o2geosocial

Alexis Robert

2020-06-30

Table of Contents

# Abstract

Individual transmission events during an outbreak give valuable information on the factors impacting the spread of an infectious disease. In recent years, methods have been developed to reconstruct transmission trees from onset dates and sequence data. Nevertheless, sequences can be uninformative, or sequencing can be scarce. We developed the package *o2geosocial* to reconstruct transmission trees using variables from routinely collected surveillance data, excluding genetic sequences. The model introduced in *o2geosocial* takes the reported age-group, onset date, location and genotype of the cases to reconstruct probabilistic transmission trees. In this vignette, we describe the structure and the different functions of the package. We also provide a tutorial on a simulated measles outbreak showing how to run a model, evaluate the output and visualise the results of the inference. In the second part of the tutorial, we customise the likelihood functions to introduce a different mobility model.

# Introduction

Regional immunity against infectious diseases is built up by past infections and, if a vaccine is available, vaccination campaigns. Social and spatial heterogeneity in disease incidence or vaccine coverage lead to under-immunised areas, also called pockets of susceptibles. Importation of cases into these areas can cause large transmission clusters and long-lasting outbreaks. The most vulnerable areas of a country could be identified using historical data on local vaccine coverage and incidence, but these data are often scarce, or unreliable. Another solution is to infer probabilistic transmission trees and clusters to identify in which regions importations repeatedly caused large transmission clusters.

The Wallinga-Teunis method was developed to infer probabilistic transmission trees from onset dates, serial interval and latent periods in a maximum likelihood framework. As genetic sequencing of cases during an outbreak became more common, new tools such as the R package [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2) showed that combining the timing of infection and the genetic sequences could improve the accuracy of inferred transmission trees. Nevertheless, sequencing cases remains costly, and the efficacy of the reconstruction methods depends on the proportion of sequenced cases, the quality of the sequences, and on the characteristics of the virus. For instance, the measles virus evolves very slowly, so sequences from unrelated cases can be very similar, which makes methods combining onset dates and genetic sequences ineffective.

Building upon the framework presented in [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2), we developed the R package *o2geosocial* to estimate the cluster size distribution from the onset date, age, location and genotype of the cases. Those variables are generally collected by surveillance systems and well reported. Using age-stratified contact matrices and mobility models, we combined the different variables into a likelihood of connection between cases. The package *o2geosocial* is ideal to study outbreaks where sequences are uninformative, either because only a small proportion of cases were sequenced or because the virus evolves too slowly. In this vignette, we first introduce the overall structure of *o2geosocial*, the components of the individual likelihood and the parameters; then we present a tutorial on how to develop a model to reconstruct the cluster size distribution with *o2geosocial*.

# Implementation

## Overall structure

The package *o2geosocial* uses Rcpp to incorporate C++ functions into a R framework. The general implementation of *o2geosocial* follows the structure of [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2): the main function of the package, so-called outbreaker(), takes five lists as inputs: i) ‘moves’, ii) ‘likelihoods’, iii) ‘priors’, iv) ‘data’, and v) ‘config’. These five lists can be generated and modified using the functions custom\_moves(), custom\_likelihoods(), custom\_priors(), create\_config() and outbreaker\_data(). Examples of how these functions are used to customize the model will be presented in the Tutorial section. The function outbreaker() uses Monte Carlo Markov Chains (MCMC) to sample from the posterior distribution, and infer the infection date, the infector, and the number of missing generations for each case. The package *o2geosocial* was designed to study datasets that include a large number of potentially unrelated cases and without information on the genetic sequence of cases, hence there are major differences between the implementation of *o2geosocial* and [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2).

In *o2geosocial* the genetic component of the likelihood is a binary value depending on the genotype of the cases. There can be only one genotype reported per transmission tree, hence the number of trees will be at least equal to the number of unique genotypes reported in the dataset. The genotype of a tree T is the genotype reported for at least one of the cases belonging to T. For each genotyped case and at every iteration, only cases from trees with the same genotype as , or without reported genotype should be listed as potential infectors. Therefore, ungenotyped cases belonging to a tree with a different genotype will not be potential infectors of at this iteration. Ungenotyped cases can be placed on any tree.

Routinely collected surveillance data may include thousands of cases from unrelated outbreaks. Therefore, it was crucial to speed up the algorithm. We added a pre-clustering step to the function outbreaker() to reduce the pool of potential infectors for each case, which can greatly reduce the computing time of the MCMC (Figure 1). In the pre-clustering step, the potential infectors for each case are listed. Cases with overlapping potential infectors, and their potential infectors, are grouped together. Cases from different groups cannot infect one another. The group each case belongs to is listed in the variable ‘is\_cluster’, which is an element of the list returned by the function outbreaker\_data(). We used three criteria to group together cases that can be connected: For each case , only cases reported in a radius of km, less than days before , and from similar or unreported genotype can be classified as potential infectors of . The parameter and are defined as inputs in the function create\_config().

Cases classified in the same group after the pre-clustering stage may still be unrelated (e.g. if several importations were reported simultaneously in the nearby region). We need to define a likelihood threshold which quantifies what is an unlikely connection. If the likelihood of connection between cases and is worse than , we consider it is more likely that was an import, unrelated to . In *o2geosocial*, the variable can either be an absolute (the likelihood threshold will be ) or relative value (a quantile of the likelihood of all connections in all samples), and is defined by the variables ‘outlier\_threshold’ and the boolean ‘outlier\_relative’ in create\_config(). If the initial number of importations is minimal, unlikely connections between cases can alter the inferred infection dates of cases and bias the generated transmission trees. Therefore, first we run a short MCMC and compute , the minimum number of connections worse than in the samples; then we add imports to the starting tree and run a long MCMC. Finally, we remove the connections worse than in the final samples and return the infector, infection date and probability of being an import for each case (Figure 1).

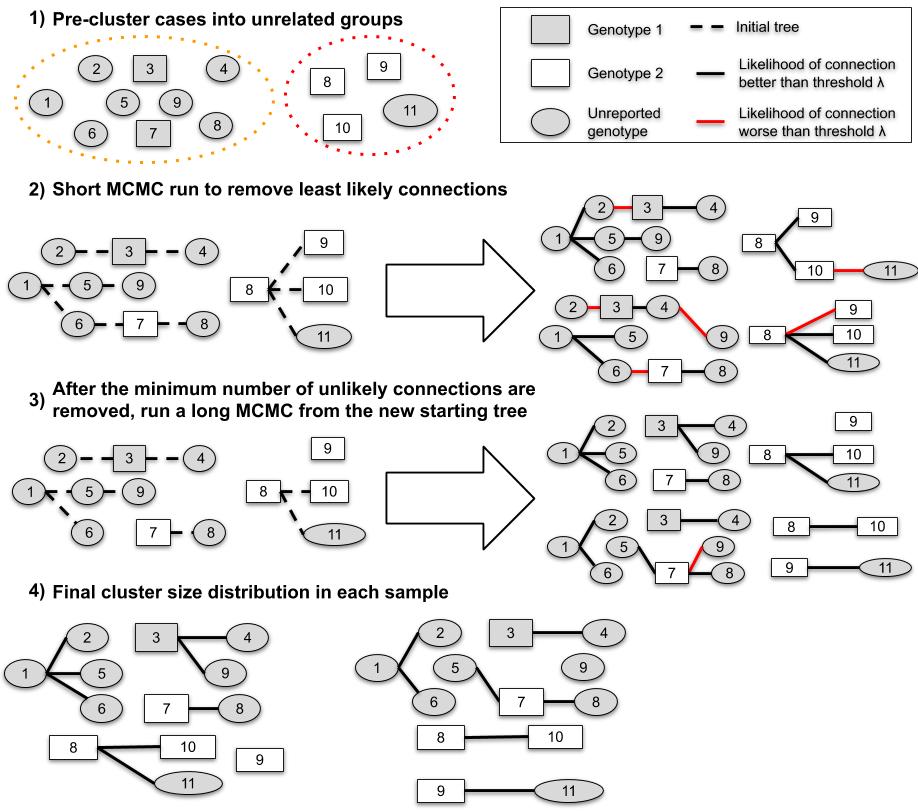


Figure 1: Example of the process to estimate the cluster size distribution and the import status of cases. In the first step, cases are split in two groups because they do not have overlapping potential infectors (they were reported in different places, or different time). In step 2, we estimate the minimum number of unlikely transmissions () in the samples (right panel). In step 3, we remove transmissions to the initial tree, and generate samples. Finally, we remove the unlikely connections in each sample of Step 3 and compute the inferred cluster size distribution.

## Likelihood and priors

The functions custom\_likelihoods() and custom\_priors() can be used to edit each component of the likelihood and priors.

### Genotype

The genetic component of the likelihood that a case of genotype was infected by a case belonging to the tree is defined as a binary value:

Therefore, ungenotyped cases may not be potential infector if they belong to a tree that contains another genotype than . The algorithm must look into the genotype of each tree containing potential infectors.

### Conditional report ratio

Like in the packages [*outbreaker*](https://CRAN.R-project.org/package=outbreaker) and [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2), we consider there can be missing cases in transmission chains. The number of generations between linked case is equal to one if infected . We define as the conditional report ratio of the trees. The conditional report ratio is not the same as the overall report ratio of an outbreak, since entirely unreported clusters, or missing cases before the ancestor of a tree will not change the value of . Only unreported cases within transmission chains will impact the conditional report ratio. By default, the probability of observing missing generation between and from the conditional report ratio follows an exponential distribution.

The conditional report ratio is estimated during the MCMC runs, it follows a beta distribution.

### Time

The time component of the likelihood is similar to what is used in the default version of outbreaker2. The probability of being the infection date of the case reported on depends on the distribution of the incubation period . The incubation period should be defined using the variable in the function outbreaker\_data().

The probability that was infected by , knowing their respective inferred dates of infection and is defined by the generation time of the disease (variable in outbreaker\_data()), and the number of generation between and . The operator was defined as , where is the convolution operator.

### Space

In *o2geosocial*, we introduce a spatial component to the likelihood of connection between cases. By default, we implemented a gravity model to illustrate the probability of connection between two regions and . Gravity models depend on the population sizes and , and the distance between regions . Given and two spatial parameters, the probability that a case coming from the region was infected by a case reported in is :

If we use an exponential gravity model, ; for a power-law gravity model: . The exponential gravity model has been shown to perform well to represent short-distance mobility patterns. As *o2geosocial* aims at reconstructing transmission in a community or a region, the default model in the function outbreaker() is an exponential gravity model. The type of gravity model can be changed by setting the parameter spatial\_method to “power-law”: create\_config(spatial\_method = "power\_law"). Other mobility models can be used by developing modules, we give an example of module in the tutorial where we replace the exponential gravity model by a Stouffer’s rank model.

The parameters and are estimated during the MCMC run. This requires re-computing the probability matrix twice per iteration, and can be time-consuming. Therefore, if or is not fixed, we allow for a maximum of 1 missing generation between cases () and only compute and for regions that could potentially be connected. By default, the uniform distribution of and is uniform.

### Age

Using the social contact matrices provided by large scale quantitative investigations such as the POLYMOD study (REF), we can calculate the probability of contact between cases of different age groups in different countries. In *o2geosocial*, given the age group of each case and the age-stratified social contact matrix, we introduced , the probability that a case aged infected a case aged . This corresponds to the proportion of contacts to that came from individuals of age .The contact matrix can be modified using the variable a\_dens in outbreaker\_data().

### Overall likelihood

The overall likelihood that a case was infected by the case is equal to where is the likelihood of connection between and and is defined as:

## Movements

At every iteration of the MCMC, a set of movements is used to propose an update of the transmission trees. This update is then accepted or rejected depending on the posterior value of the new trees. In *o2geosocial*, eight default movements are tested at each iteration. Three of them were already part of [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2) and were not modified (cpp\_move\_t\_inf() to change the infection date of cases; cpp\_move\_pi() to change the conditional report ratio; cpp\_move\_kappa() to change the number of generations between connected cases); two were edited to add the scanning of the transmissions trees to prevent from having different genotypes in the same tree (cpp\_move\_alpha(). cpp\_move\_swap\_cases()); The last three are new movements and were not part of [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2):

* cpp\_move\_a() and cpp\_move\_b() propose new values of the spatial parameters and and compute the matrix of probability of connection between regions.
* cpp\_move\_ancestor(): An ancestor is defined as the first case of a transmission tree. As only non-ancestors are moved in cpp\_move\_alpha(), we added this function to ensure good mixing of the ancestors of the transmission trees. For each ancestor , an index case is drawn from the poll of potential infectors, while another link is randomly picked and deleted. We then compare the new posterior value, and accept or reject the change.

# Tutorial

## Part one: First models and outputs

Two lists are attached to *o2geosocial*: toy\_outbreak\_short and toy\_outbreak\_long. Both describe simulated outbreaks and include 3 elements: i)cases: a data table with the ID, location, onset date, genotype, age group, import status, cluster, generation and infector of each case; ii) dt\_regions: a data table with the ID, population, longitude and latitude of each region; iii)age\_contact: a numeric matrix of the proportion of contact between age groups. Both simulations were ran using distributions of the serial interval and the latent period typically associated with measles outbreaks The dataset toy\_outbreak\_long contains 1940 cases simulated by the authors in the United States between 2003 and 2016, it can be used to reproduce the results described in <https://github.com/alxsrobert/datapaperMO>.

In this tutorial, we will analyse toy\_outbreak\_short. We will run a first model where the import status is inferred, and a second model that takes the import status from the reference dataset and only estimates the transmission trees. We will then evaluate the agreement between the inferred and the reference transmission clusters, and observe the added value of knowing the import status of the cases. Finally, we will give insight into the interpretation of the results and the geographical heterogeneity of the reconstructed transmission dynamics.

The results presented in this tutorial were generated using only 7,500 iterations to ensure a short run-time for the vignette. We usually recommend longer runs for the chains to converge and to sample from the posterior distribution.

### Run the models with `outbreaker()’

The dataset contains 75 simulated cases from different census tracts of Ohio in 2014 (variable cens\_tract). The variable cluster describes the transmission tree of each case, and “generation” is equal to the number of generations between the first case of the tree (generation = 1) and the case. We import the *o2geosocial* library and the toy\_outbreak\_short list, and extract the dataset.

library(o2geosocial)  
  
data("toy\_outbreak\_short")  
print(toy\_outbreak\_short$cases)

## ID State Date Genotype Cens\_tract age\_group import cluster  
## 1: 1 Ohio 2014-03-15 B3 39139002102 2 TRUE 1  
## 2: 2 Ohio 2014-02-04 D8 39139000700 12 TRUE 2  
## 3: 3 Ohio 2014-02-18 D8 39139001700 5 FALSE 2  
## 4: 4 Ohio 2014-04-22 B3 39139000800 15 TRUE 3  
## 5: 5 Ohio 2014-05-05 B3 39139002101 9 FALSE 3  
## ---   
## 147: 147 Ohio 2014-04-21 D8 39091004800 12 FALSE 27  
## 148: 148 Ohio 2014-04-20 D8 39091004600 11 FALSE 27  
## 149: 149 Ohio 2014-04-17 D8 39091004400 5 FALSE 27  
## 150: 150 Ohio 2014-04-28 D8 39091003800 10 FALSE 27  
## 151: 151 Ohio 2014-04-29 D8 39091004400 3 FALSE 27  
## generation infector\_ID  
## 1: 1 <NA>  
## 2: 1 <NA>  
## 3: 2 2  
## 4: 1 <NA>  
## 5: 2 4  
## ---   
## 147: 2 146  
## 148: 2 146  
## 149: 2 146  
## 150: 3 149  
## 151: 3 149

# Extract dataset  
dt\_cases <- toy\_outbreak\_short[["cases"]]  
dt\_cases <- dt\_cases[order(Date), ]

First we plot the cluster size distribution of the reference data (Figure 2). of the clusters contain less than 5 cases, of the clusters are isolated (also called singletons). Two clusters are much larger, and include and cases.

# Reference cluster size distribution  
hist(table(dt\_cases$cluster), breaks = 1:max(table(dt\_cases$cluster)),   
 ylab = "Number of clusters", xlab = "Cluster size", main = "", las=1)

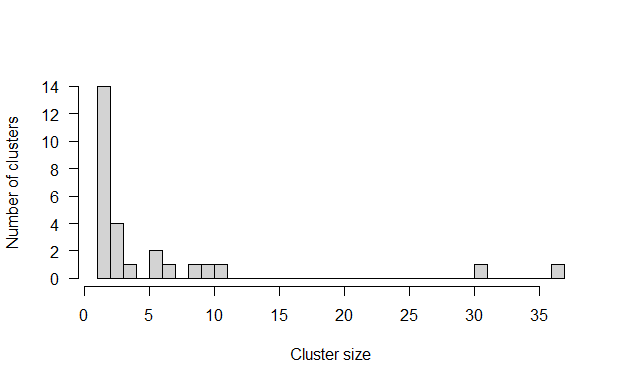


Figure 2: Cluster size distribution of the simulated dataset

We set up the distributions the model will use to reconstruct the transmission trees. We define f\_dens the duration of the latent period, and w\_dens the number of days between the infection dates of two connected cases, also called the serial interval. These distributions have been described for measles outbreaks in previous studies.

f\_dens <- dgamma(x = 1:100, scale = 0.43, shape = 26.83)  
  
w\_dens <- dnorm(x = 1:100, mean = 11.7, sd = 2.0)

The age specific social contact patterns can be imported from the element age\_contact of the list toy\_outbreak\_short. Alternatively, one can use the R package [*socialmixr*](https://CRAN.R-project.org/package=socialmixr) to import a social contact matrix from the POLYMOD survey.

# From the list toy\_outbreak\_short   
a\_dens <- toy\_outbreak\_short$age\_contact  
# Alternatively, from POLYMOD:  
polymod\_matrix <-  
 t(socialmixr::contact\_matrix(socialmixr::polymod,   
 countries="United Kingdom",  
 age.limits=seq(0, 70, by=5),  
 missing.contact.age="remove",  
 estimated.contact.age="mean",   
 missing.participant.age="remove")$matrix)  
polymod\_matrix <-data.table::as.data.table(polymod\_matrix)  
# Compute the proportion of connection to each age group  
a\_dens <- t(t(polymod\_matrix)/colSums(polymod\_matrix))

Finally, the distance matrix between regions is set up from the data table dt\_regions, element of toy\_outbreak\_short. We use the column population to set up the population vector pop\_vect. We compute the distance between each region into the distance matrix dist\_mat from the R package [*geosphere*](https://CRAN.R-project.org/package=geosphere).

# Extract all regions in the territory  
dt\_regions <- toy\_outbreak\_short[["dt\_regions"]]  
# Extract the population vector  
pop\_vect <- dt\_regions$population  
# Create the matrices of coordinates for each region (one "from"; one "to")  
mat\_dist\_from <- matrix(c(rep(dt\_regions$long, nrow(dt\_regions)),  
 rep(dt\_regions$lat, nrow(dt\_regions))), ncol = 2)  
mat\_dist\_to <- matrix(c(rep(dt\_regions$long, each = nrow(dt\_regions)),   
 rep(dt\_regions$lat, each = nrow(dt\_regions))),  
 ncol = 2)  
# Compute all the distances between the two matrices  
all\_dist <- geosphere::distGeo(mat\_dist\_from, mat\_dist\_to)  
# Compile into a distance matrix  
dist\_mat <- matrix(all\_dist/1000, nrow = nrow(dt\_regions))  
# Rename the matrix columns and rows, and the population vector  
names(pop\_vect) <- rownames(dist\_mat) <- colnames(dist\_mat) <-   
 dt\_regions$region

Now that all the distributions and matrices are set up, we create the lists data, config, moves, likelihoods and priors to run the main function of the package. In this example, we use the default parameters to build moves, likelihoods and priors with the same elements as described in section “Implementation”. The list data contains the distributions f\_dens and w\_dens, the population vector and the distance matrix, along with the onset dates, age group, location and genotype of the cases.

We implement two different models: in out1 the import status of the cases is inferred by the model, whereas in out2 the import status is set before the MCMC. Both models are run with iterations to find the import status and iterations for the main MCMC. As the import status of the cases is inferred in out1, we have to set a threshold to quantify what is an unlikely likelihood of transmission between cases. We use a relative outlier threshold at 0.9, which means that the threshold will be the decile of all the likelihoods in every sample.

# Default moves, likelihoods and priors  
moves <- custom\_moves()  
likelihoods <- custom\_likelihoods()  
priors <- custom\_priors()  
# Data and config, model 1  
data1 <- outbreaker\_data(dates = dt\_cases$Date, #Onset dates  
 age\_group = dt\_cases$age\_group, #Age group  
 region = dt\_cases$Cens\_tract, #Location  
 genotype = dt\_cases$Genotype, #Genotype  
 w\_dens = w\_dens, #Serial interval  
 f\_dens = f\_dens, #Latent period  
 a\_dens = a\_dens, #Age stratified contact matrix  
 population = pop\_vect, #Population   
 distance = dist\_mat #Distance matrix  
)  
config1 <- create\_config(data = data1,   
 n\_iter = 7500, #Iteration number: main run  
 n\_iter\_import = 2000, #Iteration number: short run  
 burnin = 1000, #burnin period: first run  
 outlier\_relative = T, #Absolute / relative threshold   
 outlier\_threshold = 0.9 #Value of the threshold  
)  
# Run model 1  
out1 <- outbreaker(data = data1, config = config1, moves = moves,   
 priors = priors, likelihoods = likelihoods)  
# Set data and config for model 2  
data2 <- outbreaker\_data(dates = dt\_cases$Date,   
 age\_group = dt\_cases$age\_group,  
 region = dt\_cases$Cens\_tract,  
 genotype = dt\_cases$Genotype, w\_dens = w\_dens,   
 f\_dens = f\_dens, a\_dens = a\_dens,  
 population = pop\_vect, distance = dist\_mat,  
 import = dt\_cases$import #Import status of the cases  
)  
config2 <- create\_config(data = data2,   
 find\_import = FALSE, # No inference of import status  
 n\_iter = 7500,   
 sample\_every = 50, # 1 in 50 iterations is kept  
 burnin = 1000)  
# Run model 2  
out2 <- outbreaker(data = data2, config = config2, moves = moves,   
 priors = priors, likelihoods = likelihoods)

The matrices out1 and out2 contain the posterior, likelihood, and prior of the trees at every iteration, along with the values of the spatial parameters a and b, the conditional report ratio pi, and the index, estimated infection date and number of generations for each case. We first remove the burnin period, which designates the number of iterations before the MCMC converged.

The function summary describes the features of the output matrix generated by outbreaker(). The function generates a list with the number of steps, the distributions of the posterior, likelihood and priors, the parameter distributions, the most likely infector and the probability of being an import for each case, and the cluster size distribution. we use it to describe the distribution of the inferred parameters.

burnin <- config1$burnin  
# Summary parameters a and b  
#Model 1  
print(summary(out1, burnin = burnin)$a)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.2052 0.5045 0.6740 0.6805 0.8481 1.4493

print(summary(out1, burnin = burnin)$b)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.09505 0.11476 0.12015 0.11981 0.12470 0.14374

# Model 2  
print(summary(out1, burnin = burnin)$a)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.2052 0.5045 0.6740 0.6805 0.8481 1.4493

print(summary(out1, burnin = burnin)$b)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.09505 0.11476 0.12015 0.11981 0.12470 0.14374

### Model evaluation

We now use barplot to show the median inferred cluster size distribution in out1 and out2 and the confidence intervals, and compare it with the reference data.

First, we group together clusters of similar sizes. The breaks of each group is written in the vector group\_cluster. In this example, the size categories are ; ; ; ; ; and . The inferred cluster size distributions are shown in the element cluster of the list generated by summary(out1), and can be aggregated using the input variable group\_cluster. The generated barplot highlights that the models generated a similar cluster size distribution, and are very close to the reference data (Figure 3). Inferring the import status in out1 did not generate more uncertainty, or more bias.

Nevertheless, the cluster size distribution when the import status of the cases is inferred depends on the likelihood threshold used to classify cases into clusters. Other models with different values of outlier\_threshold and outlier\_relative would show the impact of looser and stronger definitions of on the inferred cluster size distribution.

# We create groups of cluster size: initialise the breaks for each group  
group\_cluster <- c(1, 2, 3, 5, 10, 15, 40, 100) - 1  
# Reference data: h$counts  
h <- hist(table(dt\_cases$cluster), breaks = group\_cluster, plot = FALSE)  
  
# Grouped cluster size distribution in each run  
clust\_infer1 <- summary(out1, burnin = burnin,   
 group\_cluster = group\_cluster)$cluster  
clust\_infer2 <- summary(out2, group\_cluster = group\_cluster,   
 burnin = burnin)$cluster  
# Merge inferred and reference cluster size distributions into one matrix  
clust\_size\_matrix <- rbind(clust\_infer1["Median",], clust\_infer2["Median",],  
 h$counts)  
# Plot   
b <- barplot(clust\_size\_matrix, names.arg = colnames(clust\_infer1), las=1,  
 ylab = "Number of clusters", xlab = "Cluster size", main = "",   
 beside = T, ylim = c(0, max(c(clust\_infer1, clust\_infer2))))  
# Add the 50% CI  
arrows(b[1,], clust\_infer1["1st Qu.",], b[1,], clust\_infer1["3rd Qu.",],   
 angle = 90, code = 3, length = 0.1)  
arrows(b[2,], clust\_infer2["1st Qu.",], b[2,], clust\_infer2["3rd Qu.",],   
 angle = 90, code = 3, length = 0.1)  
# Add legend  
legend("topright", fill = grey.colors(3), bty = "n",  
 legend = c("Inferred import status",   
 "Known import status", "Simulated dataset"))

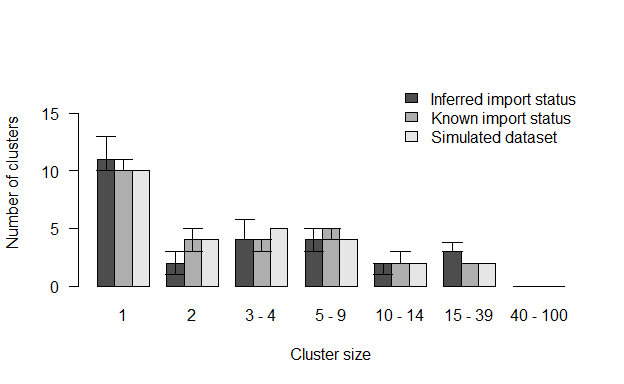


Figure 3: Comparison of inferred cluster size distribution with the reference data

Although the aggregated cluster size distributions are close to the reference data, we have to look into the reconstructed transmission trees to make sure the index assigned to each case are in agreement with the reference dataset. We write two functions: in index\_infer we compute the proportion of iterations where the inferred index is the actual index for each case (perfect match); In index\_clust we compute the proportion of iterations where the inferred index is from the same reference cluster as the actual index (close match).

#' Title: Compute the proportion of iterations in the outbreaker() output   
#` where the inferred index matches the actual index in dt\_cases  
#'  
#' @param dt\_cases: reference dataset  
#' @param out: Matrix output of outbreaker()  
#' @param burnin: Numeric, length of the burnin phase  
#'  
#' @return Numeric vector showing the proportion of iterations pointing to  
#' the correct index case  
index\_infer <- function(dt\_cases, out, burnin){  
 out\_index <- out[out$step > burnin, grep("alpha", colnames(out))]  
 ID\_index <- matrix(dt\_cases[unlist(out\_index), ID], ncol = nrow(dt\_cases))  
 match\_infer\_data <- t(ID\_index) == dt\_cases$infector\_ID  
 match\_infer\_data[is.na(t(ID\_index)) & is.na(dt\_cases$infector\_ID)] <- TRUE  
 prop\_correct <- rowSums(match\_infer\_data, na.rm = T)/ncol(match\_infer\_data)  
   
 return(prop\_correct)  
}  
# Same as index\_infer, except it returns the proportion of inferred indexes  
# who are in the same reference cluster as the case  
index\_clust <- function(dt\_cases, out, burnin){  
 out\_index <- out[out$step > burnin, grep("alpha", colnames(out))]  
 clust\_index <- matrix(dt\_cases[unlist(out\_index), cluster],   
 ncol = nrow(dt\_cases))  
 match\_infer\_data <- t(clust\_index) == dt\_cases$cluster  
 match\_infer\_data <- match\_infer\_data[!is.na(dt\_cases$infector\_ID),]  
   
   
 prop\_correct <- rowSums(match\_infer\_data, na.rm = T)/ncol(match\_infer\_data)  
   
 return(prop\_correct)  
}  
# Run index\_infer for each model  
index\_infer1 <- index\_infer(dt\_cases = dt\_cases, out = out1, burnin = burnin)  
index\_infer2 <- index\_infer(dt\_cases = dt\_cases, out = out2, burnin = burnin)  
# Run index\_clust for each model  
index\_clust1 <- index\_clust(dt\_cases = dt\_cases, out = out1, burnin = burnin)  
index\_clust2 <- index\_clust(dt\_cases = dt\_cases, out = out2, burnin = burnin)

The plots show that although the fits of the two models are close, inferring the import status of cases decreased the proportion of perfect and close match for almost every case (Figure 4). This highlights the importance of using reliable contact tracing investigations to investigate the import status of cases.

# Plot the sorted proportion in each model  
oldpar <- par(no.readonly =TRUE)  
par(bty = "n", mfrow = c(1, 2), mar = c(5,4,2,0), oma = c(0, 0, 0, 0))  
# Panel A  
plot(sort(index\_infer1), type = "l", ylab = "Proportion", xlab = "Case",   
 main = "A", las=1, col = grey.colors(3)[1], lwd = 3)  
lines(sort(index\_infer2), col = grey.colors(3)[2], lwd = 3)  
  
# Panel B  
plot(sort(index\_clust1), type = "l", xlab = "Case", ylab = "",   
 main = "B", las=1, col = grey.colors(3)[1], lwd = 3)  
lines(sort(index\_clust2), col = grey.colors(3)[2], lwd = 3)  
legend("bottomright", col = grey.colors(3)[1:2], lwd = 3, bty = "n",  
 legend = c("Inferred import status",   
 "Known import status"))

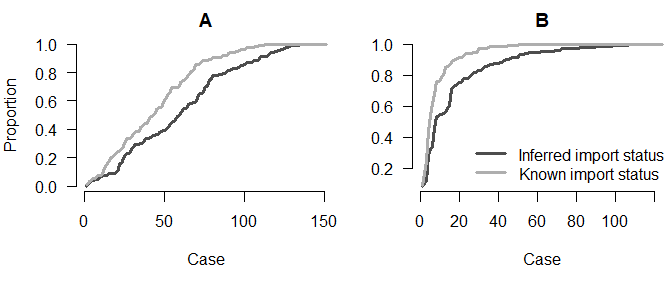


Figure 4: Panel A: Proportion of iterations with the correct index for each case; Panel B: Proportion of iterations where the index is from the correct cluster

par(oldpar)

### Interpretation of the results

In this section, we want to generate two maps to observe the geographical distribution of the importations, and the average number of secondary cases per region in out1 and out2. The maps will be generated using the package [*ggplot2*](https://CRAN.R-project.org/package=ggplot2)

First, we retrieve the boundary files of the census tract in Ohio. They will be used to generate the background of the maps, and are available using the library [*tigris*](https://CRAN.R-project.org/package=tigris). The boundary files are also available on the website [*census.gov*](https://www2.census.gov). We import them in a format compatible with the package [*sf*](https://CRAN.R-project.org/package=sf) and create one background map for each model.

library(ggplot2)  
# Read the shapefile and create one map for each model  
map1 <- tigris::tracts(state = "Ohio", class = "sf", progress\_bar = FALSE)  
map1$INTPTLON <- as.numeric(map1$INTPTLON)  
map1$INTPTLAT <- as.numeric(map1$INTPTLAT)  
map2 <- map1  
map1$model <- "Model 1"  
map2$model <- "Model 2"

We are interested in two outputs of the models: i) the number of import per region, and ii) the geographical distribution of the average number of the secondary cases per case in each model.

We compute the proportion of iterations where each case was an import in out1 and out2. The element tree of summary(out1) gives the most likely infector, the proportion of iterations where the index is the most likely infector (i.e. the support of the connection) and the median number of generations between the two cases, the most likely infection date and the chances of being an import for each case. We therefore add the columns prop\_infer1 and prop\_infer2 to dt\_cases. As the import status is not inferred in out2, prop\_infer2 is a binary value, and is the same as dt\_cases$import.

# Add the proportion of iteration in model 1 where each case is an import  
dt\_cases[, prop\_infer1 := summary(out1, burnin = burnin)$tree$import]  
# Add the proportion of iteration in model 2 where each case is an import  
dt\_cases[, prop\_infer2 := summary(out2, burnin = burnin)$tree$import]

We then aggregate the number of imports per region in each model, and name these vectors prop\_reg1 and prop\_reg2. We then add the number of imports in each region to the matrices describing the maps.

# Number of import per region in model 1  
prop\_reg1 <- dt\_cases[, .(prop\_per\_reg = sum(prop\_infer1)),   
 by = Cens\_tract]$prop\_per\_reg  
# Number of import per region in model 2  
prop\_reg2 <- dt\_cases[, .(prop\_per\_reg = sum(prop\_infer2)),   
 by = Cens\_tract]$prop\_per\_reg  
names(prop\_reg1) <- names(prop\_reg2) <- unique(dt\_cases$Cens\_tract)  
  
# Add the number of imports in each region to the maps  
map1$prop\_reg <- prop\_reg1[as.character(map1$GEOID)]  
map2$prop\_reg <- prop\_reg2[as.character(map2$GEOID)]

We now plot the number of imports per region in each model (Figure 5). Regions where no cases were reported are shown in grey. The right panel (Model out2) shows the geographical distribution of importations in the data.

We observe discrepancies between the two panels: although some regions have the correct number of imports inferred in most iterations, there are uncertainties for some imports in the left panel. Furthermore, the right panel shows there should be one import in each of the 4 regions in the central area, which seems to be underestimated in the left panel. These maps highlight the uncertainty added by the inference of the import status of each case.

# Merge maps  
maps <- rbind(map1, map2)  
  
limits\_long <- c(-84, -82)  
limits\_lat <- c(40, 41.5)  
maps <- maps[maps$INTPTLON > limits\_long[1] & maps$INTPTLON < limits\_long[2],]  
maps <- maps[maps$INTPTLAT > limits\_lat[1] & maps$INTPTLAT < limits\_lat[2],]  
  
# Plot: number of imports per region, two panels  
ggplot(maps) + geom\_sf(aes(fill = prop\_reg)) + facet\_grid(~model) +   
 scale\_fill\_gradient2(na.value = "lightgrey", midpoint = 0.8,   
 breaks = c(0, 0.5, 1, 1.5), name = "Nb imports",  
 low = "white", mid = "lightblue", high = "darkblue") +   
 coord\_sf(xlim = c(-83.8, -82.2), ylim = c(40.2, 41.3)) +  
 theme\_classic(base\_size = 9)

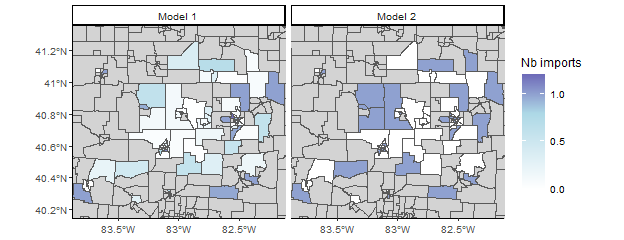


Figure 5: Average number of import cases per census tract

Then, we want to compute the average number of secondary cases per case in each region. We write a function to get the number of secondary cases in each iteration, and aggregate it per region. We compute the median across all iterations per region. We can observe the dispersion of the number of secondary cases by taking another quantile than the median, for example, we can use n\_sec1 <- apply(n\_sec\_tot1[,-1], 1, function(X) quantile(X, 0.25)) to get the first quartile in each region.

#' Title: Compute the number of secondary cases per case in each region  
#'  
#' @param dt\_cases: reference dataset  
#' @param out: Matrix output of outbreaker()  
#' @param burnin: Numeric, length of the burnin phase  
#'  
#' @return A numeric matrix: the first column is the census tract ID, the  
#' other columns show the number of secondary cases per case. Each row   
#' corresponds to a different iteration.  
n\_sec\_per\_reg <- function(dt\_cases, out, burnin){  
 n\_sec <- apply(out[out$step > burnin, grep("alpha", colnames(out))], 1,   
 function(X){  
 X <- factor(X, 1:length(X))  
 return(table(X))})  
 tot\_n\_sec\_reg <- aggregate(n\_sec, list(dt\_cases$Cens\_tract), sum)  
 tot\_n\_sec\_reg <- cbind(tot\_n\_sec\_reg[, 1],   
 tot\_n\_sec\_reg[, -1] / table(dt\_cases$Cens\_tract))  
 return(tot\_n\_sec\_reg)  
}  
   
n\_sec\_tot1 <- n\_sec\_per\_reg(dt\_cases = dt\_cases, out = out1, burnin = burnin)  
n\_sec\_tot2 <- n\_sec\_per\_reg(dt\_cases = dt\_cases, out = out2, burnin = burnin)  
n\_sec1 <- apply(n\_sec\_tot1[,-1], 1, median)  
n\_sec2 <- apply(n\_sec\_tot2[,-1], 1, median)  
names(n\_sec1) <- names(n\_sec2) <- unique(dt\_cases$Cens\_tract)

We add the number of secondary cases per region to the matrices describing the maps.

map1$n\_sec <- as.numeric(n\_sec1[as.character(map1$GEOID)])  
map2$n\_sec <- as.numeric(n\_sec2[as.character(map2$GEOID)])

We plot the geographical distribution of the median number of secondary cases in each region (Figure 6). The maps generated by the two models are very similar. Both of them show the spatial heterogeneity of the number of secondary transmissions. Some regions seem to consistently cause more secondary cases. There are minor discrepancies between the maps, but they show the same pattern. If we change the vectors n\_sec1 and n\_sec2 to plot different deciles, we can get an idea of the dispersion of the number of secondary cases in the different iterations of the models.

# Merge maps  
maps <- rbind(map1, map2)  
  
limits\_long <- c(-84, -82)  
limits\_lat <- c(40, 41.5)  
maps <- maps[maps$INTPTLON > limits\_long[1] & maps$INTPTLON < limits\_long[2],]  
maps <- maps[maps$INTPTLAT > limits\_lat[1] & maps$INTPTLAT < limits\_lat[2],]  
  
# Plot the geographical distribution of the number of secondary cases  
ggplot(maps) + geom\_sf(aes(fill = n\_sec)) + facet\_grid(~model) +   
 scale\_fill\_gradient2(na.value = "lightgrey", mid = "lightblue",  
 low = "white", midpoint = 1, high = "darkblue",  
 breaks = seq(0, 5, 0.5),name = "Sec cases") +  
 coord\_sf(xlim = c(-83.8, -82.2), ylim = c(40.2, 41.3)) +  
 theme\_classic(base\_size = 9)

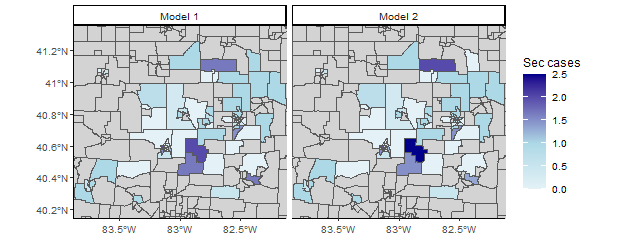


Figure 6: Median number of secondary transmission per case in each census tract

## Part two: Modify the likelihoods, priors and movements lists: the Stouffer’s rank module

In the previous section, we ran and evaluated two different models to reconstruct transmission clusters from surveillance data, and highlighted the spatial heterogeneity of measles transmission in the region. These models were run using the default likelihood and movement functions. Now we develop a third model, where the spatial connection between regions is based on the Stouffer’s rank method.

The Stouffer’s rank model, also named the “law of intervening opportunities” does not take absolute distance into account to compute the probability of connection between regions. In this model, the connectivity between the regions and only depends on the summed population of all the towns closer to than . If we define this collection of towns , the distance of Stouffer is then . From this, we deduce the probability that a case from region was infected by a case from region as

This model is actually similar to the power-law gravity model, with a two differences: \* Each cell of the distance matrix should be equal to .

* There’s only one spatial parameter to estimate.

First, we create the initial distance matrix in the Stouffer’s rank model dist\_mat\_stouffer:

# For every column of the distance matrix, use the cumulative sum of the   
# population vector ordered by the distance. Remove the values where   
# the distance between the regions is above gamma  
dist\_mat\_stouffer <- apply(dist\_mat, 2, function(X){  
 pop\_X <- cumsum(pop\_vect[order(X)])  
 omega\_X <- pop\_X[names(X)]  
 omega\_X[X > config1$gamma] <- -1  
 return(omega\_X)  
})  
# New value of gamma  
gamma <- max(dist\_mat\_stouffer) + 1  
# Negative value of distance set above gamma  
dist\_mat\_stouffer[dist\_mat\_stouffer == -1] <- max(dist\_mat\_stouffer) \* 2

We now write the function cpp\_stouffer\_move\_a to replace the default movement function cpp\_move\_a. As the Stouffer’s rank and the power law gravity models are close, we do not need to re-write cpp\_log\_like, the default function to compute the probability matrix. For other distance models, it may be required to create a new version of cpp\_log\_like and call it in the movement function. In cpp\_stouffer\_move\_a, there is no parameter , both spatial parameters are equal to . We use the package [*Rcpp*](https://CRAN.R-project.org/package=Rcpp) to source the new movement function.

// [[Rcpp::depends(o2geosocial)]]  
#include <Rcpp.h>  
#include <Rmath.h>  
#include <o2geosocial.h>  
  
// [[Rcpp::export()]]  
Rcpp::List cpp\_stouffer\_move\_a(Rcpp::List param, Rcpp::List data, Rcpp::List config,  
 Rcpp::RObject custom\_ll, Rcpp::RObject custom\_prior) {  
 // Import parameters  
 Rcpp::List new\_param = clone(param);  
 double gamma = config["gamma"];  
 int max\_kappa = config["max\_kappa"];  
 Rcpp::String spatial = config["spatial\_method"];  
 Rcpp::IntegerVector region = data["region"];  
 Rcpp::NumericMatrix distance = data["distance"];  
 Rcpp::NumericMatrix can\_be\_ances\_reg = data["can\_be\_ances\_reg"];  
 Rcpp::NumericVector population = data["population"];  
 Rcpp::NumericVector limits = config["prior\_a"];  
 // Size of the probability matrix  
 Rcpp::List new\_log\_s\_dens = new\_param["log\_s\_dens"];  
 Rcpp::NumericMatrix probs = new\_log\_s\_dens[0];  
 int nb\_cases = pow(probs.size(), 0.5);  
 // Draw new value of a  
 Rcpp::NumericVector new\_a = new\_param["a"];  
 double sd\_a = static\_cast<double>(config["sd\_a"]);  
 double old\_logpost = 0.0, new\_logpost = 0.0, p\_accept = 0.0;  
 // proposal (normal distribution with SD: config$sd\_a)  
 new\_a[0] += R::rnorm(0.0, sd\_a); // new proposed value  
 if (new\_a[0] < limits[0] || new\_a[0] > limits[1]) {  
 return param;  
 }  
 // Generate new probability matrix  
 new\_param["log\_s\_dens"] = o2geosocial::cpp\_log\_like(population, distance,  
 can\_be\_ances\_reg,   
 new\_a[0], new\_a[0],  
 max\_kappa, gamma,   
 spatial, nb\_cases);  
 // Compare old and new likelihood values  
 old\_logpost = o2geosocial::cpp\_ll\_space(data, config, param,   
 R\_NilValue, custom\_ll);  
 new\_logpost = o2geosocial::cpp\_ll\_space(data, config, new\_param,  
 R\_NilValue, custom\_ll);  
 // Add prior values  
 old\_logpost += o2geosocial::cpp\_prior\_a(param, config, custom\_prior);  
 new\_logpost += o2geosocial::cpp\_prior\_a(new\_param, config, custom\_prior);  
 // Accept or reject proposal  
 p\_accept = exp(new\_logpost - old\_logpost);  
 if (p\_accept < unif\_rand()) {  
 return param;  
 }  
 return new\_param;  
}

We now modify the element of the list moves, and replace it with cpp\_stouffer\_move\_a. As b is not estimated in this model, we create the null function f\_null, and modify the list priors.

moves3 <- custom\_moves(a = cpp\_stouffer\_move\_a)  
  
f\_null <- function(param) {  
 return(0.0)  
}  
priors3 <- custom\_priors(b = f\_null)

Finally, we set up the lists data and config: the distance matrix for this run is dist\_mat\_stouffer; we do not move the parameter b, and we change the value of gamma. We then run out\_stouffer.

# Set data and config lists  
data3 <- outbreaker\_data(dates = dt\_cases$Date, #Onset dates  
 age\_group = dt\_cases$age\_group, #Age group  
 region = dt\_cases$Cens\_tract, #Location  
 genotype = dt\_cases$Genotype, #Genotype  
 w\_dens = w\_dens, #Serial interval  
 f\_dens = f\_dens, #Latent period  
 a\_dens = a\_dens, #Age stratified contact matrix  
 population = pop\_vect, #Population   
 distance = dist\_mat\_stouffer #Distance matrix  
)  
config3 <- create\_config(data = data3,   
 gamma = gamma,  
 move\_b = FALSE, # b is not estimated  
 init\_b = 0,   
 spatial\_method = "power-law",  
 n\_iter = 7500, #Iteration number: main run  
 n\_iter\_import = 2000, #Iteration number: short run  
 burnin = 1000, #burnin period: first run  
 outlier\_relative = T, #Absolute / relative threshold  
 outlier\_threshold = 0.9 #Value of the threshold  
)  
# Run the model using the Stouffer's rank method  
out\_stouffer <- outbreaker(data = data3, config = config3, moves = moves3,   
 priors = priors3, likelihoods = likelihoods)

As we did in the previous section, we plot the inferred cluster size distribution and compare it to the reference data (Figure 7). We observe minor discrepancies between the two distributions, notably most of the iterations show one larger cluster (above 40 cases), which is not present in the reference data. Otherwise the two distributions are close.

# Grouped cluster size distribution in the Stouffer's rank model  
clust\_infer\_stouf <- summary(out\_stouffer, burnin = burnin,   
 group\_cluster = group\_cluster)$cluster  
# Merge inferred and reference cluster size distributions  
clust\_size\_matrix <- rbind(clust\_infer\_stouf["Median",], h$counts)   
# Plot the two distributions  
b <- barplot(clust\_size\_matrix, names.arg = colnames(clust\_infer\_stouf),   
 beside = T)  
# Add CIs  
arrows(b[1,], clust\_infer\_stouf["1st Qu.",], b[1,],   
 clust\_infer\_stouf["3rd Qu.",], angle = 90, code = 3, length = 0.1)  
legend("topright", fill = grey.colors(2), bty = "n",  
 legend = c("Inferred import status, Stouffer's rank method",   
 "Simulated dataset"))

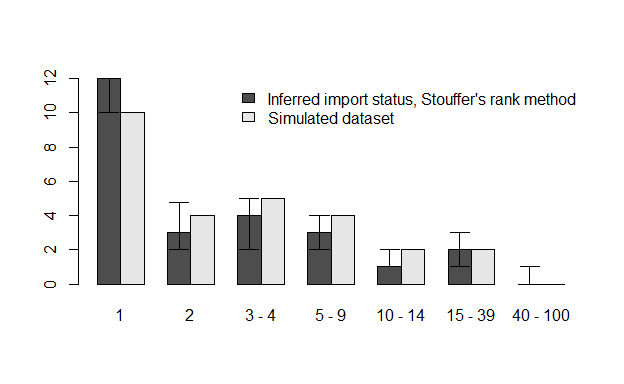


Figure 7: Comparison of inferred cluster size distribution with the reference data

Finally, we plot the proportion of perfect match and close match for each case (Figure 8). If we compare it with the first two models, we observe that the fit obtained with the Stouffer’s rank method is consistently worse than the first two models. The Stouffer’s rank method did not improve the agreement between the fitted and reference data.

The reference data used in the study were simulated using an exponential gravity model, which explains why introducing the Stouffer’s rank method did not improve the inference. This is not representative of the performance of each mobility model on real-world data.

index\_infer\_stouf <- index\_infer(dt\_cases = dt\_cases, out = out\_stouffer,   
 burnin = config1$burnin)  
index\_clust\_stouf <- index\_clust(dt\_cases = dt\_cases, out = out\_stouffer,   
 burnin = config1$burnin)  
# Plot the sorted proportion in each model  
oldpar <- par(no.readonly =TRUE)  
par(bty = "n", mfrow = c(1, 2), mar = c(5,4,2,0), oma = c(0, 0, 0, 0))  
# Panel A  
plot(sort(index\_infer\_stouf), type = "l", xlab = "Case", ylab = "",   
 main = "A", las=1, col = grey.colors(2)[1], lwd = 3)  
# Panel B  
plot(sort(index\_clust\_stouf), type = "l", ylab = "Proportion", xlab = "Case",   
 main = "B", las=1, col = grey.colors(2)[1], lwd = 3)

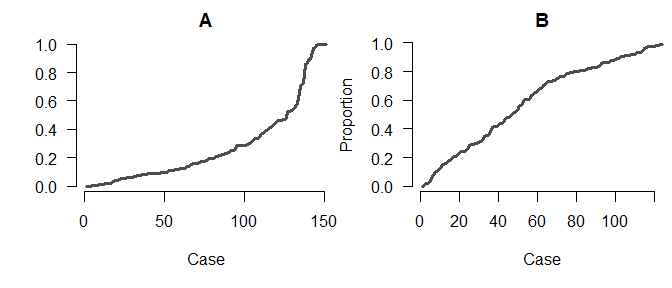


Figure 8: Panel A: Proportion of iterations with the correct index for each case; Panel B: Proportion of iterations where the index is from the correct cluster

par(oldpar)

# Conclusion

The R package *o2geosocial* is a new tool for data analysis building upon the framework developed in [*outbreaker2*](https://CRAN.R-project.org/package=outbreaker2). It is highly flexible and only requires routinely collected surveillance data. It can be used to identify large transmission clusters and highlight the dynamics of transmission in different regions. An application of the algorithm on a small geographical scale was presented in the tutorial, it can also be used to study datasets of cases scattered across larger areas. The methods presented in the tutorial can be applied to the second dataset included in the package toy\_outbreak\_long, which includes more than 1900 cases simulated in the United States. The computation time increases with the number of regions and the number of cases, so the MCMC runs will be longer.

We showed how the model could be edited to implement a range of mobility models. Describing human mobility during infectious diseases outbreaks can be challenging, and the performances of the models depend on the setting studied. The package can be extended to take into account the variety of existing mobility models. We encourage the development of extensions and the use of *o2geosocial* to study different pathogens and settings. We believe a wide use of *o2geosocial* could maximise what can be reconstructed from routinely collected data and epidemiological investigations and improve out understanding of outbreak dynamics.