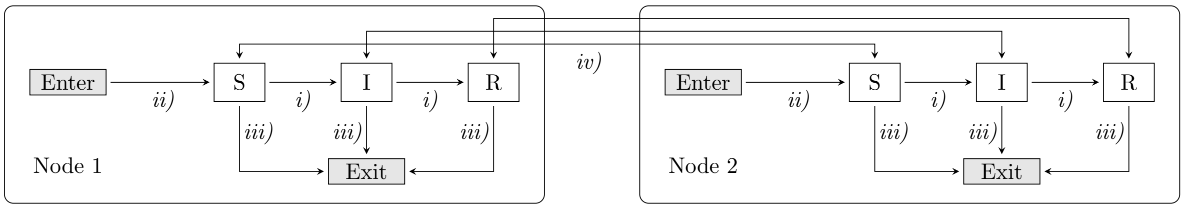


Figure 1: Schematic representation of the `SIR` compartment model in two nodes. The `SIR` model is defined by the three disease states susceptible (S), infected (I), and recovered (R). *i)* State transitions between the S, I, and R compartments are modeled as a continuous-time discrete-state Markov process. The other state transitions are due to scheduled events: *ii) enter* events, *iii) exit* events, and *iv) external transfer* events.

rate functions (§3.2). When simulated time reaches the next day in node i the E_1 events are processed for that node (§3.3). The E_2 events are processed when all nodes reaches the next day (§3.3). Thereafter, the post time step function is called to allow the model to incorporate model specific actions. When simulated time passes the next time in `tspan`, the count of the compartments and the continuous state variables are written to `U` and `V`.

4. Model construction and data analysis: basic examples

4.1. A first example: The `SIR` model

This section illustrates the specification of the predefined `SIR` model, which contains the three compartments susceptible (`S`), infected (`I`) and recovered (`R`), where $\{S, I, R\}$ represents the number of individuals in each of the three compartments, respectively. The transmission route of infection to susceptible individuals is through direct contact between susceptible and infected individuals. The `SIR` model has two state transitions in each node i,

$$\begin{array}{ccc} S_i & \xrightarrow{\beta S_i I_i/(S_i+I_i+R_i)} & I_i, \\ I_i & \xrightarrow{\gamma I_i} & R_i, \end{array} \tag{18}$$

where β is the transmission rate and γ is the recovery rate. The state change matrix `S` and the dependency graph `G` for the `SIR` model are defined as follows

$$\mathbf{S} = \begin{array}{c} \\ S \\ I \\ R \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} \end{array} \qquad \mathbf{G} = \begin{array}{c} \\ S \to I \\ I \to R \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array}$$

The first step in creating an `SIR` model object is to define `u0`, a `data.frame` with the initial condition, i.e., the number individuals in each compartment when the simulation starts. We will start in a node with 100 individuals of which 1 is infected.

```
R> u0 <- data.frame(S = 99, I = 1, R = 0)
```

Next, we define the time period over which we want to simulate the disease spread. This is a vector of integers in units of time or a vector of dates. You specify those time points in the vector that you wish the model to return results for. The model itself does not run in discrete time steps, but continuous time, so this vector affects only the start and end points of the simulation and the results that you receive from the model, not the internal calculations of disease transitions through time. In this example we simulate for 6 months returning results every 7^{th} day

```
R> tspan <- seq(1, 6 * 30, by = 7)
```

Before we can simulate from the model, we need to specify the transmission rate, β, and the recovery rate, γ.

```
R> model <- SIR(u0 = u0, tspan = tspan, beta = 0.16, gamma = 0.077)
```

We are now ready to run the model with these parameters and simulate data. We use the `threads` and `seed` arguments for reproducibility.

```
R> result <- run(model, threads = 1, seed = 22)
```

The `result` object contains the number of individuals in each compartment at the time points specified in `tspan` and can be extracted using the `U` method. The row names indicate the compartments in the model and the colnames are the time points in `tspan`. Please note that the `U` matrix is truncated here for readability.

```
R> U(result)[, 1:10]

   1  8 15 22 29 36 43 50 57 64
S 99 97 90 82 73 67 51 45 37 32
I  1  3  9  6 11 13 22 20 15 14
R  0  0  1 12 16 20 27 35 48 54
```

The package **SimInf** has a `plot` method for objects of class `SimInf_model`, such as the object `result`, which we can use to display the outcome of the disease simulation in the single node (Figure 2).

```
R> plot(result)
```

In order to run a simulation with multiple nodes we need to just add those nodes to the initial `u0` `data.frame` that we generate to start the model. If you were doing your own modeling of many nodes, you would have a `data.frame` of your node population that you read from, for example, a spreadsheet. For now we will just add another nodes to the `data.frame` here in R.

```
R> u0 <- data.frame(S = c(99, 90), I = c( 1,  2), R = c( 0,  0))
R> model <- SIR(u0 = u0, tspan = tspan, beta = 0.16, gamma = 0.077)
R> result <- run(model, threads = 1, seed = 22)
```

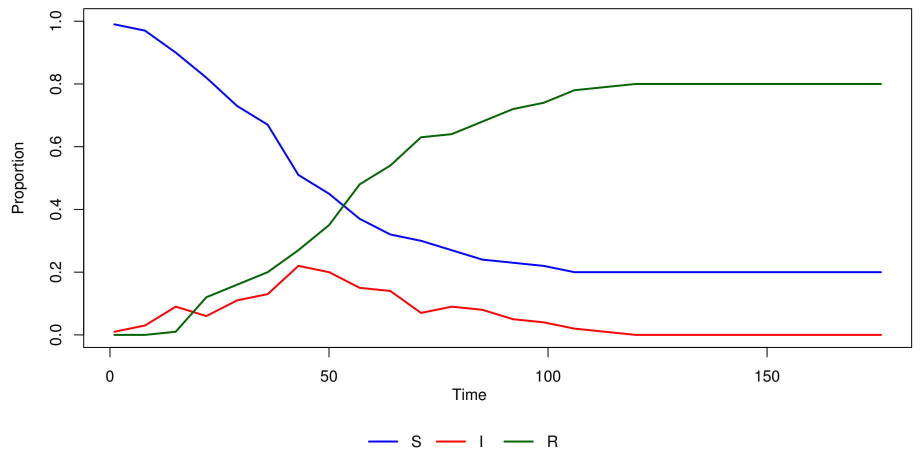


Figure 2: Example of simulated data from one realization of the SIR model (Eq. (18)) in one node, starting with 99 susceptible, 1 infected, and 0 recovered individuals.

If we inspect the U matrix, it now includes results from two nodes above one another for each time point in the tspan vector. Please note that the U matrix is truncated here for readability.

```
R> U(result)[, 1:10]
```

```
   1  8 15 22 29 36 43 50 57 64
S 99 95 86 82 79 75 68 63 58 49
I  1  5  5  6  7  5  9  9 10 13
R  0  0  9 12 14 20 23 28 32 38
S 90 89 89 81 72 56 45 35 21 12
I  2  3  1  8 16 24 22 27 28 25
R  0  0  2  3  4 12 25 30 43 55
```

Thanks to the high performance of the simulator, we can easily expand the model to include several hundreds or thousands of nodes, and still run the simulations efficiently. We will illustrate this by exploring the dynamics when perturbing the model parameters of the population- and between-node prevalence in 500 nodes after 75 days. This can be achieved using the `run_outer` method. It takes a `SimInf_model`, a `formula` specifying the model `gdata` parameters to perturb, `x` and `y` arrays with values of how much to perturb the parameters, and a callback function `FUN`. The model `gdata` parameters on the right-hand-side of the formula are scaled with `y`. Similarly, the `gdata` parameters on the left-hand-side of the formula are scaled with `x`. For each combination of `x` and `y`, the model parameters are scaled and the function `FUN` called with the perturbed model.

```
R> u0 <- data.frame(S = rep(99, 500), I = rep(1, 500), R = rep(0, 500))
R> model <- SIR(u0, 1:75, beta = 0.16, gamma = 0.077)
R> x <- seq(from = 0.2, to = 1.8, by = 0.05)
R> y <- seq(from = 0.2, to = 1.1, by = 0.05)
R> pop <- run_outer(x, y, model, gamma ~ beta,
```

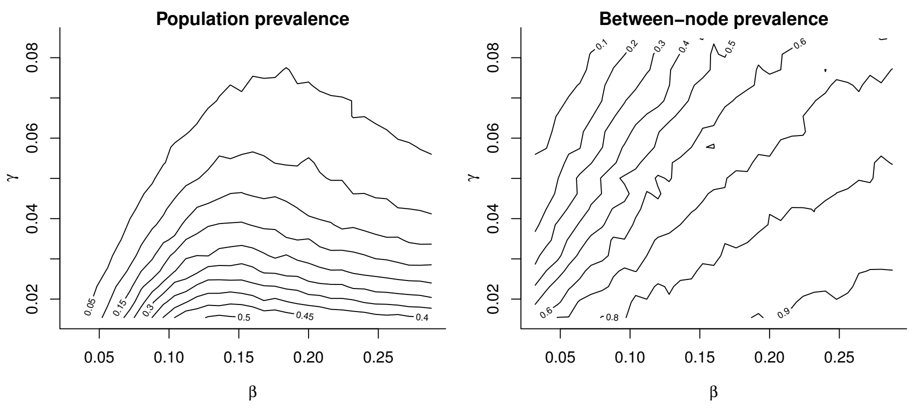


Figure 3: Contour plot of the population- and between-node prevalence after simulating 75 days of an `SIR` model in 500 nodes, starting with 99 susceptible, 1 infected, and 0 recovered individuals in each node at the first day.

```
+    function(model) {prevalence(run(model), "pop")[75]})
R> bnp <- run_outer(x, y, model, gamma ~ beta,
+    function(model) {prevalence(run(model), "bnp")[75]})
```

This illustrates that the between-node prevalence increases for increasing values of the transmission rate β, but decreases for increasing values of the recovery rate γ (Figure 3). As can be seen in the figure, the relationship is more complex for the population prevalence.

Specification of scheduled events in the `SIR` *model*

In this example, we will continue to work with the predefined `SIR` model to explore a within node disease spread similar to the first SIR example but with the additional feature that individuals enter and exit the simulation and that the disease can spread between nodes via movements of infected individuals between nodes.

We will start with the inclusion of a set of individuals that are moved between nodes in the network (*external transfer* events). These events are predefined prior to the simulation and are applied during the simulation process. For example, if your simulation time is in days, then after each day, **SimInf** will apply the events in the data and move individuals between nodes. In the following example we have five nodes with movements between nodes. To illustrate the movements themselves, we have only started with 30 individuals in the first node and the other four are empty, and we will run the simulation for five time steps:

```
R> u0 <- data.frame(S = c(10, 0, 0, 0, 0), I = c(10, 0, 0, 0, 0),
+    R = c(10, 0, 0, 0, 0))
```

Now we will define 6 scheduled events to include in the simulation. Below is a `data.frame`, that contains the events. Interpret it as follows:

1. In time step 2 we add 2 susceptible individuals to node 2
2. In time step 3 we move 2 individuals from node 1 to node 3
3. In time step 4 we remove 1 individual from node 2
4. In time step 4 we move 1 individual from node 1 to node 3
5. In time step 4 we move 1 individual from node 1 to node 4
6. In time step 5 we move 2 individuals from node 1 to node 5

```
R> events

  event time node dest n proportion select shift
1     1    2    2    0 2          0      1     0
2     3    3    1    3 1          0      2     0
3     0    4    2    0 1          0      2     0
4     3    4    1    3 1          0      2     0
5     3    4    1    4 1          0      2     0
6     3    5    1    5 2          0      2     0
```

Now we are ready to run the simulation using the events and the disease spread model as we did in earlier examples:

```
R> model <- SIR(u0, 1:5, events = events, beta = 0.16, gamma = 0.077)
R> U(run(model, threads = 1, seed = 22))

   1  2  3  4  5
S 10  8  7  7  7
I 10 12 11  9  7
R 10 10 11 11 11
S  0  2  2  1  1
I  0  0  0  0  0
R  0  0  0  0  0
S  0  0  1  1  1
I  0  0  0  0  0
R  0  0  0  1  1
S  0  0  0  0  0
I  0  0  0  1  1
R  0  0  0  0  0
S  0  0  0  0  0
I  0  0  0  0  2
R  0  0  0  0  0
```

If you inspect the result of this very simple simulation, you will see that 2 individuals were added to node 2, of which one individual was later removed, and that some individuals were removed from node 1 and placed in nodes 3, 4, and 5. So far this is not really earth-shattering, but if we do this again with a few more individuals then we can see how disease can spread from an infected node to a non-infected one. We will start with 5 nodes again, but node 1 is infected and nodes 2–5 now each have 100 susceptible and the simulation will run for 75 days:

```
R> u0 <- data.frame(S = c(90, 100, 100, 100, 100), I = c(10, 0, 0, 0, 0),
+    R = c(0, 0, 0, 0, 0))
```

And then we have a few movements of individuals from node 1 to the others:

```
R> events

  event time node dest n proportion select shift
1     3    3    1    3 1          0      2     0
2     3   25    1    2 9          0      2     0
3     3    5    1    3 1          0      2     0
4     3   10    1    3 5          0      2     0
```

Initialize and run the model

```
R> model <- SIR(u0, 1:75, events = events, beta = 0.16, gamma = 0.077)
R> result <- run(model, threads = 1, seed = 5)
```

Figure 4 shows the result from running the model. Note the time that it takes for node 2 to become positive relative to the movements. On day 25, 10 individuals are moved to the node causing the outbreak. Also despite the fact that we moved individuals from node 1 to node 3, there was no outbreak in node 3. This is because of the random selection process that select susceptible, infected or recovered individuals to move between the nodes.

```
R> plot(result, N = TRUE, spaghetti = TRUE, compartments = "I")
```

C code for the SIR *model*

The C code for the SIR model is defined in the source file 'src/models/SIR.c'. This file contains the SIR_run function to initialize the core solver (Listing 1 in the appendix), the transition rate functions (Listing 2 in the appendix) and the post time step function (Listing 3 in the appendix).

4.2. A second example: The SISe3_sp model

This section illustrates the specification of the SISe3_sp model, which is also predefined in the **SimInf** package. Two additional features, compared to the SIR model, are used by the SISe3_sp model: ageing of individuals by *internal transfer* events, and usage of the real valued continuous state V. The SISe3_sp model contains the two compartments susceptible (S) and infected (I) divided into the three age categories $j = \{1, 2, 3\}$ and the local environmental compartment (e) contaminated with free living pathogens (Figure 5). Moreover, nodes are connected spatially with local spread of the contaminated environmental compartment among

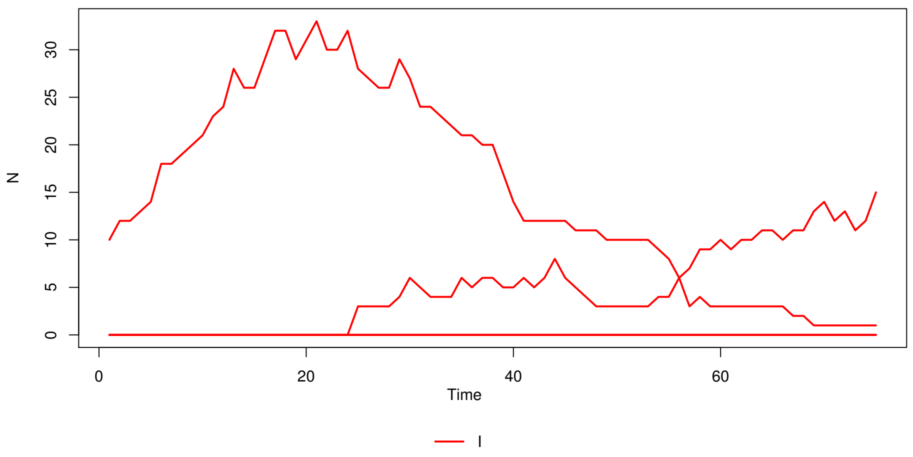


Figure 4: The number of infected individuals in each node when simulating 75 days of an `SIR` model in 5 nodes, starting with 90 susceptible and 10 infected individuals in node 1 and 100 susceptible individuals in nodes 2–5, and moving individuals from node 1 to nodes 2 and 3.

proximal nodes. The transmission route of infection to susceptible individuals is indirect via the local environment, contaminated by infected individuals. The number of individuals in each of the six compartments $\{S_1, I_1, S_2, I_2, S_3, I_3\}$ in node i at time t is `u[, i]`. The `SISe3_sp` model has two state transitions within each of the three age categories,

$$\begin{array}{ccc} S_{ij} & \xrightarrow{\upsilon_j \varphi_i} & I_{ij}, \\ I_{ij} & \xrightarrow{\gamma_j} & S_{ij}, \end{array} \tag{19}$$

where the state transition from susceptible to infected depends on the concentration of the environmental contamination $\varphi_i(t)$ of the pathogen in node i and the indirect transmission rate υ_j. The state transition from infected to susceptible depends on the recovery rate γ_j. Finally, the environmental infectious pressure is evolved by

$$\frac{d\varphi_i(t)}{dt} = \frac{\alpha I_i(t)}{N_i(t)} + \sum_k \frac{\varphi_k(t)N_k(t) - \varphi_i(t)N_i(t)}{N_i(t)} \cdot \frac{D}{d_{ik}} - \beta(t)\varphi_i(t), \tag{20}$$

where the constant α is the average shedding rate of the pathogen to the environment per infected individual, and $N_i = S_i + I_i$ the size of node i. The decay and removal of the pathogen is captured by β, which is allowed to vary with time. A spatial component with local spread among proximal nodes is also included in the model, where D is the rate of the local spread and d_{ik} the distance between the two holdings i and k. When running a trajectory of the `SISe3_sp` model, the environmental infectious pressure $\varphi_i(t)$ in each node is evolved in the post-time-step function by the Euler forward method. The value of $\varphi_i(t)$ is saved to the `V` matrix at the time-points specified by `tspan`.

Illustration of the stochastic step in the `SISe3_sp` *model*

To illustrate the stochastic step in Eq. (17), consider that the number of individuals in each compartment in node i `u[, i]` at time t is $\{12, 3, 63, 14, 92, 2\}$ and that the time to the

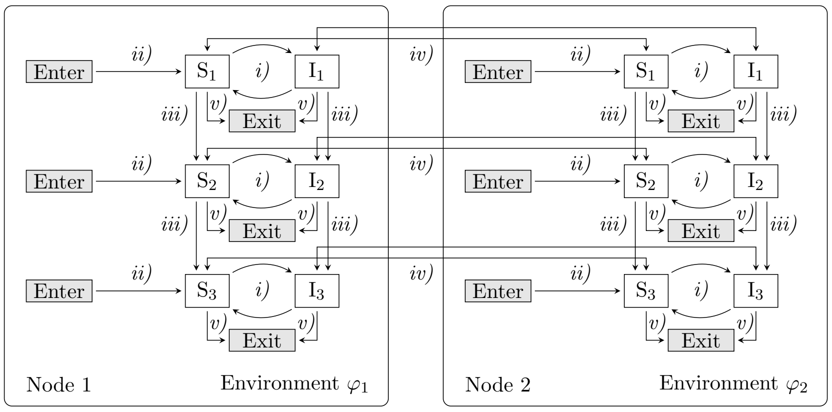


Figure 5: Schematic representation of the `SISe3_sp` compartment model in two nodes. The `SISe3_sp` model is defined by the two disease states susceptible (S) and infected (I) in three age categories and indirect transmission via an environmental infectious pressure φ_i. *i)* State transitions between the S and I compartments are modeled as a continuous-time discrete-state Markov process. The other state transitions are due to scheduled events: *ii) enter* events, *iii) internal transfer* events, *iv) external transfer* events, and *v) exit* events. The environmental infectious pressure φ_i is modeled with an ordinary differential equation and by default evolved by the Euler forward method.

next state transition is τ_i. Furthermore, assume that the next state transition is infected to susceptible in the first age category which corresponds to the second column in the state change matrix `S`. Thus, updating the compartments corresponds to adding `S[, 2]` to `u[, i]`, which gives $\{13, 2, 63, 14, 92, 2\}$. This also implies that the transition rates for the two transitions $S_{i,1} \to I_{i,1}$ and $I_{i,1} \to S_{i,1}$ must be recalculated according to `G[, 2]`.

$$\mathbf{S} = \begin{array}{c} \\ S_1 \\ I_1 \\ S_2 \\ I_2 \\ S_3 \\ I_3 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \qquad \mathbf{G} = \begin{array}{c} \\ S_1 \to I_1 \\ I_1 \to S_1 \\ S_2 \to I_2 \\ I_2 \to S_2 \\ S_3 \to I_3 \\ I_3 \to S_3 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Specification of scheduled events in the `SISe3_sp` *model*

The `SISe3_sp` model have been specified to handle births, imports, ageing, moving and exits of individuals (Table 4 in the appendix). When individuals enter the model (births, imports), they are added to the susceptible state in the age category specified by column 1, 2 or 3 in `E`. The columns 4, 5 and 6 in `E` are used for the other events (ageing, moving, exit) to sample individuals from the susceptible and infected states from each of the three ages categories. To move the sampled individuals in a *internal transfer* event (ageing) from age category 1 to age category 2, the susceptible and infected states should shift two steps to the corresponding state in age category 2, as specified in `N[, 1]`. Similarly, `N[, 2]` move susceptible and infected individuals from age category 2 to age category 3.

$$\mathbf{E} = \begin{array}{c} \\ S_1 \\ I_1 \\ S_2 \\ I_2 \\ S_3 \\ I_3 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad \mathbf{N} = \begin{array}{c} \\ S_1 \\ I_1 \\ S_2 \\ I_2 \\ S_3 \\ I_3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 2 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

Case study using the `SISe3_sp` *model*

In this section we will demonstrate how to construct and use the `SISe3_sp` model together with synthetic cattle events and population data included with the package **SimInf**. The data defines the initial population divided into three age categories (0 days $\leq$ *age 1* $<$ 120 days, 120 days $\leq$ *age 2* $<$ 365 days and *age 3* $\geq$ 365 days) in 1600 nodes and scheduled events over 4×365 days. The two data sets for the model are loaded with

```
R> data("u0_SISe3", package = "SimInf")
R> data("events_SISe3", package = "SimInf")
```

To model local spread of the environmental infectious pressure among proximal nodes, we have to generate a distance matrix. Let us use the synthetic data set (`nodes`) included in the **SimInf** package, which defines a cattle population consisting of 1600 herds located in a 50 square kilometer region, and let proximal neighbors be defined as neighbors within 2500m.

```
R> data("nodes", package = "SimInf")
R> d <- distance_matrix(x = nodes$x, y = nodes$y, cutoff = 2500)
```

Next we define `tspan` to simulate for 4 consecutive years returning the results every 7^{th} day

```
R> tspan <- seq(1, 4 * 365, by = 7)
```

We are now ready to create an `SISe3_sp` object using the `SISe3_sp` generating method. The parameters for this example have been chosen such that the between-node prevalence for a sample trajectory is around 10% and have a seasonal pattern. We assume that the average duration of infection is 10 days in all age categories, such that the recovery rate $\gamma_{1,2,3} = 0.1$ per animal per day, and assume that the indirect transmission rate of the environmental infectious pressure γ is equal in all age categories. Moreover, we fix the average shedding rate $\alpha = 1$ and allow the decay of the bacteria to vary in four seasonal intervals, $\beta_1, ..., \beta_4$ over the year. The end point (day of the year) for each season `beta_t1`, ..., `beta_t4` is determined by `end_t1`, ..., `end_t4`, respectively. We let the local environmental infectious pressure $\varphi_i = 0$ at the beginning of the simulation and let 10% of the herds start with 5% infected individuals.

```
R> set.seed(123)
R> i <- sample(1:1600, 160)
```

```
R> u0_SISe3$I_1[i] <- as.integer(u0_SISe3$S_1[i] * 0.05)
R> u0_SISe3$I_2[i] <- as.integer(u0_SISe3$S_2[i] * 0.05)
R> u0_SISe3$I_3[i] <- as.integer(u0_SISe3$S_3[i] * 0.05)
R> u0_SISe3$S_1[i] <- u0_SISe3$S_1[i] - u0_SISe3$I_1[i]
R> u0_SISe3$S_2[i] <- u0_SISe3$S_2[i] - u0_SISe3$I_2[i]
R> u0_SISe3$S_3[i] <- u0_SISe3$S_3[i] - u0_SISe3$I_3[i]
```

Let us now construct an `SISe3_sp` model object

```
R> model <- SISe3_sp(u0 = u0_SISe3, tspan = tspan, phi = rep(0, 1600),
+    events = events_SISe3, upsilon_1 = 0.013, upsilon_2 = 0.013, upsilon_3
+    = 0.013, gamma_1 = 0.1, gamma_2 = 0.1, gamma_3 = 0.1, alpha = 1,
+    beta_t1 = 0.095, beta_t2 = 0.12, beta_t3 = 0.10, beta_t4 = 0.15,
+    end_t1 = 91, end_t2 = 182, end_t3 = 273, end_t4 = 365, distance = d,
+    coupling = 0.1)
```

where `phi`, `beta_t1`, ..., `beta_t4`, `end_t1`, ..., `end_t4`, and `distance` is local data to each node, and passed to the transition rate functions and the post-time step function in the `ldata` parameter.

The package **SimInf** has a `summary` method for objects of class `SimInf_model`, such as the object `model`, which displays the model name, the number of nodes N_{nodes}, the number of compartments N_{comp}, the number of transitions N_{trans}, and the total number of scheduled events and information regarding each event type. If the model has not yet been run, the size of `U` and `V` is `0 x 0`.

```
R> summary(model)

Model: SISe3_sp

Number of nodes: 1600
Number of compartments: 6
Number of transitions: 6
Number of scheduled events: 783773
 - Exit: 182535 (n: min = 1 max = 1 avg = 1.0)
 - Enter: 182685 (n: min = 1 max = 1 avg = 1.0)
 - Internal transfer: 317081 (n: min = 1 max = 4 avg = 1.1)
 - External transfer: 101472 (n: min = 1 max = 1 avg = 1.0)

U: 0 x 0
V: 0 x 0
```

Furthermore, we can display the distribution of the scheduled events over time. Note that the synthetic events have been generated to illustrate features of the **SimInf** package and not to incorporate the full complexity of population demographics and a temporal network. The synthetic data mimics seasonality in real livestock data Bajardi *et al.* (2011); Dutta *et al.* (2014); Widgren, Engblom, Bauer, Frössling, Emanuelson, and Lindberg (2016), such

```
R> plot(events(model))
```

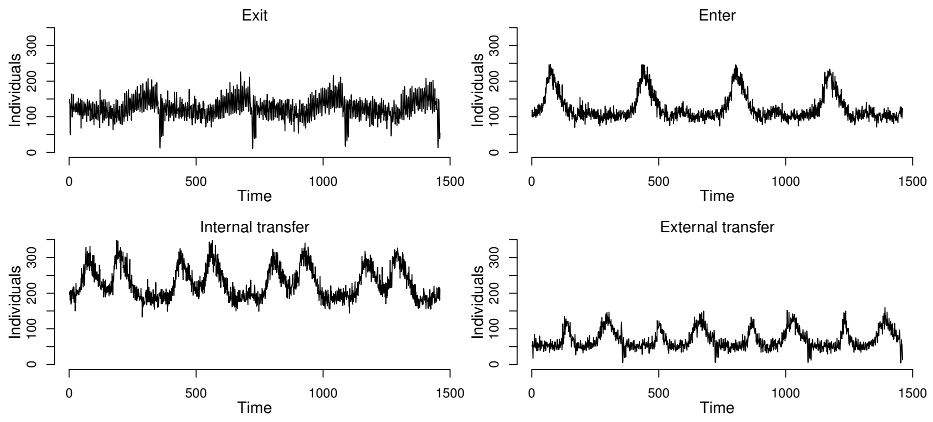


Figure 6: The distribution of the synthetic scheduled events data set `events_SISe3`, that is distributed with the **SimInf** package. The *enter* event adds individuals to a node. The *internal transfer* event changes the number of individuals in the compartments within one node. The *external transfer* event moves individuals from compartments in one node to compartments in a destination node. Finally, the *exit* event removes individuals from a node.

that the number of scheduled events varies over time, see Figure 6. However, the between-node movements (*external transfer*) have been generated from random connections, which is probably not realistic for real livestock data.

The scheduled events give rise to a seasonal variation of the population size in each age category that can be explored using data in the `U` matrix. The `susceptible` and `infected` methods take an `age` argument that can be used to determine the population size by age category. Let us first run a trajectory to generate data in `U`, and then plot the number of individuals in each age category (Figure 7).

```
R> result <- run(model, threads = 1, seed = 1)
R> N1 <- colSums(susceptible(result, age = 1) + infected(result, age = 1))
R> N2 <- colSums(susceptible(result, age = 2) + infected(result, age = 2))
R> N3 <- colSums(susceptible(result, age = 3) + infected(result, age = 3))
R> opar <- par(mfrow = c(3, 1), mar = c(2, 5, 1, 1))
R> plot(N3 ~ tspan, ylab = "N", type = "l", bty = "l", lty = 1)
R> plot(N2 ~ tspan, ylab = "N", type = "l", bty = "l", lty = 2)
R> plot(N1 ~ tspan, ylab = "N", type = "l", bty = "l", lty = 3)
R> mtext("Day", side = 1, line = 1, at = 750)
R> par(opar)
```

We might now be interested in exploring how various changes to the model could influence the spread of infection in the population of interconnected nodes. Let us compare the spatio-temporal distribution of disease spread when removing the effect of animal movements and

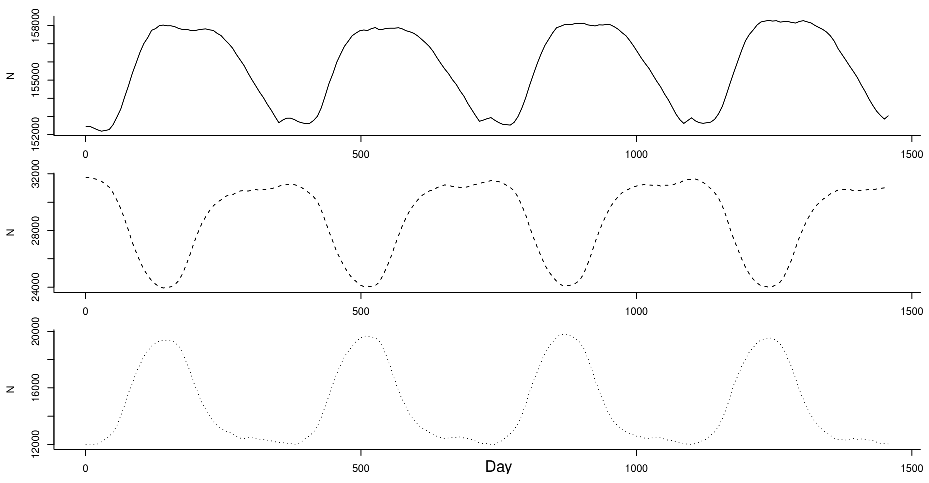


Figure 7: The population dynamics after running one trajectory over 4 × 365 days of the `SISe3` model using the scheduled events data contained in the **SimInf** package. Top: Plot showing the number of individuals older than 356 days. Middle: Plot showing the number of individuals between 120 and 365 days. Bottom: Plot showing the number of individuals younger than 120 days. Please note that the y-axis do not start at zero and have different scales.

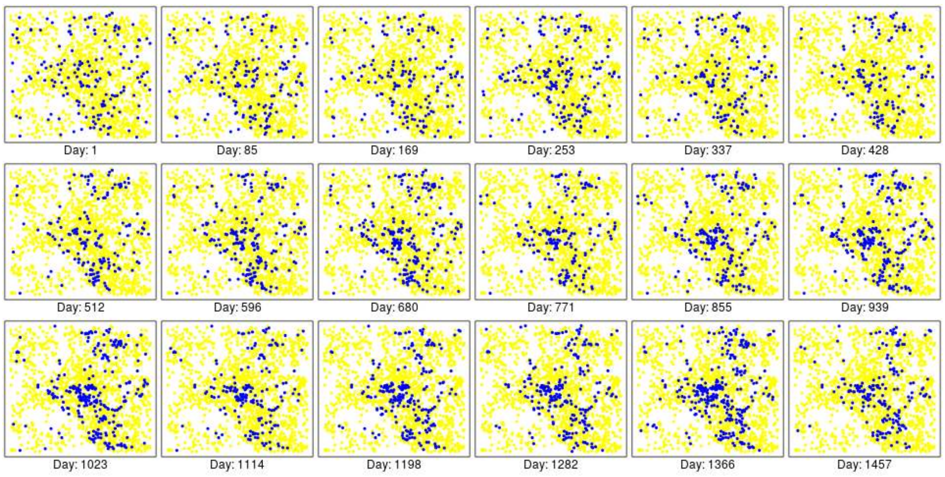
local spread. First, we need to create a utility function to visualize the spatio-temporal spread at evenly spaced time-points.

```
R> plot_nodes <- function(x) {
+    wnp <- prevalence(x, type = "wnp") > 0
+    opar <- par(mfrow = c(3, 6), mar = c(2, 0.3, 0.3, 0.3))
+    on.exit(opar)
+    for(i in seq(1, dim(wnp)[2], length.out = 18)) {
+      Si <- which(!wnp[, i])
+      Ii <- which(wnp[, i])
+      plot(nodes$x[Si], nodes$y[Si], col = "yellow", pch = 20, xlab = "",
+        ylab = "", xaxt = "n", yaxt = "n", ann = FALSE)
+      day <- 7 * as.integer(i - 1) + 1
+      mtext(sprintf("Day: %i", day), side = 1, line = 0.5)
+      points(nodes$x[Ii], nodes$y[Ii], col = "blue", pch = 20)
+    }
+  }
```

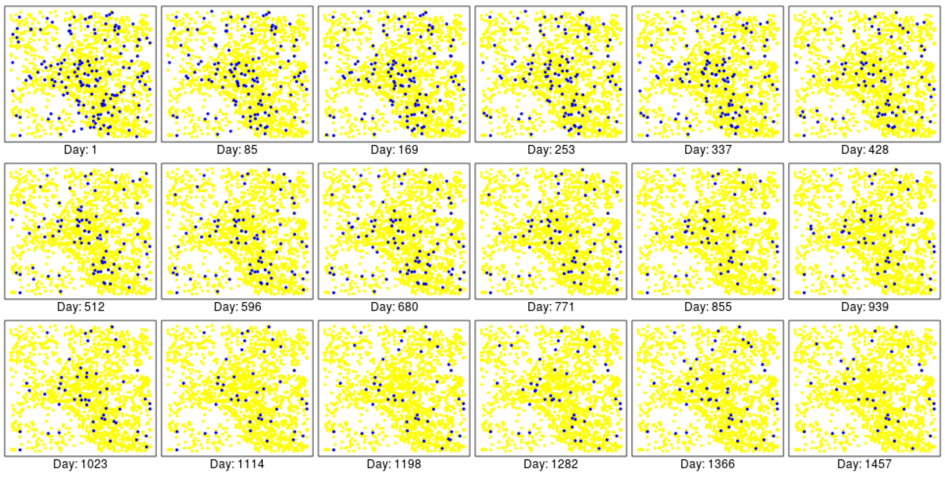
To remove local spread, we have only to set the `coupling` parameter to zero and run the model (Figure 8b).

```
R> model@gdata["coupling"] <- 0
R> plot_nodes(run(model, threads = 1, seed = 1))
```

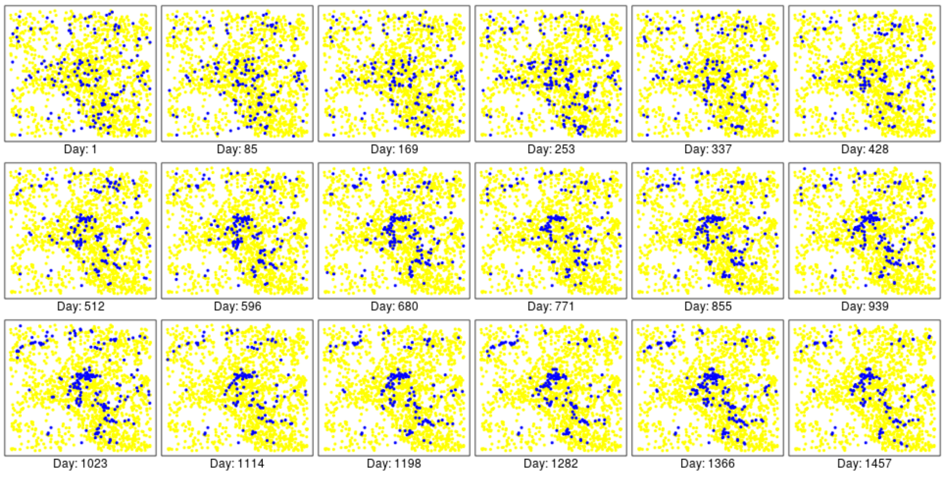
We would also like to explore the effect on the disease-spread when considering local spread only. However, removing all the *external transfer* events could potentially set a node in a state



(a) Between-node disease-spread by both movements of infected individuals and local spread among proximal nodes.



(b) Between-node disease-spread by movements of infected individuals.



(c) Between-node disease-spread by local spread among proximal nodes.

Figure 8: The distribution of infected nodes (among 1600 nodes within a 50 square kilometer region) at evenly spaced time points from one realization of the `SISe3_sp` model over 4×365 days using synthetic data included in the **SimInf** package.

where the other event types cannot be handled due to too few animals in the node. Another approach to solve this is to shift all infected animals to the susceptible state during an *external transfer* event so that infection is not transferred, even though an animal is moved. This can be achieved in **SimInf** by adding one column to the shift matrix `N` and use that column for the *external transfer* events. Figure 8c shows the result from running one trajectory with the modified model.

```
R> model@gdata["coupling"] <- 0.1
R> model@events@N <- cbind(model@events@N,
+    "3" = c(0L, -1L, 0L, -1L, 0L, -1L))
R> i <- which(model@events@event == 3)
R> model@events@shift[i] <- 3L
R> plot_nodes(run(model, threads = 1, seed = 1))
```

In this example, removing either local spread or animal movements was not sufficient to eradicate the disease (Figure 8).

5. Extending SimInf: New models

One of the design goals of **SimInf** was to make it extendable. The current design supports two ways to extend **SimInf** with new models, and this section describes the relevant steps to implement a new model. Since extending **SimInf** requires that C code can be compiled, you will first need to install a compiler. To read more about interfacing compiled code from R and creating R add-on packages, the `'Writing R extensions'` (`https://cran.r-project.org/doc/manuals/r-release/R-exts.html`) manual is the official guide and describes the process in detail. Another useful resource is the `'R packages'` book by Wickham (2015) (`http://r-pkgs.had.co.nz/`).

5.1. Using the model parser to define a new model

The easiest way to define a new model for **SimInf** is to use the model parser method `mparse`. It takes a character vector of transitions in the form of `"X -> propensity -> Y"` and generates both C code for the model and the two matrices `S` and `G`. The left hand side of the first '`->`'-sign is the initial state, the right hand side of the last '`->`'-sign is the final state, and the propensity is written between the '`->`'-signs. The special symbol '`@`' is reserved for the empty set $\emptyset$. We suggest to first draw a schematic representation of the model that includes all compartments and arrows for all state transitions. Then list the compartments in the order they should appear in the `U` matrix. For example, `transitions <- c("S -> b*S*I/(S+I+R) -> I", "I -> g*I -> R")` expresses an SIR model in a closed population, where `b` is the transmission rate, and `g` is the recovery rate. We also need to specify the compartments that defines the model.

```
R> transitions <- c("S -> b*S*I/(S+I+R) -> I", "I -> g*I -> R")
R> compartments <- c("S", "I", "R")
```

We can now use the `transitions` and `compartments` variables, together with constants `b` and `g` to build an object of class `'SimInf_mparse'` via a call to `mparse`. Before we can simulate from the model, it needs to be initialized with the initial condition `u0` and `tspan`.

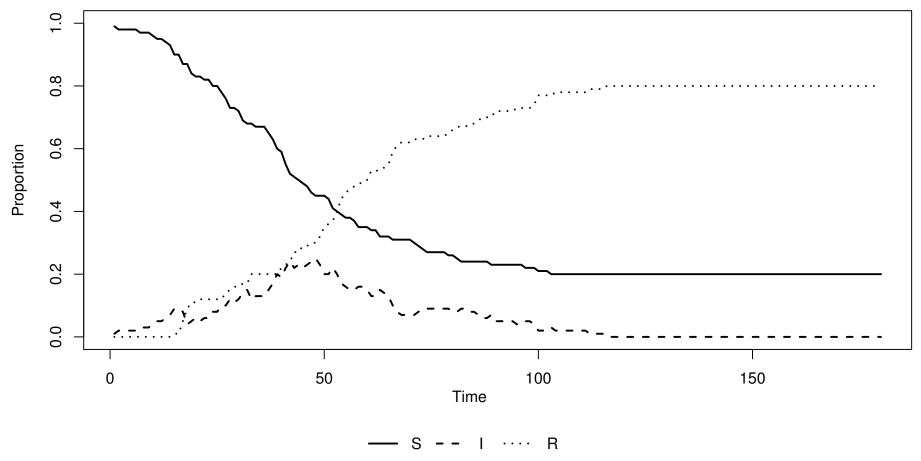


Figure 9: Example of simulated data from one realization of the `mparse` `SIR` model (Eq. (18)) in one node, starting with 99 susceptible, 1 infected, and 0 recovered individuals.

```
R> m <- mparse(transitions, compartments, b = 0.16, g = 0.077)
R> model <- init(m, u0 = cbind(S = 99, I = 1, R = 0), tspan = 1:180)
```

As in earlier examples, the `model` object can now be used to simulate data and plot the results. Internally, the C code that was generated by `mparse` is written to a temporary file when the `run` method is called. If the temporary file is compiled successfully, the resulting DLL is dynamically loaded and used to run one trajectory of the model. Once the simulator completes, the DLL is unloaded and the temporary files are removed.

```
R> result <- run(model, threads = 1, seed = 22)
R> plot(result)
```

The flexibility of the `mparse` approach allows for quick prototyping of new models or features. Let us elaborate on the previous example and explore the incidence cases per day. This can easily be done by adding one compartment `'Icum'` whose sole purpose is to keep track of how many individuals who become infected over time. We then calculate successive differences at each time-point in `'Icum'` and plot the result as an epidemic curve (Figure 10).

```
R> m <- mparse(c("S -> b*S*I/(S+I+R) -> I + Icum", "I -> g*I -> R"),
+    c("S", "I", "Icum", "R"), b = 0.16, g = 0.077)
R> model <- init(m, cbind(S = 99, I = 1, Icum = 0, R = 0), 1:180)
R> result <- run(model, threads = 1, seed = 22)
R> plot(stepfun(result@tspan[-1], diff(c(0, U(result)["Icum",]))),
+    main = "", xlab = "Time", ylab = "Number of cases",
+    do.points = FALSE)
```

Although **SimInf** was designed to study disease spread over temporal networks, it is not limited to that use case but can also be used to study the dynamics of other systems. In a final `mparse`

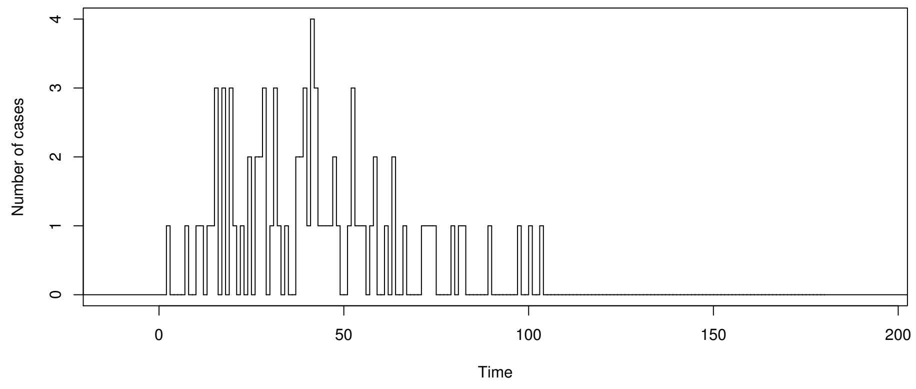


Figure 10: One realization of an epidemic curve displaying the number of incident cases per day in a node when simulating 180 days of the `mparse` `SIR` model (Eq. (18)), starting with 99 susceptible, 1 infected and 0 recovered individuals.

example we will illustrate the Rosenzweig-MacArthur predator-prey model demonstrated in the **GillespieSSA** package (Rosenzweig and MacArthur 1963; Pineda-Krch 2008). The model has a density-dependent growth in the prey and and a nonlinear Type-2 functional response in the predator (Rosenzweig and MacArthur 1963). Let R and F denote the number of prey and predators, respectively. The model consists of five transitions (Eq. (21)): *i)* prey birth, *ii)* prey death due to non-predatory events, *iii)* prey death due to predation, *iv)* predator birth, and *v)* predator death

$$\left.\begin{array}{ccc} \emptyset & \xrightarrow{b_R \cdot R} & R \\ R & \xrightarrow{(d_R+(b_R-d_R)\cdot R/K)\cdot R} & \emptyset \\ R & \xrightarrow{\alpha/(1+w\cdot R)\cdot R\cdot F} & \emptyset \\ \emptyset & \xrightarrow{b_F\cdot\alpha/(1+w\cdot R)\cdot R\cdot F} & F \\ F & \xrightarrow{d_F\cdot F} & \emptyset \end{array}\right\}, \tag{21}$$

where b_R, d_R, b_F, and d_F are the per capita birth and death rate of the prey and predator, respectively. Furthermore, K is the carrying capacity of the prey, α is the predation efficiency, and w is the degree of predator saturation (Pineda-Krch 2008). Using parameter values from Pineda-Krch (2008), we define the model as

```
R> m <- mparse(transitions = c("@ -> bR*R -> R",
+    "R -> (dR+(bR-dR)*R/K)*R -> @", "R -> alpha/(1+w*R)*R*F -> @",
+    "@ -> bF*alpha/(1+w*R)*R*F -> F", "F -> dF*F -> @"),
+    compartments = c("R", "F"), bR = 2, bF = 2, dR = 1, K = 1000,
+    alpha = 0.007, w = 0.0035, dF = 2)
R> model <- init(m, cbind(R = 1000, F = 100), 1:100)
```

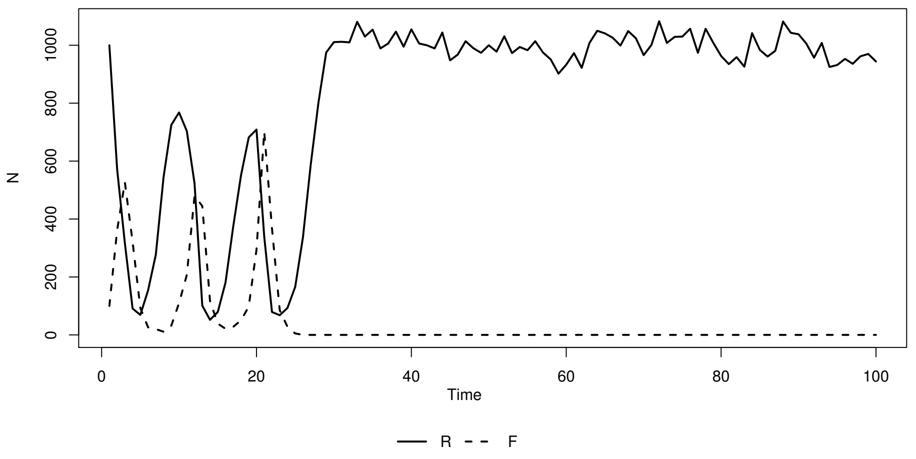


Figure 11: One realization of the `mparse` Rosenzweig-MacArthur predator-prey model. In this trajectory, the predator (`F`) go extinct after about 25 time steps while the prey (`R`) then fluctuates around a plateu of 1000 individuals.

```
R> result <- run(model, threads = 1, seed = 1)
R> plot(result, N = TRUE)
```

5.2. Use the SimInf solver from another package

Another possibility is to extend **SimInf** by creating an R add-on package that uses **SimInf** by linking to its core solver native routine, and this section describes the relevant steps to achieve this. Consider we wish to create a new add-on package **PredatorPrey** based on the Rosenzweig-MacArthur predator-prey model from the previous section. To facilitate this, the **SimInf** package includes a method (`package_skeleton`) that automates some of the setup for a new source package. It creates directories, saves R and C code files to appropriate places, and creates skeleton help files.

```
R> path = tempdir()
R> package_skeleton(m, "PredatorPrey", path = path)
```

where the first argument is the result (`SimInf_mparse` object) from the `mparse` method, and the second argument is the name of the package to create a skeleton for. The created R file (`'R/models.R'`) defines the S4 class `PredatorPrey` that contains the `SimInf_model` and a generating function to create a new object of the `PredatorPrey` model. The generating function is a template that might need to be updated to meet the specific requirements of the model, e.g. if the model should handle scheduled events, then `E` must be specified to select the subset of compartments involved in each specific event. Furthermore, if one of the events is an *internal transfer* event, `N` must be specified to shift individuals between compartments within one holding. It is also possible to define parameters that are either local (`ldata`) to each node or shared (`gdata`) between all nodes.

The C file (`'src/models.c'`) defines one function for each state transition, the post time step function and the model specific run function, and is automatically compiled when installing the package. The header file `"SimInf.h"` contains the declarations for these functions and should be included. The `SimInf_model_run` function is the interface from R to the core solver in C and list all function pointers to the transition rate functions in a vector in the order the state transitions appear in the dependency graph `G`, see Listing 1 in the appendix for an example from the `SIR` model and the use of the address of operator `'&'` to obtain the address of a function. The `SimInf_model_run` function must return the result from the call to the core solver with `SimInf_run`. The arguments to `SimInf_run` are the arguments passed to the `SimInd_model_run` function plus the vector of function pointers to the transition rate functions and the function pointer to the post time step function. To facilitate extraction of specific values from the data structures passed as arguments to the transition rate functions and the post time step function, we recommend using enumeration declarations to name the values, see Listing 2 in the appendix for an example.

The add-on **PredatorPrey** source package can now be built and installed with

```
R> install.packages(file.path(path, "PredatorPrey"), repos = NULL)
```

If the installation was successful, the newly installed package **PredatorPrey** can be loaded in R with the following command.

```
R> library("PredatorPrey")
```

6. Conclusion

In this paper we have introduced the R package **SimInf** which supports data-driven simulations of disease transmission over spatio-temporal networks. The package offers a very efficient and highly flexible tool to incorporate real data in simulations at realistic scales.

We hope that our package will facilitate incorporating large volumes of available data, for example, livestock data, in network epidemic models to better understand disease transmission in a temporal network and improve design of intervention strategies for endemic and emerging threats. Future efforts will be concentrated on a software development driven predominantly by actual use cases.

7. Acknowledgments

This work was financially supported by the Swedish Research Council within the UPMARC Linnaeus centre of Excellence (P. Bauer, S. Engblom), the Swedish Research Council Formas (S. Engblom, S. Widgren), the Swedish Board of Agriculture (S. Widgren), and by the Swedish strategic research program eSSENCE (S. Widgren).

References

Anonymous (2000). "Regulation (EC) No 1760/2000 of the European Parliament and of the Council of 17 July 2000 establishing a system for the identification and registration of bovine animals and regarding the labelling of beef and beef products." *Official Journal of the European Union*, **L 204**, 1–10.

Anonymous (2004). "Commission Regulation (EC) No 911/2004 of 29 April 2004 implementing Regulation (EC) No 1760/2000 of the European Parliament and of the Council as regards eartags, passports and holding registers." *Official Journal of the European Union*, **L 163**, 65–70.

Bajardi P, Barrat A, Natale F, Savini L, Colizza V (2011). "Dynamical Patterns of Cattle Trade Movements." *PloS one*, **6**(5), e19869. doi:10.1371/journal.pone.0019869.

Bauer P, Engblom S, Widgren S (2016). "Fast Event-Based Epidemiological Simulations on National Scales." *International Journal of High Performance Computing Applications*, **30**(4), 438–453. doi:10.1177/1094342016635723.

Brooks-Pollock E, de Jong M, Keeling M, Klinkenberg D, Wood J (2015). "Eight challenges in modelling infectious livestock diseases." *Epidemics*, **10**, 1–5. doi:http://dx.doi.org/10.1016/j.epidem.2014.08.005.

Büttner K, Krieter J, Traulsen A, Traulsen I (2016). "Epidemic Spreading in an Animal Trade Network - Comparison of Distance-Based and Network-Based Control Measures." *Transboundary and Emerging Diseases*, **63**(1), e122–e134. doi:10.1111/tbed.12245.

Chambers J (2008). *Software for Data Analysis: Programming with R*. Springer Science & Business Media.

Danon L, Ford AP, House T, Jewell CP, Keeling MJ, Roberts GO, Ross JV, Vernon MC (2011). "Networks and the Epidemiology of Infectious Disease." *Interdisciplinary perspectives on infectious diseases*, **2011**. doi:10.1155/2011/284909.

Drawert B, Engblom S, Hellander A (2012). "URDME: A Modular Framework for Stochastic Simulation of Reaction-Transport Processes in Complex Geometries." *BMC Systems Biology*, **6**(1), 1–17. doi:10.1186/1752-0509-6-76.

Dutta BL, Ezanno P, Vergu E (2014). "Characteristics of the Spatio-Temporal Network of Cattle Movements in France Over a 5-Year Period." *Preventive veterinary medicine*, **117**(1), 79–94. doi:10.1016/j.prevetmed.2014.09.005.

Engblom S, Ferm L, Hellander A, Lötstedt P (2009). "Simulation of Stochastic Reaction-Diffusion Processes on Unstructured Meshes." *SIAM Journal on Scientific Computing*, **31**(3), 1774–1797. doi:10.1137/080721388.

Engblom S, Widgren S (2017). "Data-driven computational disease spread modeling: from measurement to parametrization and control." In CR Rao, AS Rao, S Payne (eds.), *Disease Modeling and Public Health*, volume 36 of *Handbook of Statistics*. Elsevier, Amsterdam.

Galassi M, Davies J, Theiler J, Gough B, Jungman G, Alken P, Booth M, Rossi F (2009). *GNU Scientific Library Reference Manual*. Network Theory Limited, 3rd ed. edition. ISBN: 0-9546120-7-8 (http://www.gnu.org/software/gsl/).

Gillespie DT (1977). "Exact Stochastic Simulation of Coupled Chemical Reactions." *The Journal of Physical Chemistry*, **81**(25), 2340–2361. doi:10.1021/j100540a008.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2008). "statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data." *Journal of Statistical Software*, **24**(1), 1–11. doi:10.18637/jss.v024.i01.

Holme P, Saramäki J (2012). "Temporal Networks." *Physics Reports*, **519**(3), 97 – 125. doi:10.1016/j.physrep.2012.03.001.

Kernighan BW, Ritchie DM (1988). *C Programming Language*. 2nd edition. Prentice-Hall, Inc, Upper Saddle River, NJ 07458.

Merl D, Johnson LR, Gramacy RB, Mangel M (2010). "amei: An R Package for the Adaptive Management of Epidemiological Interventions." *Journal of Statistical Software*, **36**(6), 1–32. doi:10.18637/jss.v036.i06.

Meyer S, Held L, Höhle M (2017). "Spatio-Temporal Analysis of Epidemic Phenomena Using the R Package surveillance." *Journal of Statistical Software*, **77**(11), 1–55. doi:10.18637/jss.v077.i11.

OpenMP Architecture Review Board (2008). "OpenMP Application Program Interface Version 3.0." URL http://www.openmp.org/mp-documents/spec30.pdf.

Pineda-Krch M (2008). "GillespieSSA: Implementing the Gillespie Stochastic Simulation Algorithm in R." *Journal of Statistical Software*, **25**(1), 1–18. doi:10.18637/jss.v025.i12.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rosenzweig ML, MacArthur RH (1963). "Graphical representation and stability conditions of predator-prey interactions." *The American Naturalist*, **97**(895), 209–223. doi:10.1086/282272.

Samuel M Jenness SM, Goodreau SM, Morris M (2017). "EpiModel: An R Package for Mathematical Modeling of Infectious Disease over Networks." *Journal of Statistical Software*, **In press**.

Shirley MD, Rushton SP (2005). "The Impacts of Network Topology on Disease Spread." *Ecological Complexity*, **2**(3), 287–299. doi:10.1016/j.ecocom.2005.04.005.

Wickham H (2015). *R Packages*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. URL http://r-pkgs.had.co.nz/.

Widgren S, Engblom S, Bauer P, Frössling J, Emanuelson U, Lindberg A (2016). "Data-driven network modelling of disease transmission using complete population movement data: spread of VTEC O157 in Swedish cattle." *Veterinary Research*, **47**(1), 81. doi:10.1186/s13567-016-0366-5.

8. Appendix

8.1. Pseudo-code for the core simulation solve

Algorithm 1 Pseudo-code for the core simulation solver using direct SSA

```
 1: Run trajectory: Dispatch to model specific run method.
 2: C interface: Initialize model transition rate functions and post time step function.
 3: for all nodes i=1 to $N_{\text{nodes}}$ do in parallel
 4:    Compute transition rates for all transitions $\omega_{i,j}$, $j = 1, \ldots, N_{\text{trans}}$.
 5: end for
 6: while $t < T_{\text{End}}$ do
 7:    for all nodes i=1 to $N_{\text{nodes}}$ do in parallel
 8:       loop
 9:          Compute sum of transition rates $\lambda_i = \sum_{j=1}^{N_{\text{trans}}} \omega_{i,j}$
10:          Sample time to next stochastic event $\tau_i = -\log(r_1)/\lambda_i$ where $r_1$ is a uniformly distributed
             random number in the range (0, 1)
11:          if $\tau_i + t_i >= T_{\text{Next day}}$ then
12:             Move simulated time forward $t_i = T_{\text{Next day}}$
13:             go to 20
14:          end if
15:          Move simulated time forward $t_i = t_i + \tau_i$
16:          Determine which state transition happened; by inversion, find $n$ such that
             $\sum_{j=1}^{n-1} \omega_{i,j} < \lambda r_2 \leq \sum_{j=1}^{n} \omega_{i,j}$ where $r_2$ is a uniformly distributed random number in the
             range (0, 1)
17:          Update the compartments u[, i] using the state-change vector S[, n]
18:          Use the dependency graph G[, n] to recalculate affected transition rates $\omega_{i,j}$
19:       end loop
20:       Process $E_1$ events
21:    end for
22:    Process $E_2$ events
23:    for all nodes i=1 to $N_{\text{nodes}}$ do in parallel
24:       Call post time step function and update the continuous state variable v[ ,i].
25:    end for
26:    $T_{\text{Next day}} = T_{\text{Next day}} + 1$
27: end while
```

8.2. C code for the SIR model

Listing 1: Implementation of the function to init and run a simulation with the SIR model

```
SEXP SIR_run(SEXP model, SEXP threads, SEXP seed)
{
    TRFun tr_fun[] = {&SIR_S_to_I, &SIR_I_to_S};

    return SimInf_run(model, threads, seed, tr_fun, &SISe3_post_time_step);
}
```

Listing 2: Implementation of the transition rate functions in the SIR model for the transitions in Eq. (18) between the susceptible and infected compartments. The enumeration declarations are used to name the variable offsets and facilitate extraction of the values from the various data vectors.

```
/* Offset in integer compartment state vector */
enum {S, I, R};

/* Offsets in global data (gdata) to parameters in the model */
enum {BETA, GAMMA};

/* susceptible to infected: S -> I */
double SIR_S_to_I(const int *u, const double *v, const double *ldata,
                  const double *gdata, double t)
{
    const double S_n = u[S];
    const double I_n = u[I];
    const double n = S_n + I_n + u[R];

    if (n > 0.0)
        return (gdata[BETA] * S_n * I_n) / n;
    return 0.0;
}

/* infected to susceptible: I -> S */
double SIR_I_to_S(const int *u, const double *v, const double *ldata,
                  const double *gdata, double t)
{
    return gdata[GAMMA] * u[I];
}
```

Listing 3: Implementation of the post time step function in the SIR model. The post time step function should return a value > 0 if the node needs to recalculate the transition rates for the node, a value (error code) < 0 if an error is detected, or otherwise 0. Since the post time step function for the SIR model does not make any changes to a node, it always return 0.

```
int SIR_post_time_step(double *v_new, const int *u, const double *v,
                       const double *ldata, const double *gdata,
                       int node, double t)
{
    return 0;
}
```

8.3. Illustration of scheduled events

This section illustrates how the scheduled events for the `SISe3_sp` model are specified (Table 4) and how each event type is executed Figures 12,13,14,15,16

Action	`event`	`time`	`node`	`dest`	`n`	`proportion`	`select`	`shift`
Exit individuals in S_1 and I_1	0	t	x	0	n	0	4	0
Exit individuals in S_2 and I_2	0	t	x	0	n	0	5	0
Exit individuals in S_3 and I_3	0	t	x	0	n	0	6	0
Enter individuals in S_1 and I_1	1	t	x	0	n	0	1	0
Enter individuals in S_2 and I_2	1	t	x	0	n	0	2	0
Enter individuals in S_3 and I_3	1	t	x	0	n	0	3	0
Age individuals in S_1 and I_1	2	t	x	0	n	0	4	1
Age individuals in S_2 and I_2	2	t	x	0	n	0	5	2
Move individuals in S_1 and I_1	3	t	x	y	n	0	4	0
Move individuals in S_2 and I_2	3	t	x	y	n	0	5	0
Move individuals in S_3 and I_3	3	t	x	y	n	0	6	0

Table 4: Examples of the specification of a single row of scheduled event data in the `SISe3_sp` model to add, move or remove individuals during the simulation.

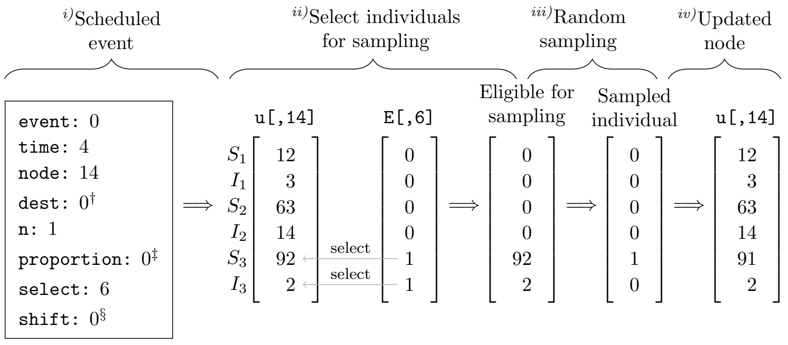


Figure 12: Illustration of a scheduled *exit* event (event = 0) in the `SISe3_sp` model at time = 4. The removal of one individual in the third age category $\{S_3, I_3\}$ from node 14. Interpreting the figure from left to right: *i)* A single row of the event data operating on node 14. *ii)* `u[, 14]` is the current state of node 14; `E[, 6]` is the 6th column in the select matrix that determines which compartments (age categories) that are eligible for sampling. *iii)* The operation of randomly sampling one individual (n = 1) to move from the compartments selected in step *ii*. *iv)* The resultant state of node 14 after subtracting the sampled individual in step *iii* from node 14. †`dest` and §`shift` are not used in a scheduled *exit* event. ‡`proportion` is not used when $n > 0$.

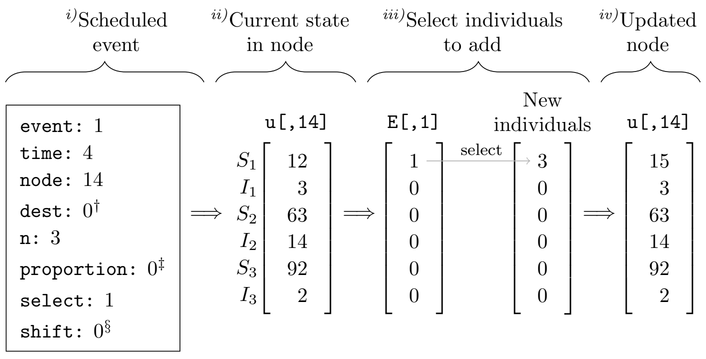


Figure 13: Illustration of a scheduled *enter* event (event = 1) in the `SISe3_sp` model at time = 4. Add three susceptible individuals to the first age category $\{S_1\}$ in node 14. Interpreting the figure from left to right: *i)* A single row of the event data operating on node 14. *ii)* `u[, 14]` is the current state of node 14. *iii)* `E[, 1]` is the first column in the select matrix that determines which compartments (age categories) the new individuals are added. *iv)* The resultant state of node 14 after adding the individuals in step *iii*. †`dest`, ‡`proportion` and §`shift` are not used in a scheduled *enter* event.

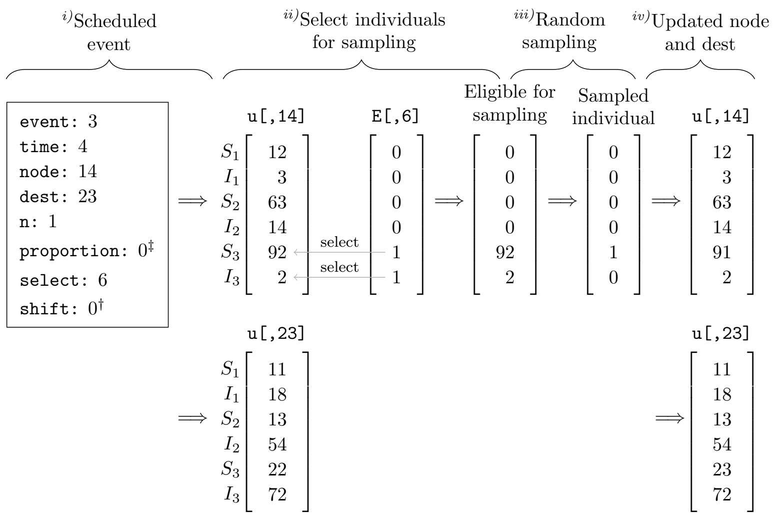


Figure 14: Illustration of a scheduled *external transfer* event (event = 3) in the `SISe3_sp` model at time = 4. The movement of one individual in the third age category $\{S_3, I_3\}$ from node 14 to destination node 23. Interpreting the figure from left to right: *i)* A single row of the event data operating on node 14 and destination node 23. *ii)* `u[, 14]` is the current state of node 14; `u[, 23]` is the current state of the destination node 23; `E[, 6]` is the 6^{th} column in the select matrix that determines which compartments (age categories) that are eligible for sampling. *iii)* The operation of randomly sampling one individual (n = 1) to move from the compartments selected in step *ii*. *iv)* The resultant state of node 14 and destination node 23 after subtracting the sampled individuals in step *iii* from node 14 and adding them to destination node 23. †`shift` can be used in a scheduled *external transfer* event, see Figure 16. ‡`proportion` is not used when $n > 0$.

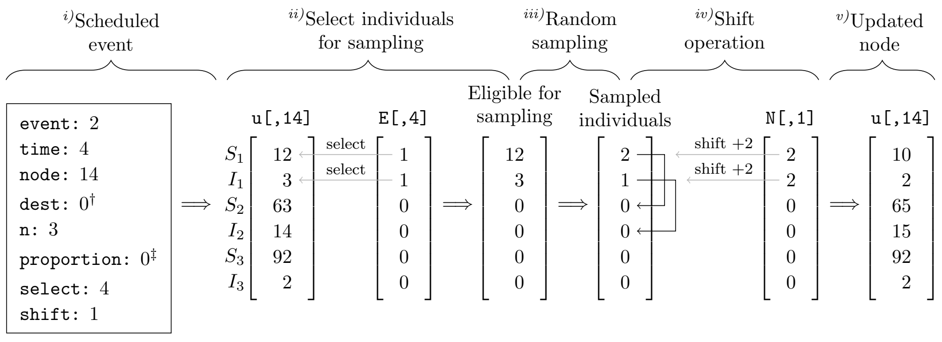


Figure 15: Illustration of a scheduled *internal transfer* event (event = 2) in the `SISe3_sp` model at time = 4. The ageing of three individuals in the first age category $\{S_1, I_1\}$. Interpreting the figure from left to right: *i)* A single row of the event data operating on node 14. *ii)* `u[, 14]` is the current state of node 14; `E[, 4]` is the 4th column in the select matrix that determines which compartments (age categories) that are eligible for sampling. *iii)* The operation of randomly sampling three individuals (n = 3) to age from the compartments selected in step *ii*. *iv)* The shift operation applies the shift specified in column 1 of the shift matrix (`N`) to the individuals sampled in step *iii*. *v)* The resultant state of node 14 after subtracting the sampled individuals in step *iii* and adding the individuals after the shift operation in step *iv*. †`dest` is not used in *internal transfers*. ‡`proportion` is not used when $n > 0$.

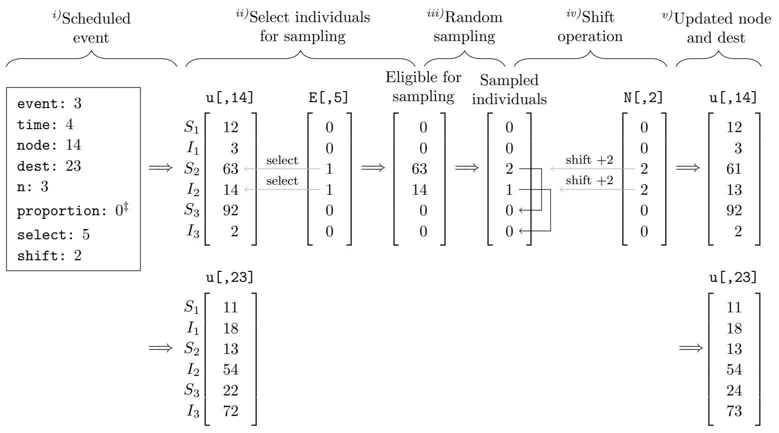


Figure 16: Illustration of a scheduled *external transfer* event (event = 3) in the `SISe3` model at time = 4. The ageing of three individuals in the second age category $\{S_2, I_2\}$ that are subsequently moved. Interpreting the figure from left to right: *i)* A single row of the event data operating on node 14 and destination node 23. *ii)* `u[, 14]` is the current state of node 14; `u[, 23]` is the current state of the destination node 23; `E[, 5]` is the 5th column in the select matrix that determines which compartments (age categories) that are eligible for sampling. *iii)* The operation of randomly sampling three individuals (n = 3) to move from the compartments selected in step *ii*. *iv)* The shift operation applies the shift specified in column 2 of the shift matrix (`N`) to the individuals sampled in step *iii*. *v)* The resultant state of node 14 and destination node 23 after subtracting the sampled individuals in step *iii* from node 14 and adding them to the destination node 23 after the shift operation in step *iv*. ‡`proportion` is not used when $n > 0$.

Affiliation:

Stefan Widgren
Department of Disease Control and Epidemiology
National Veterinary Institute
SE-751 89 Uppsala, Sweden
E-mail: stefan.widgren@sva.se
and
Division of Scientific Computing
Department of Information Technology
Uppsala University
SE-751 05 Uppsala, Sweden
E-mail: stefan.widgren@it.uu.se

Pavol Bauer
Division of Scientific Computing
Department of Information Technology
Uppsala University
SE-751 05 Uppsala, Sweden
E-mail: pavol.bauer@it.uu.se

Stefan Engblom
Division of Scientific Computing
Department of Information Technology
Uppsala University
SE-751 05 Uppsala, Sweden
E-mail: stefane@it.uu.se