Supplemental Digital Content: A cluster-based method to quantify individual heterogeneity in tuberculosis transmission

Jonathan Smith, PhD, Neel Gandhi, MD, Benjamin Silk, PhD, Theodore Cohen, MD, Dr.PH, Benjamin Lopman, PhD, Kala Raz, MPH, Kathryn Winglee, MPH, Steve Kammerer, MBA, David Benkesser, PhD, Michael Kramer, PhD, Andrew Hill, PhD

November 15, 2021

Contents

1 Branching Processes

1.1 General overview of branching processes

Branching processes are stochastic, individual-based processes commonly used in epidemiology to model disease transmission when depletion of susceptible individuals by infection is negligible, such as in the early stages of an epidemic or when infectious diseases are introduced in a non-endemic setting. While branching processes represent a range of model types, this analysis models transmission using a single-type branching process, also known as a discrete time Galton-Watson process. This is the most well-studied and validated approach to branching processes. Under this approach, each infected individual is associated with a fixed length time interval known as a generation; at the end of each generation, each individual in the generation will have produced a random number of secondary infections (or "offspring"), denoted Z, drawn from some probability distribution.

1.2 Analysis of branching process models

The offspring distribution is the probability distribution for the observed number of secondary cases caused by each individual infectious case, Z (i.e. $p_z = P(Z = z)$ for $z = 0, 1, 2, 3, \dots$). A fundamental tool in the analysis of any branching process model is the probability generating function (pgf) of the offspring distribution. In a branching process framework, the pgf is a mathematical tool to study the sequence of probabilities and contains all the information needed to recover the probabilities associated with each Z value. The pgf in branching processes can generally be expressed as $G_Z(s) = \sum_{z=0}^{\infty} p_z s^z$, where s, is a dummy variable with no tangible meaning yet whose powers serve as a placeholder to recover the probabilities associated with Z and facilitate the use of high-order derivatives in their recovery.

Probability generating functions in branching processes

Secondary cases (z value)	0	1	2	3	...	z
Coefficient (s^z)	s^0	s^1	s^2	s^3	...	s^z
$P(Z = z) = p_z$	p_0	p_1	p_2	p_3	...	p_z

$$G_Z(s) = p_0 + p_1 s + p_2 s^2 + p_3 s^3 + \cdots + p_z s^z + \cdots = \sum_{z=0}^{\infty} p_z s^z$$

Where s is a dummy variable whose powers serve to recover the p_z probabilities in a power series expansion of the probability generating function.

2 Extracting the Probability from the Generating Function

Of primary interest in branching process models is the probability that an infectious individual in a given generation transmits Z secondary cases, $p_z = P(Z = z)$. As shown above, this probability is embedded in the generating function of Z. To extract this probability, we exploit a well-known pgf property related to distributional moments. Given the pgf $G_Z(s) = \sum_{z=0}^{\infty} p_z s^z$, the coefficient of s^z in its expansion for any z can be extracted by the following three steps:

1. Taking the z^{th} derivative of $G_Z(s)$, or $\frac{d^Z G_Z(s)}{ds^z}$
2. Evaluating $\frac{d^Z G_Z(s)}{ds^z}$ at $s = 0$
3. Normalizing the expression by the constant, $1/z!$

For a walk-through proof of this process, we can begin with the probability that an individual results in a single secondary case (i.e. p_1 since $z = 1$). After taking the 1^{st} derivative, all other terms in the polynomial except the p_1 term contain s, thus

evaluating the expression at $s = 0$ extracts p_1 (note that $1! = 1$):

$$\begin{aligned} G_Z(s) &= p_0 + p_1 s + p_2 s^2 + p_3 s^3 + \cdots + p_z s^z + \cdots \\ G'_Z(s) &= p_1 + 2p_2 s + 3p_3 s^2 + \cdots + z p_z s^{z-1} + \cdots \\ G'_Z(0) &= p_1 \end{aligned}$$

To evaluate the probability that an individual infectious case results in two secondary cases, we similarly take the second derivative and evaluate at $s = 0$. In this case, we must also divide $G''_Z(0)$ by a constant (in this case, $2! = 2$) to properly extract p_2:

$$\begin{aligned} G''_Z(s) &= (2 \times 1)p_2 + (3 \times 2)p_3 s + \cdots + z(z-1)p_z s^{z-2} + \cdots \\ G''_Z(0) &= 2p_2 \\ \frac{1}{2} G''_Z(0) &= p_2 \end{aligned}$$

To evaluate the probability that an individual case results in three secondary cases, we similarly take the third derivative, evaluate at $s = 0$, and divide by $(3 \times 2) = 3!$.

$$\begin{aligned} G'''_Z(s) &= (3 \times 2 \times 1)p_3 + (4 \times 3 \times 2)p_4 s + \cdots + z(z-1)(z-2)p_z s^{z-3} + \cdots \\ G'''_Z(0) &= (3!)p_3 \\ \frac{1}{3!} G'''_Z(0) &= p_3 \end{aligned}$$

To generalize this, if we would like to extract the probability that an infectious case generates any number of z secondary cases, $P(Z = z)$, we take the z^{th} derivative, evaluate at 0, and divide by the appropriate constant:

$$p_z = \frac{1}{z!} \frac{d^z G_Z(s)}{ds^z} \bigg|_{s=0}$$

Where $z = 0, 1, 2, 3, ...$

We expand this concept to the multiplication of generating functions, a commonly used manipulation of branching process theory. For example, the z^{th} coefficient of multiplying three generating functions of Z, $[G_Z(s)]^3$, provides the probability that three initial cases cause a total of z secondary infections; it follows that the z^{th}

coefficient of $G_Z(s)^i$ specifies the probability that i cases result in z secondary cases, thus:

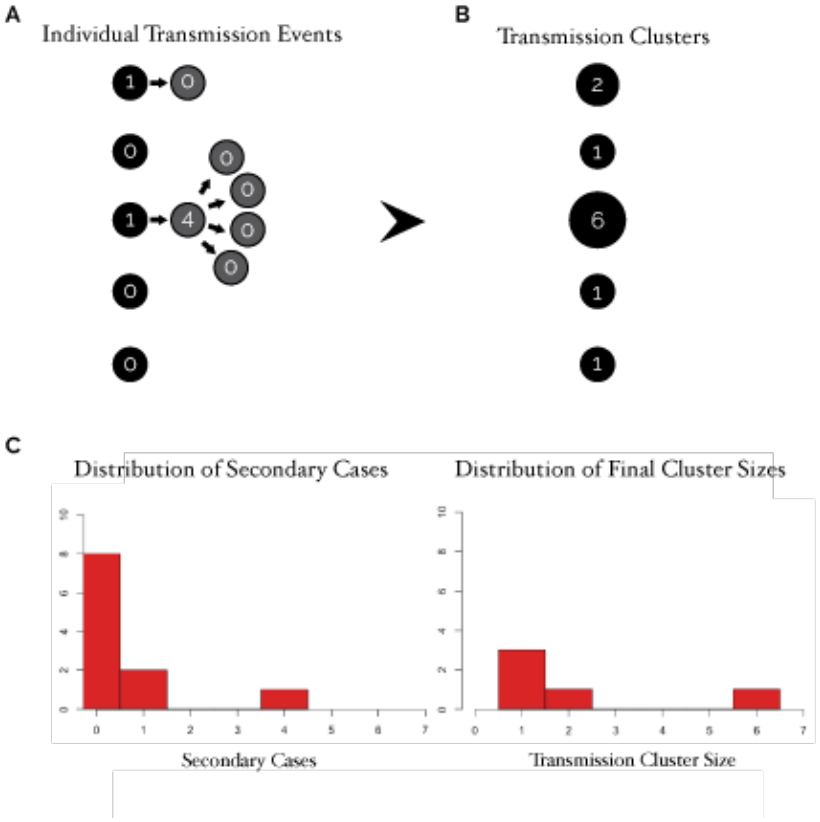
$$P(Z = z|i) = \frac{1}{z!} \frac{d^z G_Z(s)^i}{ds^z}\bigg|_{s=0}$$

where $z = 0, 1, 2, 3, ...$ and $i = 1, 2, 3...$ $(i \leq z)$.

3 Extracting the Probability of Transmission Cluster Sizes

The above methods apply to individual transmission events, which are rarely observed in practice. However, obtaining cases in the same transmission cluster (e.g., cases in the same chain, regardless of transmission pattern) is more plausible. We extend the above principles of branching process theory to mathematically relate the distribution of individual secondary cases to the distribution of final cluster sizes.

Visualizing individual- and cluster-level data. *In this example, 11 individual cases in a population are represented in 5 transmission chains. (A) Individual data allow for the reconstruction of transmission trees; integers represent the number of secondary cases from each case for individual-level data (Z values) (B) Transmission clusters do not contain information on transmission patterns. Integers in cluster level data are the sum of all cases in a transmission chain. (C) Distributions of the number of secondary cases and transmission cluster sizes, respectively.*



3.1 Recurrence relationship for $G_Y(s)$

In branching process theory, each individual in a given generation causes a set of independent and identically distributed (i.i.d.) secondary offspring infections also defined by $G_Z(s)$. The process continues until all individuals in a generation, g, produce zero secondary cases (e.g., $Z_{g-1} > 0$ and $Z_g = 0$), at which point the chain of transmission becomes extinct. Let Y represent sum of all cases in a transmission chain (including the index case), referred to as a transmission cluster. The primary focus of this analysis is to relate the offspring distribution of individual secondary cases, Z, and the offspring distribution of final cluster sizes, Y. The distribution of the total number of secondary infections in a single transmission chain (not including the index case at generation 0) can be defined by the pgf $G_Z(G_Y(s))$. In order to include the initial index case, we must further increase the exponent of s by one, accomplished simply by multiplying the expression by s, yielding the implicit relationship: [5]:

$$G_Y(s) = sG_Z((G_Y(s))$$

Following the principles outlined in Section 2, we can treat this as $G_Y(s) = \sum_{y=1}^{\infty} q_y s^y$, where $q_y = P(Y = y)$, allowing us to extract the probability: [8]:

$$P(Y = y) = \frac{1}{y!} \frac{d^y G_Y(s)}{ds^y}\bigg|_{s=0}$$

where $Y = 1, 2, 3, \ldots$

The solution to $G_Y(s) = s(G_Z(G_Y(s))$ may sometimes be solved recursively with several helpful conventions. Since there must always be at least one case to initiate the transmission chain (e.g., a cluster cannot be of size 0), $P(Y = 0) = 0$ and Y must be a positive nonzero integer. Furthermore, a cluster of size one, $Y = 1$ represents the probability that a single index case transmits zero secondary cases; noting that $G'_Y(0) = G_Z(0)$ therefore $P(Y = 1) = P(Z = 0)$. We can extend this to $Y = 2$ to demonstrate the recursive procedure and the continued relationship between the generating functions of the total cluster size and individual secondary cases. When $Y = 2$, the only permutation available is that the index case transmitted to one other person who subsequently did not transmit to anyone:

$$\begin{aligned} G''_Y(0) =& 2G'_Z(G_Y(0))G'_Y(0) \\ =& 2G'_Z(0)G_Z(0) \end{aligned}$$

Thus:

$$P(Y=2) = \frac{1}{2}G''_Y(0) = P(Z=1)P(Z=0)$$

When $Y=3$, two permutations are available. The index case may cause two secondary cases and neither subsequently transmit, or the index case results in one case who subsequently results in another case that does not transmit, thus:

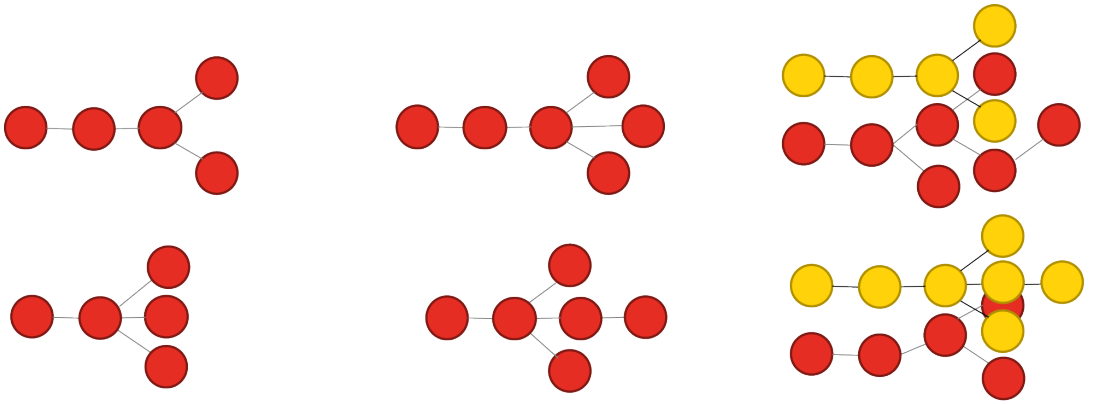
$$\begin{aligned} P(Y=3) =& \frac{1}{6}G_Y^{(3)}(0) \\ =& \frac{1}{6}\left(3G''_Z(G_Y(0))[G'_Y(0)]^2 + 3G'_Z(G_Y(0))G''_Y(0)\right) \\ =& \frac{1}{2}G''_Z(0)(G_Z(0))^2 + ((G'_Z(0))^2(G_Z(0)) \\ =& P(Z=2)P(Z=0)^2 + P(Z=1)^2P(Z=0) \end{aligned}$$

This relationship continues for all $Y = 1, 2, 3, \ldots$.

3.2 Constraints when considering transmission chains in infectious disease

Recall that p_z in $G_Z(s) = \sum_{z=0}^{\infty} p_z s^z$ specifies the probability that a given case will infect z secondary cases and that the z^{th} coefficient of $G_Z(s)^y$ specifies the probability that y cases result in z secondary cases over the course of a single generation. While this is a useful starting point, there are several important constraints imposed when considering biologically valid chains of transmission. First, for every cluster initiating with n index cases and which collectively go extinct at size y, there must be exactly $y - n$ transmission events, regardless of the sequence of transmission.

Visualizing the $y - n$ ***constraint*** *For every cluster of size* y *originating with* n *index cases, there are exactly* $y - n$ *transmission events. Here, nodes represent infected individuals and arcs represent transmission events.*



Cluster size (y): 5
Index cases (n): 1
Transmission events (y-n): 4

Cluster size (y): 6
Index cases (n): 1
Transmission events (y-n): 5

Cluster size (y): 12
Index cases (n): 2
Transmission events (y-n): 10

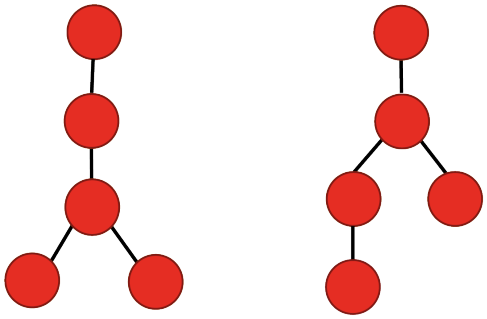
This constraint is consequential in that if we randomly draw a sequence of n integers from the offspring distribution (representing secondary cases for each individual in the cluster, and including the possibility of zero secondary cases), the probability that they sum to $y - n$ is the coefficient of s^{y-n} in $G_Z(s)^y$ normalized by $1/(y-n)!$ and evaluated at $s = 0$, or:

$$\frac{1}{(y-n)!}\frac{d^{y-n}G_Z(s)^y}{ds^{y-n}}\bigg|_{s=0}$$

Second, for a given cluster size y originating with n index cases, even if the sum of secondary cases comes to exactly $y - n$, not all of these permutations will result in valid transmission chains. This is because when considering true chains of infectious disease transmission, the order of events must be biologically relevant (e.g., the order of infections matters). As a result, only a certain proportion of these sequences, which is shown to be n/y, will result in a valid chain of transmission.[2, 1, 6]

To demonstrate this concept, it is helpful to draw on notation from Stanley *et al.*[7] We denote infectious cases in a given transmission chain as an integer corresponding to the number of secondary cases, and order these integers such that we enumerate the number of secondary cases starting from the top and moving left to right:

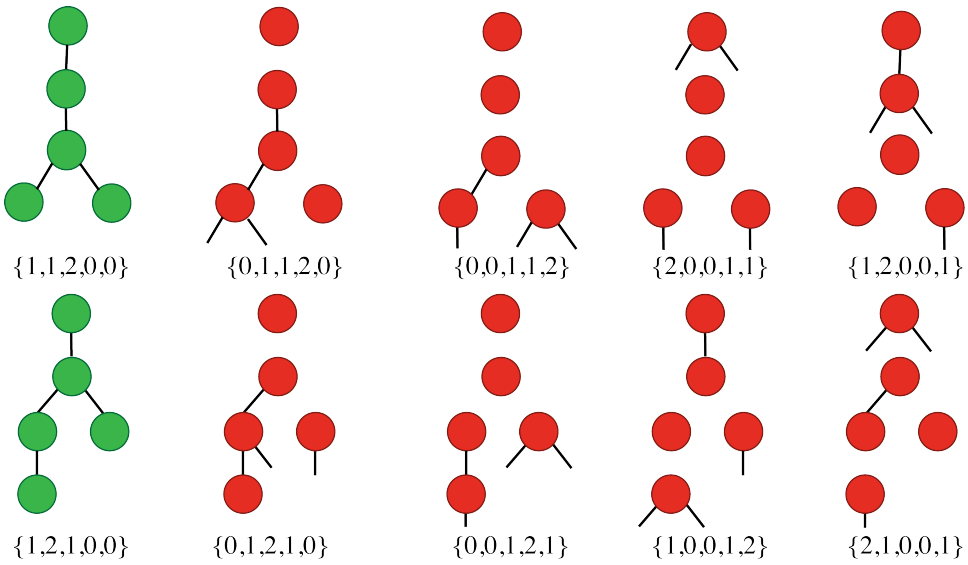
Visualizing the nomenclature of possible transmission trees. *This figure demonstrates the nomenclature of two valid transmission trees originating with a single index case and resulting in a final cluster size of 5. Each position in the sequence represents a case and the integer represents the number of offspring for that case. Sequences are ordered from top to bottom, left to right.*



Using this notation, one can easily verify the constraint of requiring exactly $y - n$ by summing the integers in the sequence and noting the sequence length (e.g., in the above figure, both have length 5 and sum to 4). It is also straightforward to see that for every valid transmission tree of size y, there are y cyclic permutations of that sequence that also sum to $y - n$. Put in other words, one can shift the integers one position over y times before arriving at the original sequence. As shown below, if we define $\mathscr{L}$ as a sequence of length y summing to $y - n$, $\mathscr{L} = \{x_1 x_2 \dots x_y\}$, only n of the y cyclic permutations of $\mathscr{L}$ result in a valid transmission sequence. Given the i.i.d. assumption of our branching process model, each of these permutations are equally likely to be observed. Thus, n/y of these permutations result in a valid transmission sequence.

To help conceptualize this, the below figure provides a visual example of this constraint for a simple cluster of size $Y = 5$ originating with $n = 1$ index case. Two valid $\mathscr{L}$ sequences are shown, e.g., $Y - n = 5 - 1 = 4$ (or using the above notation, $\sum_{i=1}^{5} = x_i = 4$).

Visualizing the cyclic permutations of valid transmission sequences. *This figure demonstrates the cyclic permutations of two valid transmission trees originating with a single index case and resulting in a final cluster size of 5. Green indicates a valid transmission sequence, and red indicates an invalid transmission sequence.*



While all cyclic permutations in the figure sum to $Y - n = 5 - 1 = 4$, only one of the five cyclic permutations associated with the sequence results in a biologically valid transmission chain; given that each chain is equally likely to be observed, the probability of observing the valid chain is $n/y = 1/5$.

As a result of this second constraint, the probability of a transmission cluster originating with n index cases and collectively going extinct at exactly size y must be normalized by a factor of (n/y), thus:

$$P(Y = y|n) = \left(\frac{n}{y}\right) \frac{1}{(y-n)!} \frac{d^{y-n} G_Z(s)^y}{ds^{y-n}}\bigg|_{s=0}$$

4 The Negative Binomial Distribution

Until this point we have not specified a distribution for $G_Z(s)$. Our model assumes $G_Z(s)$ is negative binomially distributed with mean R (the basic reproductive number) and dispersion parameter k. The negative binomial distribution is a two-parameter distribution popular in biology and epidemiology where the distribution

is highly overdispersed (e.g., higher than expected variation in the distribution). Unlike single parameter distributions popular in epidemiology, such as the Poisson and geometric distributions, the free parameter k allows for an unknown degree of heterogeneity and quantifies the degree of overdispersion in the distribution.

In general probability theory, the negative binomial distribution typically expresses the number of successes (p) in a sequence of i.i.d. Bernoulli trials before a specified number of failures occurs (k). This has been previously reparameterized for infectious disease by noting that the probability of success can be expressed as $p = R/(R+k)$ [4] thus:

$$\begin{aligned} P(Z=z) =& \binom{z+k-1}{k-1}(1-p)^k p^z \\ =& \binom{z+k-1}{k-1}\left(\frac{k}{R+k}\right)^k\left(\frac{R}{R+k}\right)^z \\ =& \frac{\Gamma(z+k)}{\Gamma(z)\Gamma(k+1)}\left(\frac{k}{R+k}\right)^k\left(\frac{R}{R+k}\right)^z \end{aligned}$$

where $z = 0, 1, 2, \ldots$ and $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}$. The pgf of the negative binomial distribution is [3]:

$$G_Z^{NB}(s) = \left(1 + \frac{R}{k}(1-s)\right)^{-k}$$

Reverting to $R/(R+k) = p$ for clarity, we can write the pgf of the negative binomial as:

$$\begin{aligned} G_Z^{NB}(s) =& \left(\frac{1-p}{1-ps}\right)^k \\ =& (1-p)^k(1-ps)^{-k} \end{aligned}$$

Recall the multiplication of y generating functions yields the generating function $G_Z(s)^y = (1-p)^{ky}(1-ps)^{-ky}$. We can use Taylor expansion of $G_Z(s)^y$ to help derive the final cluster size distribution. Recall Taylor series expansion for a function

$f(x) = (1+x)^\alpha$ can be generally expressed as:

$$\begin{aligned}(1+x)^\alpha =& \sum_{i=0}^{\infty} \binom{\alpha}{i} x^i \\ =& 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!}x^2 + \cdots + \frac{\alpha(\alpha-1)\cdots(\alpha-i+1)}{i!}x^i + \cdots\end{aligned}$$

Letting $x = -ps, \alpha = -ky, i = y - n$ and $p = R/(R+k)$, substituting into the generating function $G_Z^{NB}(s) = (1-p)^k(1-ps)^{-k}$ yields:

$$\begin{aligned}G_Z^{NB}(s)^y =& (1-p)^{ky}(1-ps)^{-ky} \\ =& (1-p)^{ky}\Bigg(1 + kyps + \frac{ky(ky+1)}{2!}p^2s^2 + \cdots \\ & + \frac{ky(ky+1)\cdots(ky+(y-n)-1)}{(y-n)!}p^{y-n}s^{y-n} + \cdots\Bigg)\end{aligned}$$

This results in the coefficient of s^{y-n} evaluated at $s = 0$ as:

$$\begin{aligned}\frac{1}{(y-n)!}\frac{d^{y-n}G_Z^{NB}(s)^y}{ds^{y-n}}\bigg|_{s=0} =& \frac{(ky+y-n-1)!}{(ky-1)!(y-n)!}(1-p)^{ky}p^{y-n} \\ =& \binom{ky+y-n-1}{y-n}(1-p)^{ky}p^{y-n}\end{aligned}$$

This only holds when k is an integer, though we can extend this to any real number k by noting that:

$$\frac{ky(ky+1)+\cdots+(ky+y-n-1)}{(y-n)!} = \frac{\Gamma(ky+y-n)}{\Gamma(ky)(y-n)!}$$

Substituting $p = R/(R+k)$ with algebraic manipulation and recalling that only an n/y proportion of the coefficient of s^{y-n} result in valid transmission sequences, we can express the final probability of a cluster initiating with n index cases and going extinct at exactly y cases as:

$$P(Y = y|n) = \binom{n}{y}\frac{\Gamma(ky+y-n)}{\Gamma(ky)(y-n)!}\frac{(\frac{R}{k})^{y-n}}{(1+\frac{R}{k})^{ky+y-n}}$$

References

[1] Niels Becker. On parametric estimation for mortal branching processes. *Biometrika*, 61(3):393–399, 1974.

[2] Meyer Dwass. The total progeny in a branching process and a related random walk. *Journal of Applied Probability*, 6(3):682–686, 1969.

[3] J. O. Lloyd-Smith, S. J. Schreiber, P. E. Kopp, and W. M. Getz. Superspreading and the effect of individual variation on disease emergence. *Nature*, 438(7066):355–359, November 2005.

[4] James O. Lloyd-Smith. Maximum likelihood estimation of the negative binomial dispersion parameter for highly overdispersed data, with applications to infectious diseases. *PLOS ONE*, 2(2):1–8, 02 2007.

[5] Charles J Mode and Candace K Sleeman. *Stochastic Processes in Epidemiology*. World Scientific, 2000.

[6] Blumberg S and Lloyd-Smith JO. Inference of r0 and transmission heterogeneity from the size distribution of stuttering chains. *PLoS Comput Biol.*, 5(9):393–399, 2013.

[7] Richard P. Stanley and Sergey Fomin. *Enumerative Combinatorics*, volume 2 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1999.

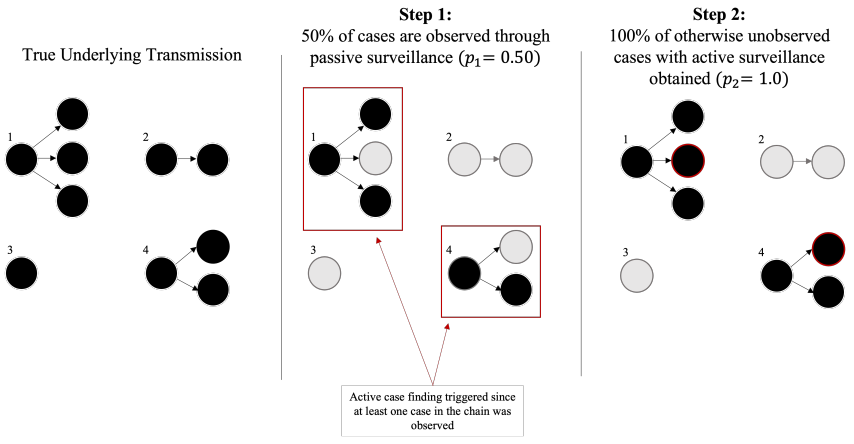
[8] Ping Yan. *Distribution Theory, Stochastic Processes and Infectious Disease Modelling*, pages 229–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

5 Simulating Imperfect Surveillance

5.1 Missing Cases

The mechanism in which a case is included in the surveillance system is important to the distribution of cluster sizes in a surveillance system. As described in the main text, we simulate both passive and active surveillance. In particular, active surveillance (i.e., contact tracing) is likely differential by cluster size: public health programs typically initiate active case finding measures only after an initial case or outbreak is observed.To account for this in evaluating the cluster-based method of parameter inference, we simulated missing cases in a two step process as described in the primary text. Briefly, we assumed each cases was observed via passive surveillance with p_1 (where $p_1 = 1.0$ implies perfect surveillance). After evaluation with p_1, active case finding was triggered in chains with at least one other case identified via passive surveillance. In chains undergoing active case finding, otherwise unobserved cases were observed with probability p_2. A visual representation of this is below:

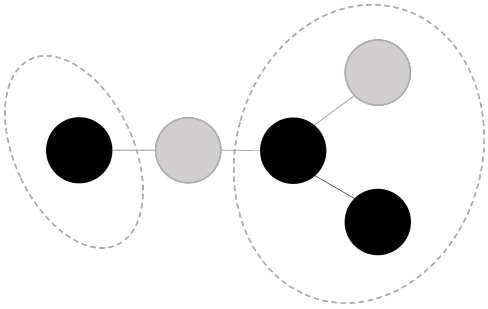
Simulating imperfect surveillance To simulate the mechanisms of case ascertainment, each case in the true underlying transmission chain is first observed with probability p_1. In this example, $p_1 = 0.50$ and so each case has a 50% probability of being observed. Second, any unobserved case within a chain where at least one other case observed by passive surveillance (chains 1 and 4) is re-evaluated for observation with probability p_2 to emulate active case finding measures. Here, $p_2 = 1.0$ and so all cases in chains where active case finding was triggered are observed. Wholly unobserved chains (chains 2 and 3) are not subject to active case finding measures.



5.2 Broken chains

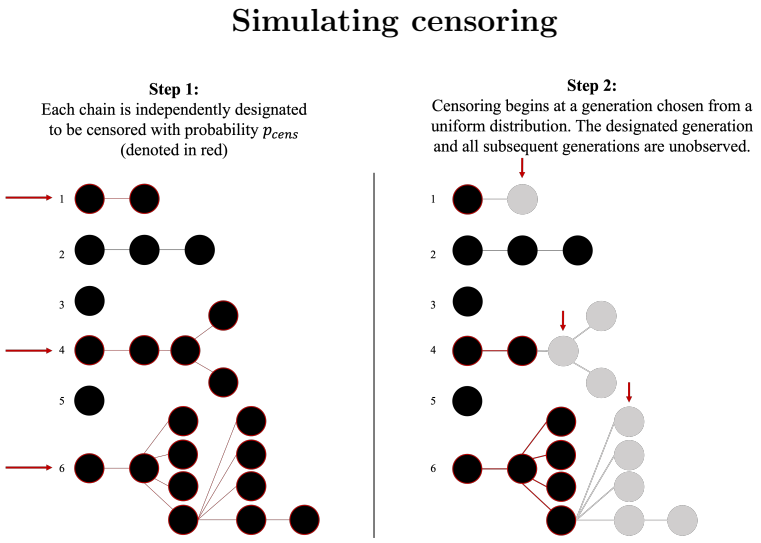
As described in the primary text, after evaluation of p_1 and p_2, we further considered the position of the missing case in the chain of transmission. If the missing case was the sole link between the previous and future generations of spread, the chain was "broken" into j psuedo-clusters (which themselves could contain unobserved cases).

Broken chains In this example, a transmission chain of size $Y = 5$ with 2 missing cases ans is broken into two psuedo-clusters of sizes $y_1 = 1$ and $y_2 = 2$ (dotted outlines) given the position of the missing cases.



5.3 Censoring

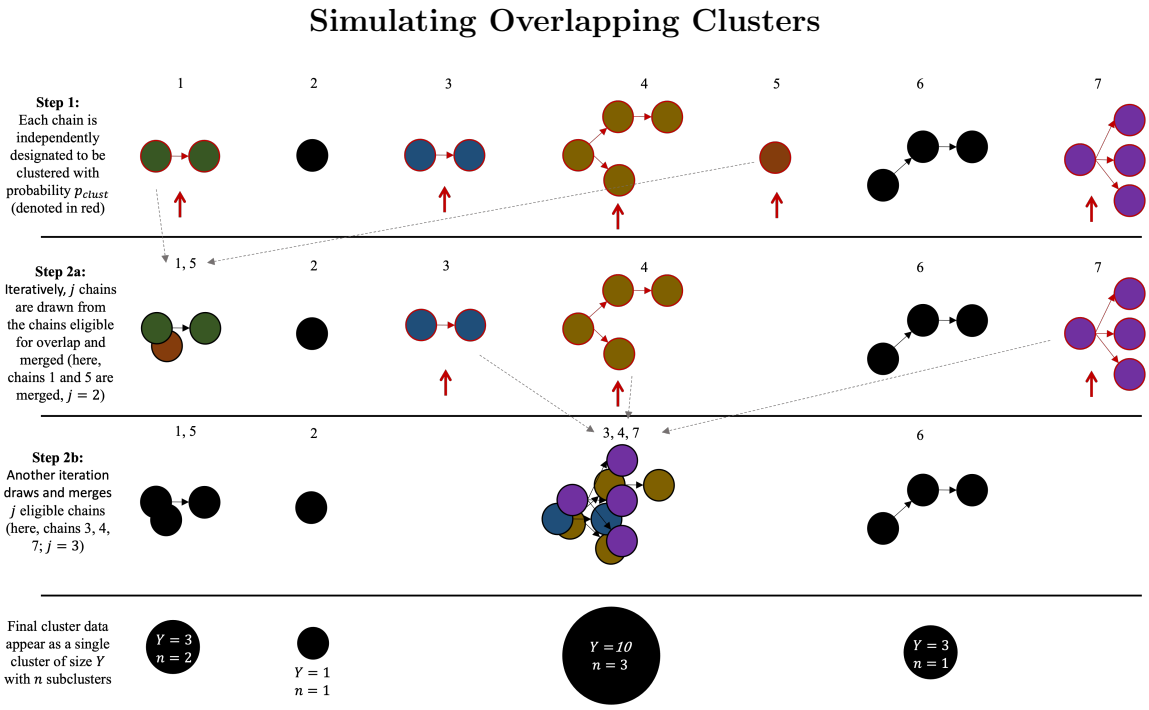
Censoring is a consequence of ongoing chains at the time of data collection and is an intrinsic property of surveillance data. We approached censoring as a two step process. First, each chain was designated as censored with probability p_{cens}. Second, among chains designated as censored, the generation of spread where censoring began was drawn from a uniform distribution (e.g., all generations had an equal probably of being the censored generation), with the exception of the first generation which was not subject to censoring. All cases in the designated generation and all subsequent generations were unobserved. A visual representation of the censoring scheme is below.



5.4 Overlapping chains (subclustering)

The final limitation in assessed in this analysis that may play a major role in the distribution of cluster sizes in a surveillance system is the inability to cleanly and unambiguously tease apart multiple chains of transmission. For instance, it has been shown through whole genome sequencing in tuberculosis surveillance that MIRU-VNTR genotypic clusters often contain multiple transmission subclusters. This may shift the distribution of cluster sizes to the right, and make transmission appear more homogenous.

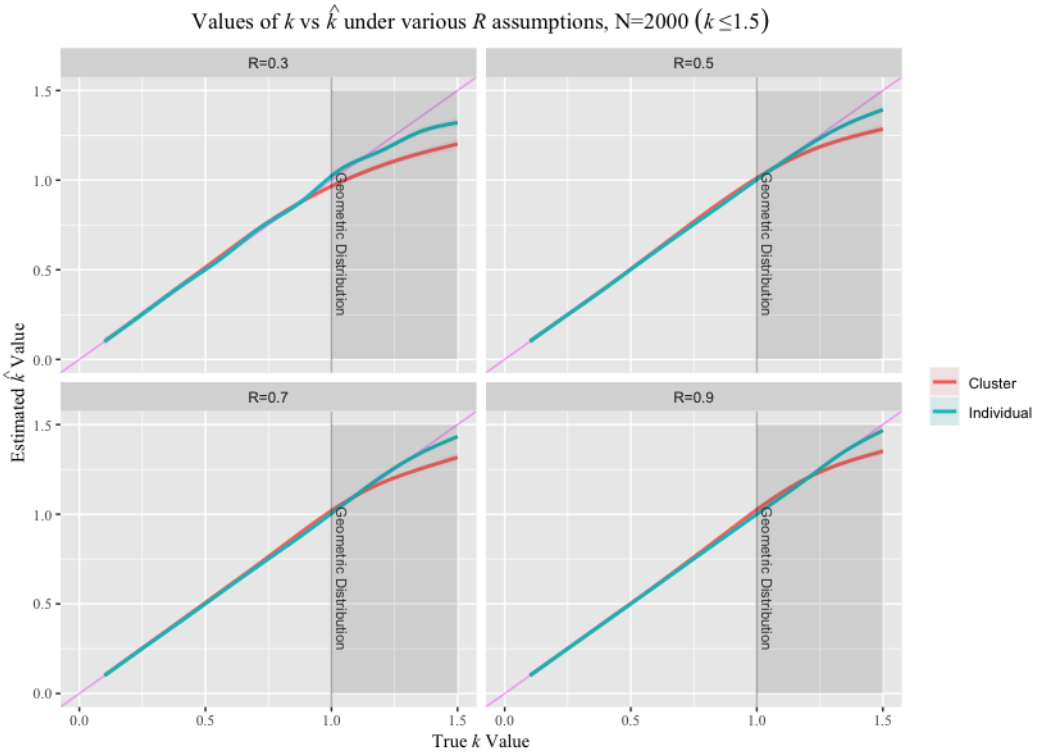
As described in the main text, simulating overlap was an two-step, iterative process. First all chains were designated to overlap or not with probability p_{cens} (e.g., if $p_{cens} = 0.20$, 20 percent of chains in the true underlying surveillance system overlap with *at least* one other chain. Second, j chains were randomly drawn from the pool of chains designated for overlap and merged, resulting in a final cluster of size Y with n index cases. Merged clusters were then removed from the pool of eligible chains and the iterative process repeated until the pool of eligible clusters was exhausted. A visual representation of this process is below.


Simulating Overlapping Clusters
Step 1:
Each chain is independently designated to be clustered with probability p_{clust} (denoted in red)
1
2
3
4
5
6
7
Step 2a:
Iteratively, j chains are drawn from the chains eligible for overlap and merged (here, chains 1 and 5 are merged, $j = 2$)
1, 5
2
3
4
6
7
Step 2b:
Another iteration draws and merges j eligible chains (here, chains 3, 4, 7; $j = 3$)
1, 5
2
3, 4, 7
6
Final cluster data appear as a single cluster of size Y with n subclusters
$Y = 3$
$n = 2$
$Y = 1$
$n = 1$
$Y = 10$
$n = 3$
$Y = 3$
$n = 1$

6 Supplemental Figures and Tables from Main Text

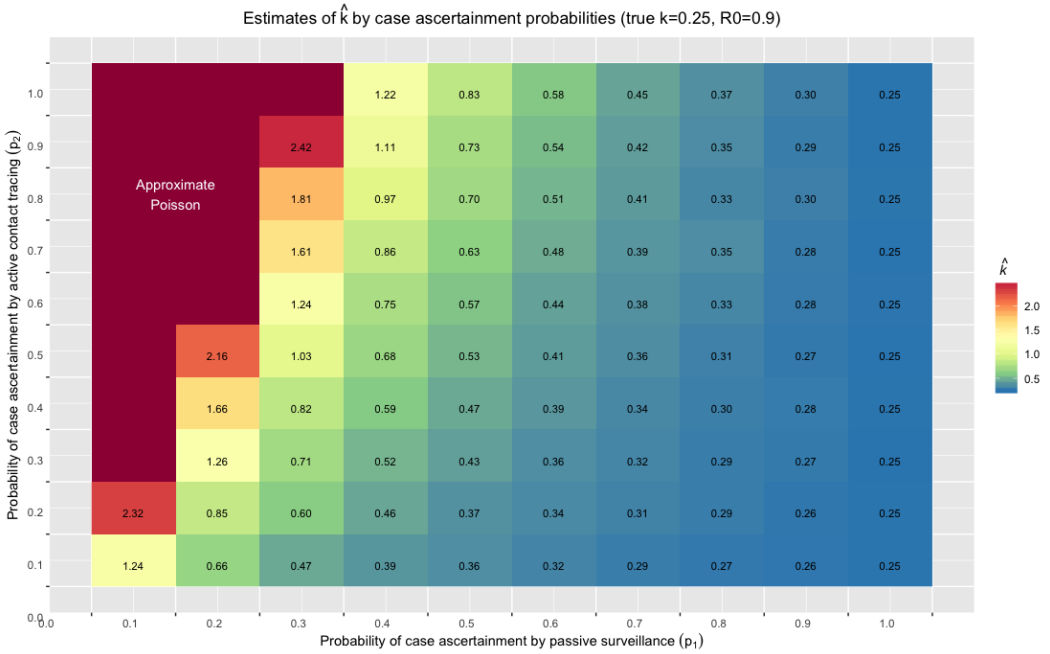
6.1 Supplemental Figures

eFigure 1: Inference of k under various R values for individual- and cluster-level data. Each surveillance system contained 2000 simulated transmission chains under perfect surveillance. Each surveillance system was simulated 100 times for underlying values of k between 0.1 and 1.5 according to the methods. The purple line indicates perfect inference. Values above the purple line indicate an overestimation of k; below the line indicate an underestimation of k. R values are indicated at the top of each panel ($R = 0.3, 0.5, 0.7$, and 0.9). This analysis focuses on k values below 1.0; the grey shaded areas represents k values above 1.0.

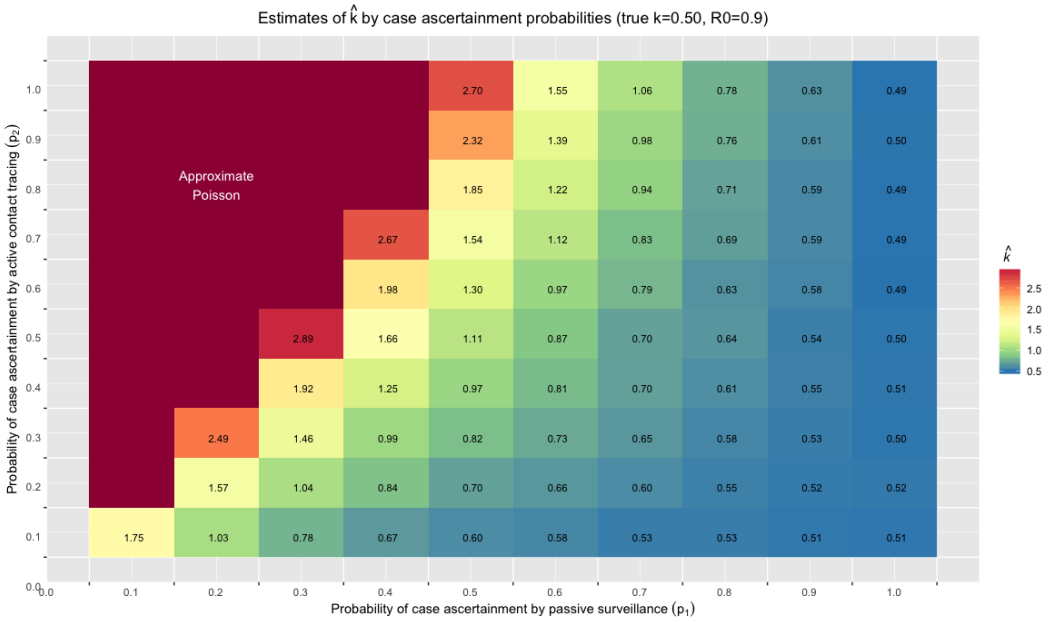


eFigure 2A: Estimates of k by case ascertainment probabilities under alternative R and k assumptions. This figure visualizes the bias of missing cases through passive and active surveillance modeled under alternative R and k values: (a) $R = 0.9, k = 0.25$; and (b) $R = 0.5, k = 0.25$. Numbers in the center of each combination of p_1 and p_2 represent the median of 1000 simulated surveillance systems, each originating with 2,000 chains.

(a) $k = 0.25, R = 0.90$

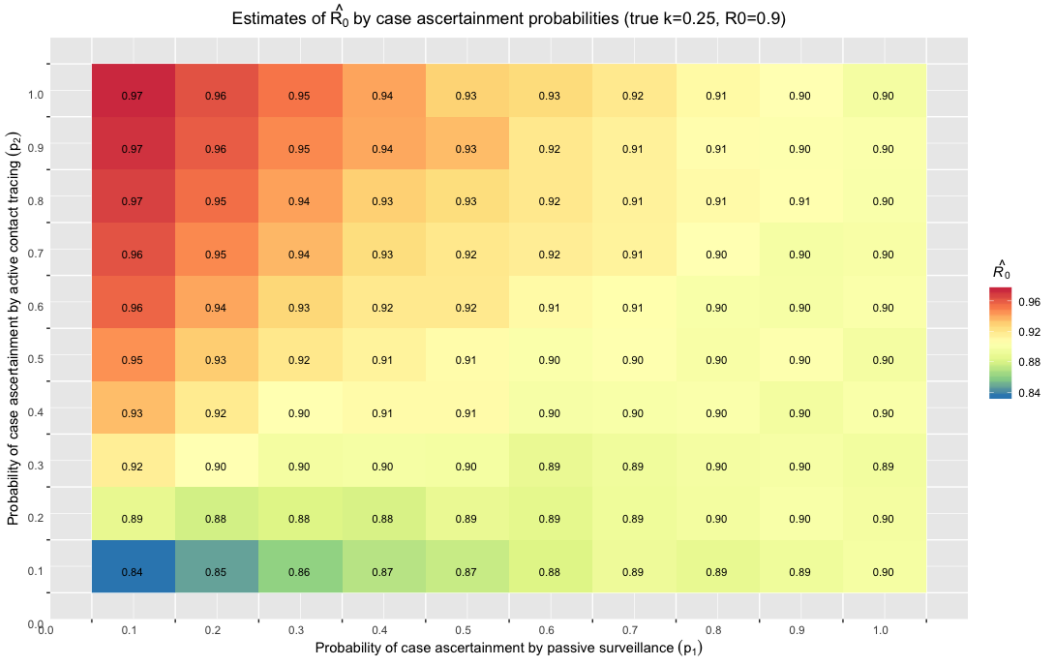


(b) $k = 0.50, R = 0.90$

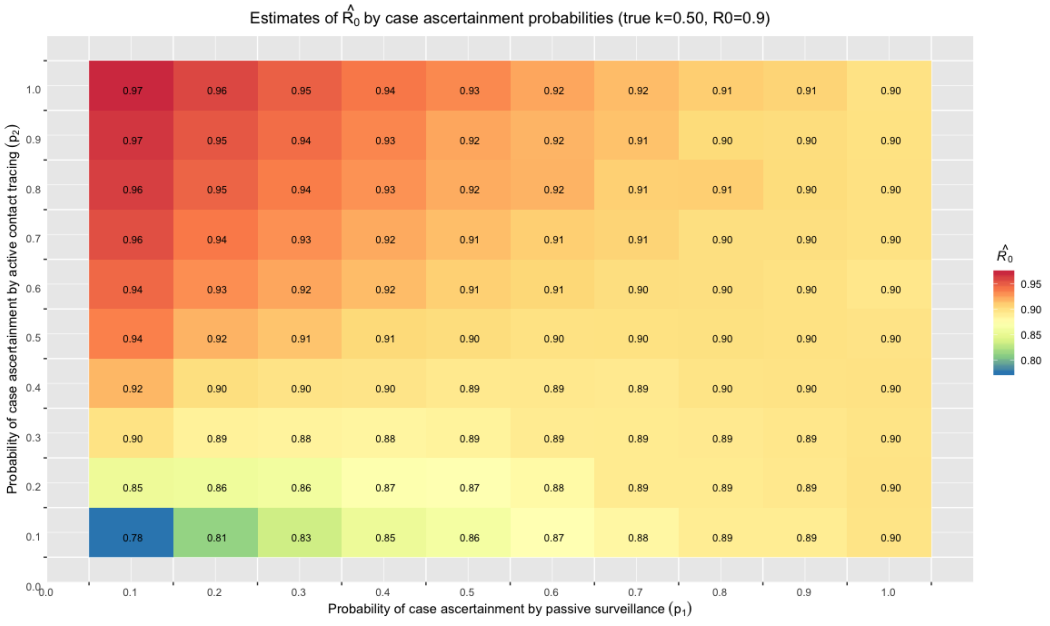


eFigure 2B: Estimates of R by case ascertainment probabilities under alternative R and k assumptions. This figure visualizes the bias of missing cases through passive and active surveillance modeled under alternative R and k values: (a) $R = 0.9, k = 0.25$; and (b) $R = 0.5, k = 0.25$. Numbers in the center of each combination of p_1 and p_2 represent the median of 1000 simulated surveillance systems, each originating with 2,000 chains.

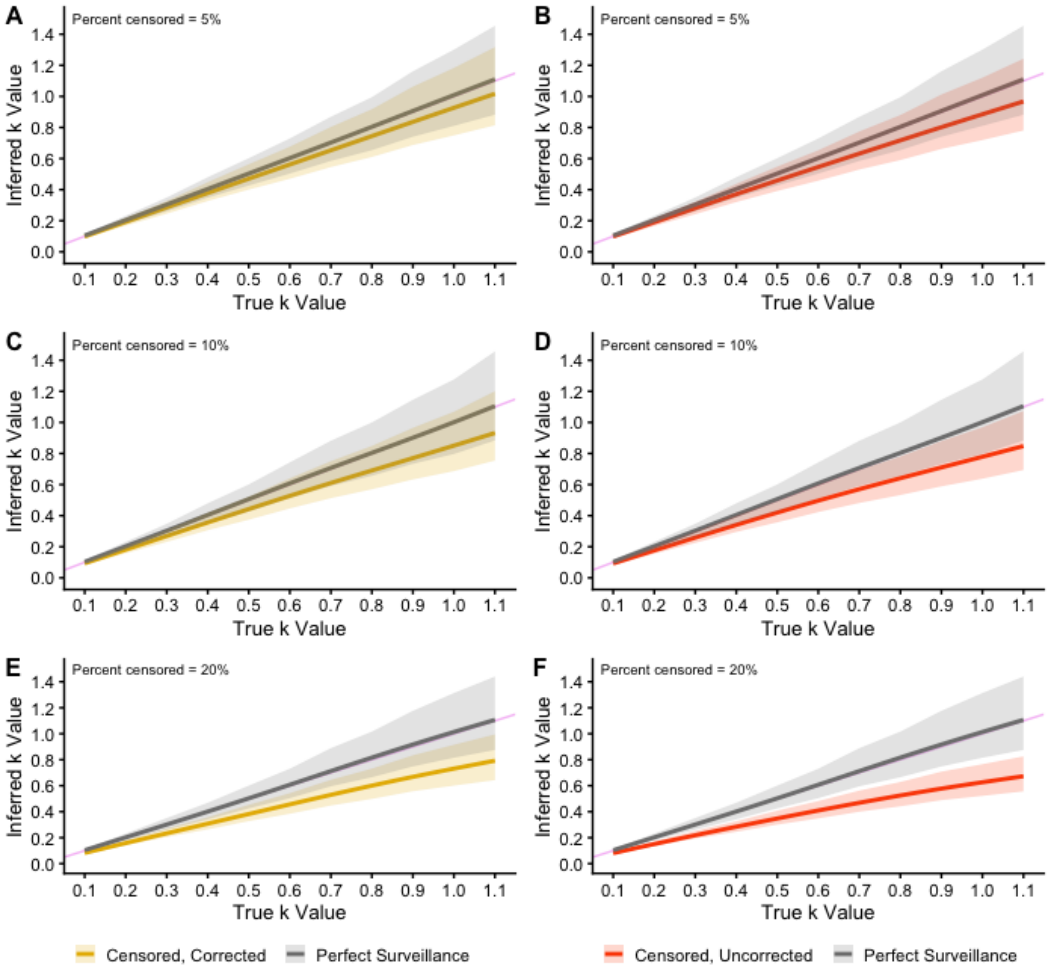
(a) $k = 0.25, R = 0.90$



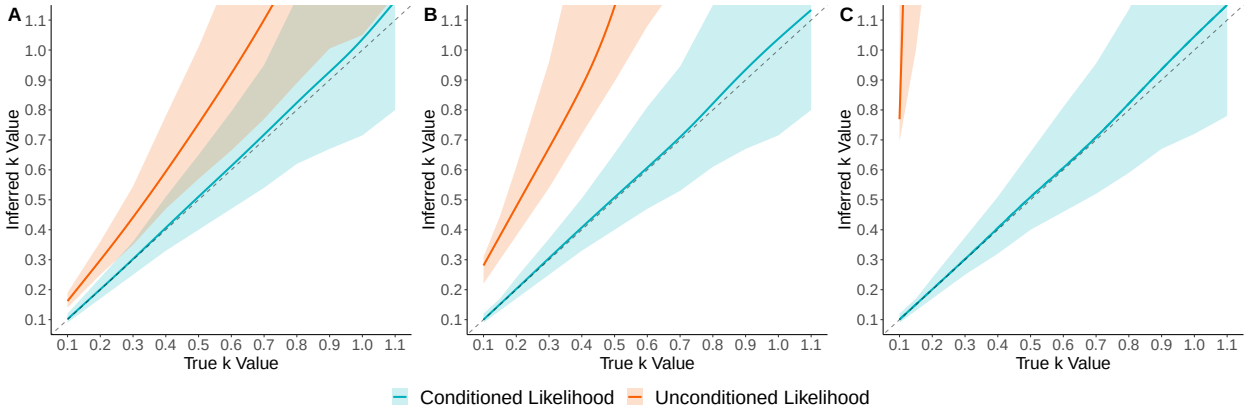
(b) $k = 0.50, R = 0.90$



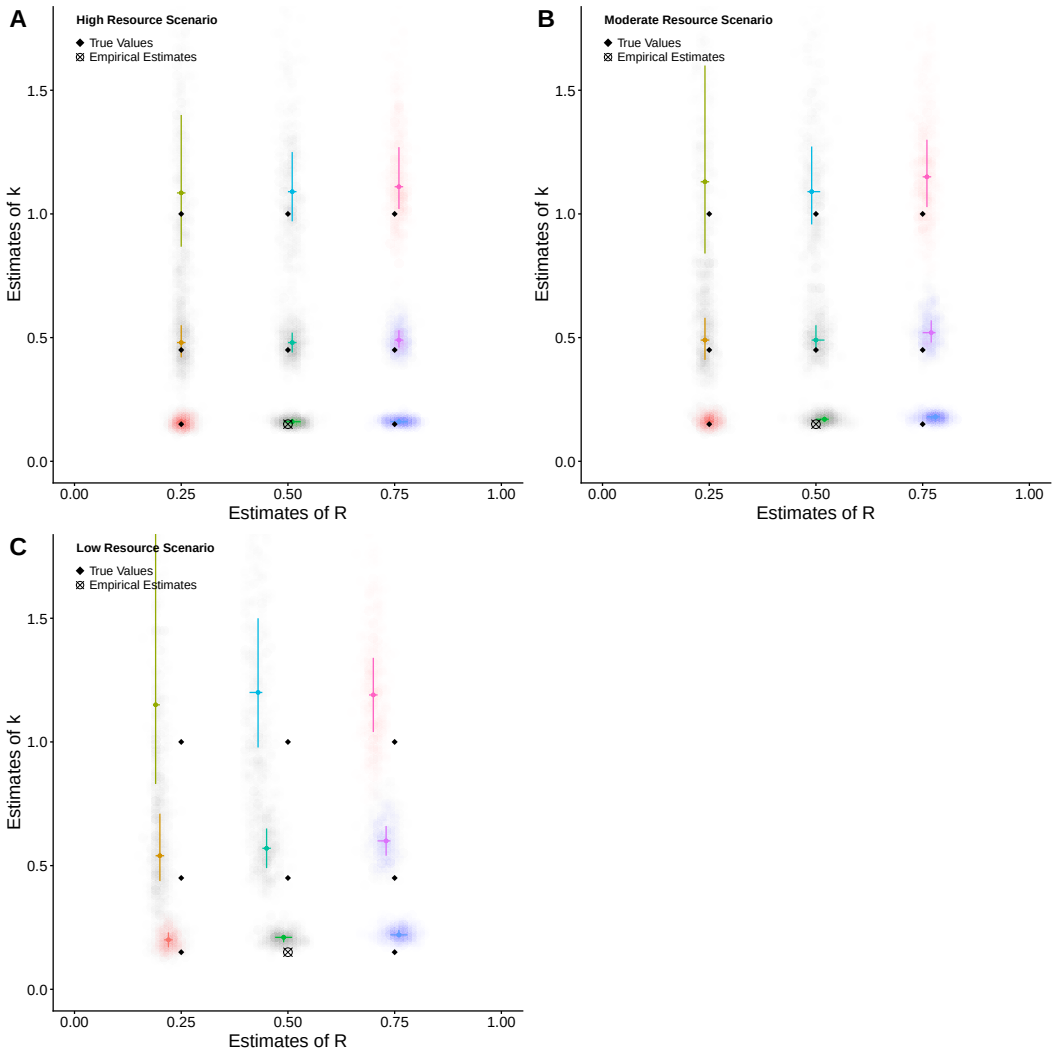
eFigure 3: Accounting for censoring in the likelihood. We corrected for censoring in the likelihood by jointly estimating the probability of fully observed clusters, $P(Y = y)$, and censored clusters, $P(Y \geq y)$. Panels A, C, and E represent 5%, 10%, and 20% censoring, respectively, and appear in Figure 3 of the main text using the full likelihood ("corrected"). Panels B, D, and F are their analogues that do not account for censoring ("uncorrected," i.e. all clusters are assumed fully observed, $P(Y = y)$). The approach to correct for censoring demonstrates a modest correction of estimates.



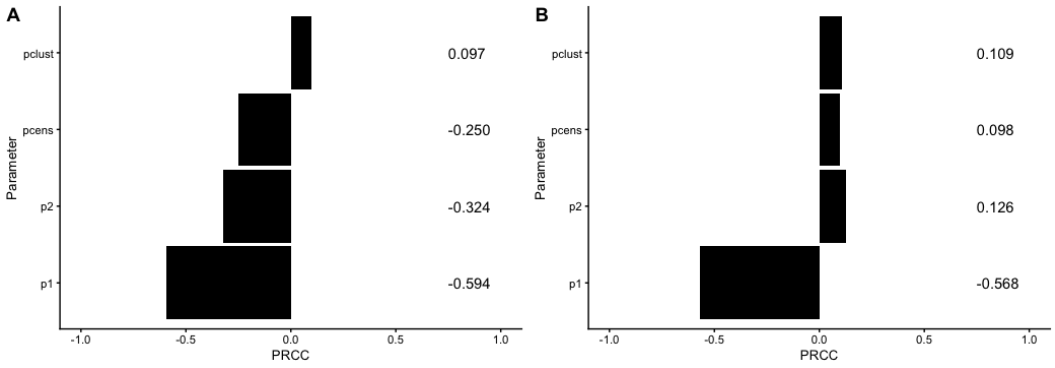
eFigure 4: Bias in $\hat{k}$ arising due to overlapping clusters ($R = 0.50; k = 0.15$). Results of simulated tuberculosis surveillance systems with overlapping chains according to the methods, for each value of k between 0.1 and 1.1 in 0.01 increments ($R = 0.50$). Simulations with (A) 5 percent, (B) 10 percent, and (C) 20 percent of TB chains in the surveillance system unable to be separated ("overlapping" per methods). Red represents results when the probability is conditioned on the number of overlapping chains in a given cluster ("Conditioned Likelihood"); blue represents the results when the number of overlapping chains is ignored ("Unconditioned Likelihood"). Shaded areas indicate 95 percent confidence intervals; grey line represents perfect inference.



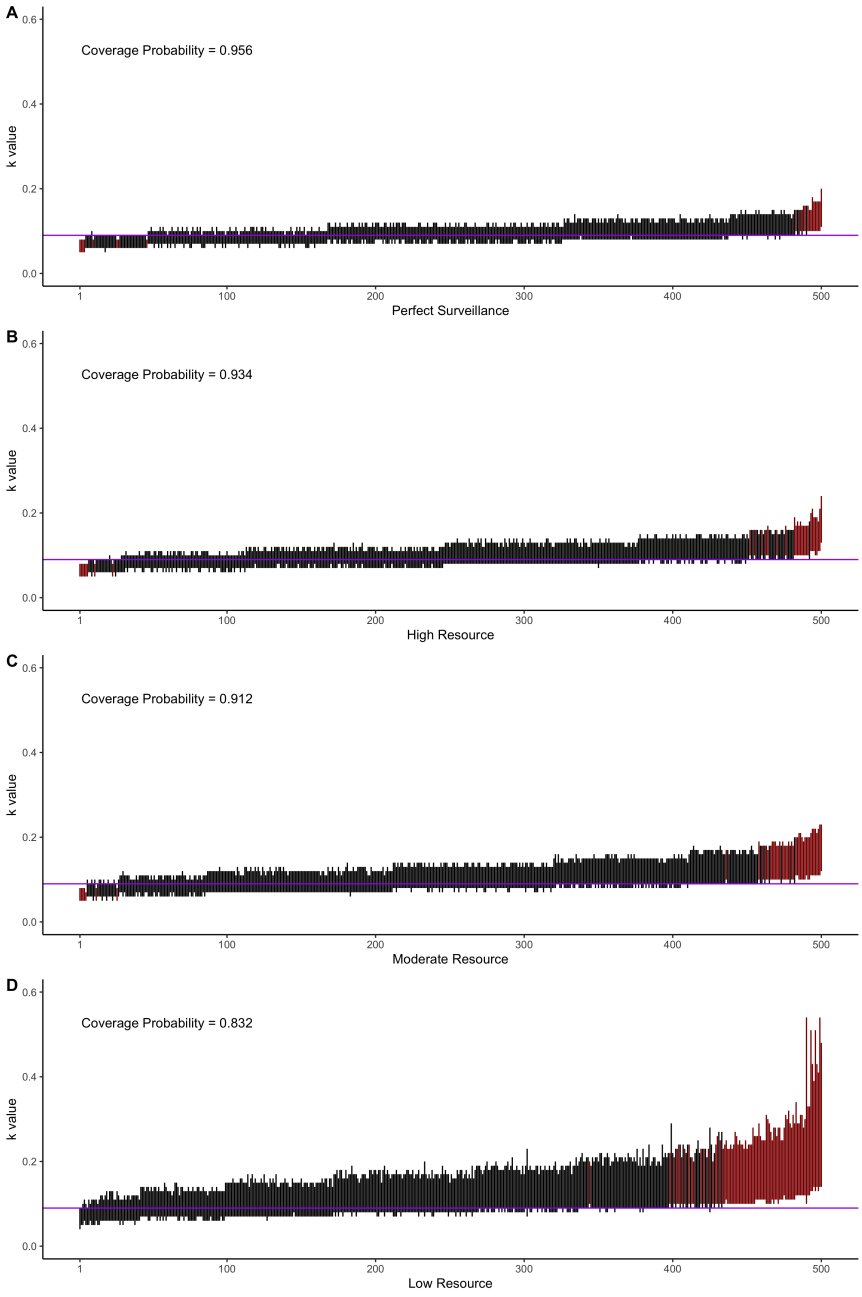
eFigure 5: Performance of inference procedure under imperfect surveillance ($R = 0.50; k = 0.15$). Results of 500 simulated TB surveillance systems under combined imperfect surveillance scenarios of A) high-resource; B) moderate-resource; and C) low-resource settings as defined in Table 2 of main text. Colored lines represent the interquartile range for each R and k combination; dots represent median values. Diamonds represent true values. R values were simulated at 0.25, 0.50 (empirical), and 0.75. k values were simulated at 0.15 (empirical), 0.45, and 1.0.



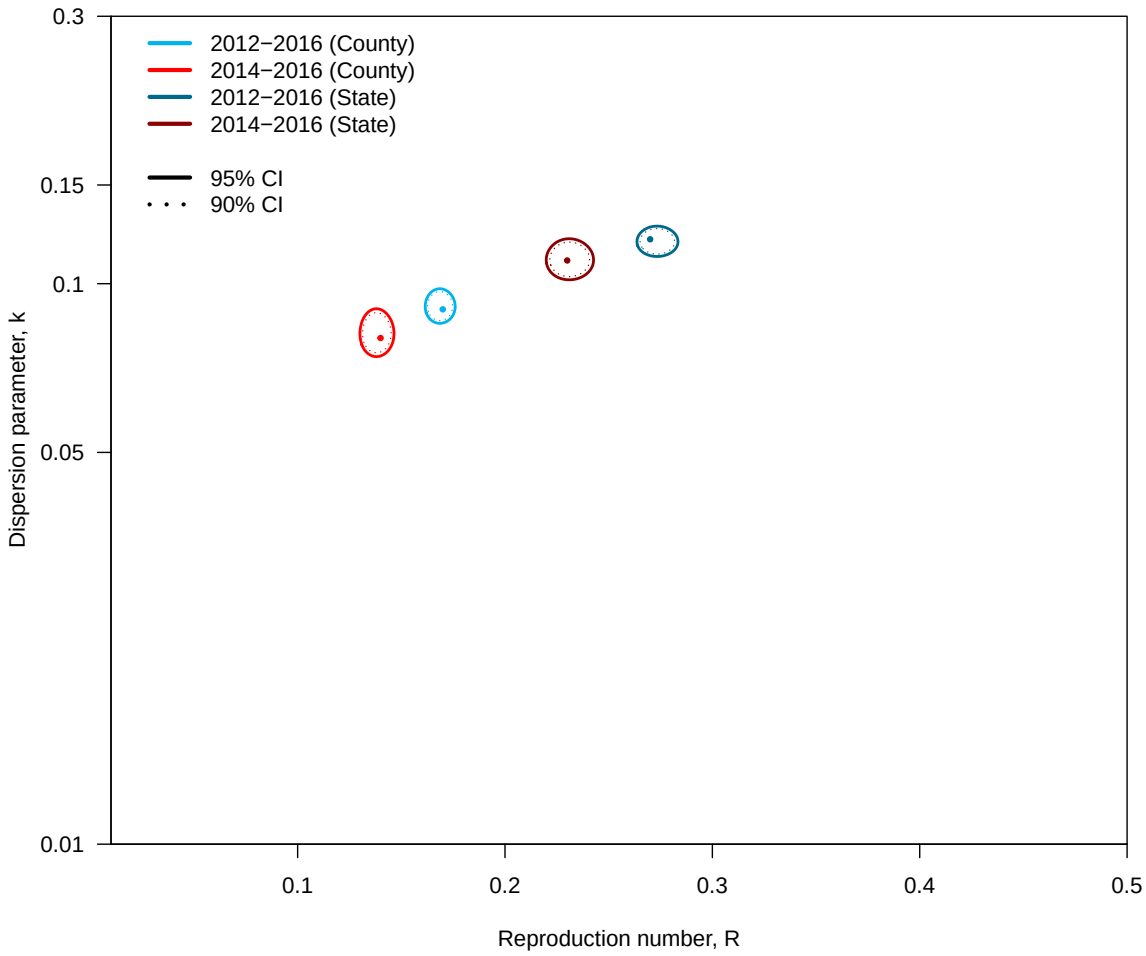
eFigure 6: Partial rank correlation coefficients (PRCCs) values for the cluster-based model under empirical parameter assumptions ($R = 0.50; k = 0.15$). A) PRCC values for the dispersion parameter, k. B) PRCC values for the reproductive number, R. Parameters p_1, p_2, p_{cens}, and p_{clust} represent the probability of passive observation, active case finding, censorship, and the proportion of overlapping clusters, respectively, as defined in the methods.



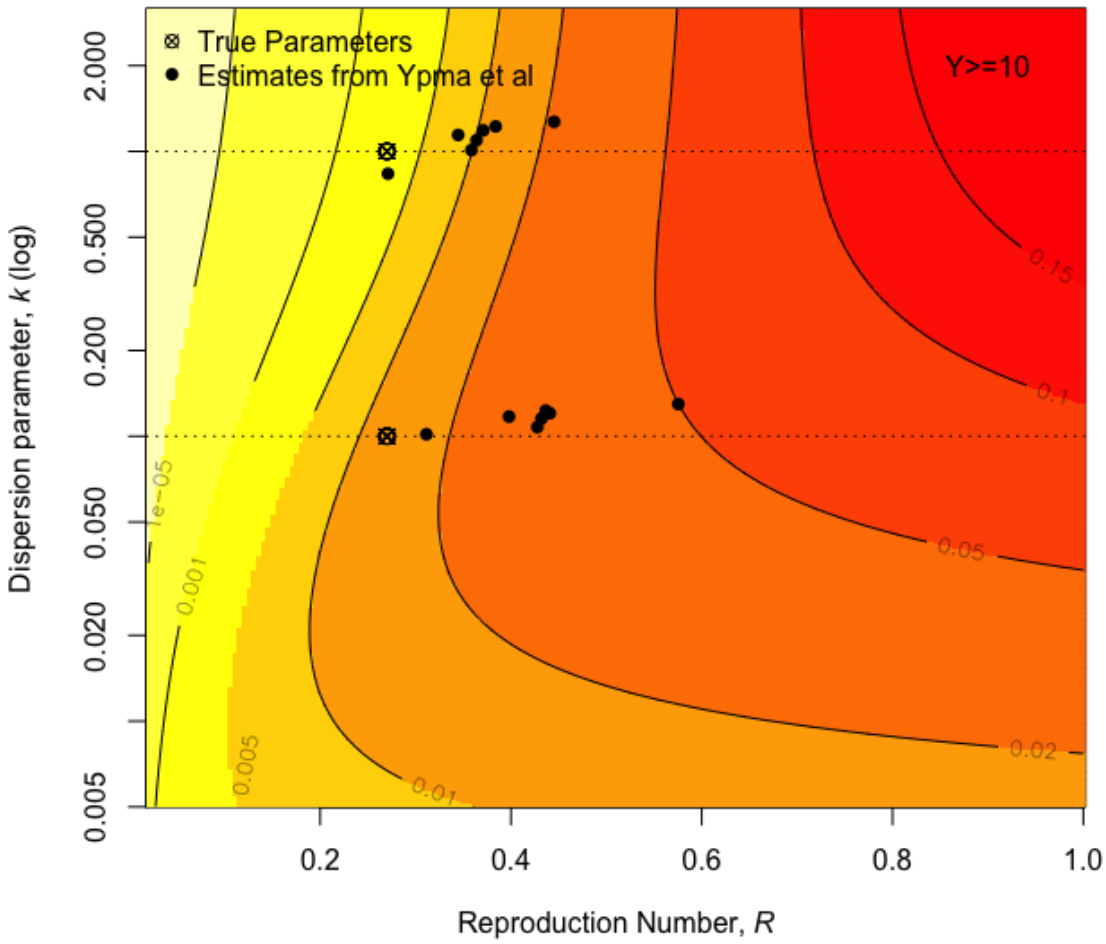
eFigure 7: Coverage probabilities under $R = 0.17$ and $k = 0.09$. Using maximum likelihood estimates (MLE) of $\hat{R} = 0.17$ and $\hat{k} = 0.19$ derived from the U.S. data in the main text, we simulated 500 surveillance systems under A) Perfect surveillance (coverage probability: 0.956); B) High-resource setting (coverage probability: 0.934); C) Moderate-resource setting (coverage probability: 0.912); D) Low-resource setting (coverage probability: 0.832). Vertical lines represent 95% confidence intervals (CI) for each simulation. The purple horizontal line represents the true parameter value of interest ($k = 0.17$). Black represents simulations containing the true parameter in the 95% CI; red represents simulations that do not contain the true parameter in the 95% CI.



eFigure 8: Simultaneous Confidence Regions of R and k in the United States under various cluster definitions. CI indicates confidence interval.



eFigure 9: The relationship between R, k, and $P(Y = y)$. Previous methods using *IS*6110 restriction fragment length polymorphism (RFLP) TB genotypic cluster sizes were unable to accurately infer R (see Ypma *et al, Epidemiology*, 2013). The authors simulated RFLP data with true $R = 0.27$ and true k as either 0.10 or 1.0 (cross-marked circles). The filled dots represent the authors median estimated R and k values upon sensitivity analysis. The shaded contours indicate the probability of observing a cluster of size $Y \geq 10$ (red indicates higher probability, with probabilities denoted as numbers on contour lines). While $\hat{k}$ was robust using these methods (all values are close to the respective dotted lines), $\hat{R}$ was highly sensitive. Overlaying the probability demonstrates that k must be interpreted with R to accurately characterize the propensity for a large outbreak.



6.2 Supplemental Tables

Supplemental Table S1: Coverage probabilities of the inference procedure under various k values. Each coverage probability was the result of 500 simulated surveillance systems, each with 2000 chains and $R = 0.25$.

True k value	Perfect Surveillance	High-resource Setting	Moderate-resource Setting	Low-resource Setting
0.10	0.966	0.900	0.886	0.646
0.30	0.966	0.944	0.934	0.860
0.50	0.960	0.938	0.930	0.926

Supplemental Table S2: Genotype TB cluster sizes in the United States by timeframe and geographic. Cluster sizes are defined by 24-locus MIRU-VNTR genotyping.

	5 Years (2012 to 2016)			3 Years (2014 to 2016)		
Cluster Size	Number of Clusters	Total Cases	Percent of Cases	Number of Clusters	Total Cases	Percent of Cases
County-level definition						
1	26580	26580	75%	16779	16779	81%
2	1638	3276	9%	893	1786	9%
3	474	1422	4%	224	672	3%
4	203	812	2%	90	360	2%
5	98	490	1%	42	210	1%
6	66	396	1%	33	198	1%
7	52	364	1%	21	147	1%
8	29	232	1%	5	40	≤1%
9	14	126	≤1%	12	108	1%
10	12	120	≤1%	5	50	≤1%
11	12	132	≤1%	6	66	≤1%
≥ 12	60	1363	4%	18	364	2%
State-level definition						
1	22154	22154	63%	14379	14379	69%
2	1921	3842	11%	1136	2272	11%
3	629	1887	5%	291	873	4%
4	250	1000	3%	128	512	2%
5	139	695	2%	73	365	2%
6	90	540	2%	47	282	1%
7	66	462	1%	35	245	1%
8	51	408	1%	22	176	1%
9	34	306	1%	14	126	1%
10	26	260	1%	17	170	1%
11	18	198	1%	11	121	1%
≥12	1621	3561	10%	59	1259	6%

7 Computer Code (R Language)

All code, including a tutorial on using these methods with external datasets, is available as R files in the following GitHub repository:

https://github.com/jpsmithuga/nbbpClusterAnalysis/

Key R functions are also presented below for posterity and convenience. Note: below code may differ from code found in GitHub as code in the dynamic repository may be refined.

7.1 Probability Density Function

The probability of a transmission chain originating with n index cases resulting in a final transmission cluster size of y can be expressed as:

$$P(Y = y|n) = \binom{n}{y} \frac{\Gamma(ky + y - n)}{\Gamma(ky)\Gamma(y - n - 1)} \frac{(\frac{R}{k})^{y-n}}{(1 + \frac{R}{k})^{ky+y-n}}$$

The following code returns the probability of observing a cluster size Y initiating with n index cases with specified negative binomial parameters R and k. The code returns the log if $logp = true$ and the probability if $logp = false$

```
nb_yn <- function(y,n,R,k, logp=TRUE) {
  logp_yn <- log(n)-log(y)+lgamma(k*y+y-n)-(lgamma(k*y)+lgamma
     (y-n+1))+(y-n)*log(R/k)-(k*y+y-n)*log(1+R/k)
  p_yn <- exp(logp_yn)
  if(logp){
    return(logp_yn)
  } else{return(p_yn)}
}
```

7.2 Branching Process Function

```
bp <- function(gens=20, init.size=1, offspring, ...){
  Z <- list() #initiate the list
  Z[[1]] <- init.size
  i <- 1
  while(sum(Z[[i]]) > 0 && i <= gens) {
    Z[[i+1]] <- offspring(sum(Z[[i]]), ...)
```

```
    i <- i+1
  }
  return(Z)
}
```

7.3 Imperfect Simulation Function

```
##'
   ------------------------------------------------------------------------

##' Simulating imperfect observation
##'      @param true_R       True underlying R value for NB
   branching process
##'      @param true_k       True underlying k value for NB
   branching process
##'      @param num_chains   Number of simulated transmission
   chains in a surveillance system
##'      @param p1           Probability of ascertaining cases
    by passive surveillance
##'      @param p2           Probability of ascertaining cases
    by active surveillance
##'      @param prob_cens    Probability that a chain will be
   censored
##'      @param perc_overlap Proportion of clusters that
   overlap (i.e. multiple index cases)
##'      - - - - - - - - - - - - - -
##'      @return Output is a list of length 2, each list
   contains a data frame of cluster sizes, index
##'              cases, and censored status, for:
##'                 [[1]] Perfect Surveillance
##'                 [[2]] Imperfect Surveillance
##'
   ------------------------------------------------------------------------

##'
imperfect <- function(true_R, true_k, num_chains = 2000, p1, p
   2, prob_cens, perc_overlap){
  z <- replicate(num_chains,bp(offspring = rnbinom, mu = true_
     R, size = true_k))
  z.pass <- z; z.act <- z
  ## - - - - - - - - - - - - - - - - - - - - - - -
  ## Imperfect case ascertainment
  ## - - - - - - - - - - - - - - - - - - - - - - -
  #Passive surveillance
  for (i in 1:length(z.pass)){
    for (j in 1:length(z.pass[[i]])){
```

```
    for (k in 1:length(z.pass[[i]][[j]])){
      for (l in 1:length(z.pass[[i]][[j]][[k]])){
        if (runif(1) < (1 - p1)){
          z.pass[[i]][[j]][[k]] <- NA
        }}}}}
#Active case finding (only chains with at least one case
   observed by passive surveillance)
##' Note: Index cases with no secondary cases present in the
    branching process as a
##' list of 2 (always [1] and [0]). The second value is what
    we are concerned about having an NA value via passive
##' surveillance. In the scenario where the list is [1] and
   [NA], this would artificially make the chain eligible for
##' reevaluation since the cluster is not completely missing
   . So before reevaluation with p2 we need to
##' assign the first position in all of the lists as NA.
   That way, if the index case is missing,
##' both values will be missing.
a <- z.pass
for (i in 1:length(a)){
  a[[i]][[1]] <- NA
}
b <- cbind(a, sapply(a, function(x) all(is.na(unlist(x)))))
   # Determine if at least one case in the chain has been
   seen
for (i in 1:length(z.pass)){
  if (b[i,2] == TRUE){
    z.act[[i]] <- a[[i]] # Skip if cluster no cases are
       observed
  } else {
    for (j in 1:length(z.pass[[i]])){
      for (k in 1:length(z.pass[[i]][[j]])){
        for (l in 1:length(z.pass[[i]][[j]][[k]])){
          if (is.na(z.pass[[i]][[j]][[k]])){
            if(runif(1) <= (p2)){       #Active probability
               of being seen by case detection
              z.act[[i]][[j]][[k]] <- z[[i]][[j]][[k]] #
                 Reassign original value
            } else {z.act[[i]][[j]][[k]] <- z.pass[[i]][[j
               ]][[k]]}
          }}}}}
```

```
# "Break" chains based on the position of missing cases in
   the chain
l <- z.act  #dummy/temp data to not change z.act
for (i in 1:length(l)){
  l[[i]][[1]] <- NULL #remove first position of the nested
     list so that it eases summing lengths (can't sum based
     on integer values in imperfect observations)
}
#"Break apart" the chains
t1 <- lapply(lapply(seq_along(l), function(nm) {split(l[[nm
   ]], cumsum(sapply(l[[nm]], function(x) all(is.na(x)))))})
   , function(lstA) lapply(lstA,function(x) Filter(function(
   y) !all(is.na(y)), x)))
t2 <- rapply(unlist(t1,recursive=FALSE),function(x) x[!is.na
   (x)], how="replace") #Remove NA values.
z.broken <- Filter(length,t2) #remove all with length 0 (
   missing/unobserved)

## - - - - - - - - - - - - - - - - - - - - - - -
## Censoring
## - - - - - - - - - - - - - - - - - - - - - - -

z.cen <- z.broken                          # Initialize the
   censored list
for (i in 1:length(z.broken)){             # Iterate through
    the list
  if (length(z.broken[[i]]) > 1) {           # List must
     have at least length of two (cant be censored if the
     index case isn't seen, then it is unobserved as above)
    if(runif(1) <= prob_cens){               # Stochastic
       process to determine if the nested list will be
       censored
      if(length(z.broken[[i]]) == 2){
        n <- 2} else {                       # n has to be 2
           for chains with only two generations
          n <- sample(2:length(z.broken[[i]]), 1)}   #
             Randomly determine what list position in the
             nested list will be the censor threshold
      z.cen[[i]][n:length(z.broken[[i]])] <- NA   # Fill all
         positions from n to the end of the nested list
        with NA
```

```
    }}}
out_list <- lapply(z.cen, function(x) {    # Remove all
   nested list elements that contain NA
  inds <- sapply(x, function(x) any(is.na(x)))
  if(any(inds)) x[seq_len(which.max(inds) - 1)] else x})
cens <- numeric(length(out_list))
true <- numeric(length(out_list))
for (k in 1:length(out_list)){
  cens[k] <- sum(lengths(out_list[[k]])) # Get cluster size
     of censored clusters
  true[k] <- sum(lengths(z.broken[[k]])) # Get cluster size
     of uncensored (but imperfect obs) clusters
}
Y_cens <- data.frame(y.cens = cens, censor = ifelse(cens!=
   true,1,0)) #Create a censoring index (1=censored, 0=
   uncensored)

## - - - - - - - - - - - - - - - - - - - - - - - -
## Overlapping clusters
## - - - - - - - - - - - - - - - - - - - - - - - -
#' Determine sampling space - i.e. how many clusters get
   merged with each iteration
#'     choose j from Poisson with lambda=2
lamda <- 2
a <- rpois(10000000,lamda) # Drawn from a Poisson with lamda
   =2 for main text
n <- round(nrow(Y_cens)*perc_overlap)          # Determine
   the number to be merged
sample_clust <- sample(a[a > 1], size = n, replace = TRUE) #
    Randomly choose j (discard 0 and 1)
names(sample_clust) <- paste0("S", 1:n)
m <- nrow(Y_cens) - sum(sample_clust) # Determine the number
    that will not be merged (m)
non_merge_clust <- rep(1, m)         # Create a vector with
   replicated 1 based on m
names(non_merge_clust) <- paste0("N", 1:m)
combine_clust <- c(sample_clust, non_merge_clust) # Combine
   sample_clust and non_merge_clust, and then randomly sort
   the vector
combine_clust2 <- sample(combine_clust, size = length(
   combine_clust))
```

```
  expand_list <- list(lengths = combine_clust2, values = names
     (combine_clust2))# Expand the vector
  expand_clust <- inverse.rle(expand_list)
  dat <- data.frame(Y_cens, group = factor(expand_clust,
     levels = unique(expand_clust)))# Create a data frame with
      y and expand_clust
  dat$index <- 1 # add the index case number for summing
  dat2 <- aggregate(cbind(dat$y.cens, dat$index, dat$censor),
     by = list(group = dat$group), FUN = sum)  # Convert dat2
     to a matrix, sum the index cases and censoring index,
     remove the group column
  dat2$group <- NULL
  y.merged <- as.matrix(dat2); colnames(y.merged) <- c("clust_
     size","index_cases","censor_status")
  y.final <- data.frame(y.merged) #just to be safe
  y.final$censor_status <- ifelse(y.final$censor_status >= 1,1
     ,0) # if more than 1 censored clusters merged
  ## - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      - - - - - - - - - - - - - -
  y.true <- unlist(lapply(z,function(x) sum(unlist(x)))) # Sum
      true cluster sizes
  Y.true <- data.frame(y.true = y.true,  index = rep(1, times
     = length(y.true)), censor = rep(0, times = length(y.true)
     ))  # Format data for perfect surveillance (no censoring
     or subclustering)
  names(Y.true) <- c("clust_size","index_cases","censor_status
     ")
  return(list(Y.true, y.final))
  #return(list(z, z.pass, z.act, z.broken, out_list, cens,
     true, Y_cens, y.final)) #for validation
}
```

7.4 Likelihood Function

```
cens_likelihood <- function(Y,R,k) {
  p_function <- function(y,n){          #Dummy function to
     apply
    exp(log(n)-log(y)+lgamma(k*y+y-n)-(lgamma(k*y)+lgamma(y-n+
       1))+(y-n)*log(R/k)-(k*y+y-n)*log(1+R/k)) #PDF as
       defined un methods
  }
  ya <- Y[Y[,3]==0,] # Uncensored clusters
  yb <- Y[Y[,3]==1,] # Censored clusters

  liks_a <- log(p_function(ya[,1],ya[,2])) # Can apply P(Y=y)
     via vectorization

  liks_b <- numeric(nrow(yb))             # Not sure how to
     vectorize with the $P(Y \geq y)$ being of the 1-sum(p_
     function(1:(y-1),n)) below
  if(nrow(yb)>0){                         # This for loop
     approach is reasonably fast (about 9 seconds on a list of
      2000 cluster sizes)
    for (i in 1:nrow(yb)){
      y <- yb[i,1]
      n <- yb[i,2]
      if (y==1){                         # If the cluster size
         is 1, the P(1)=1, thus log(1)=0
        liks_b[i] <- 0                    # This trick
           prevents issues with running the code
      } else{
        if (is.nan(log(max(10^-300,1-sum(p_function(1:(y-1),n)
           )))))){ # Trick to avoid NaN due to extremely
           unlikely clusters (very rare, but was causing
           numeric overflow problems)
          liks_b[i] <- log(10^-300)
        } else {liks_b[i] <- log(max(10^-300,1-sum(p_function(
           1:(y-1),n))))}
      }}}
  sumliks <- sum(liks_a,liks_b)
  return(sumliks)
    #return(list(liks_a,liks_b)) #validate
}
```

7.5 Parameter Estimation Functions

```
##'
   ------------------------------------------------------------------------

##' Surface/profile likelihood function
##' Calculates likelihoods over a range of R and k values
##'      @param data 3-column data frame or matrix containing
##'                  [,1] Custer Size
##'                  [,2] Index Cases
##'                  [,3] Censored status
##'      @param Rrange Range of R values
##'      @param krange Range of k values
##'      - - - - - - - - - - - - - -
##'      @return Rrange by krange Matrix with likelihoods
##'
   ------------------------------------------------------------------------


surflike <- function(data, Rrange, krange){
  likesurf <- matrix(NA, nrow = length(Rrange),length(krange))
  for(i in 1:length(Rrange)){
    for(j in 1:length(krange)){
      likesurf[i,j] <- likelihood(data, Rrange[i],krange[j])
    }
  }
  return(likesurf)
}
##'
   ------------------------------------------------------------------------

##' Parameter Estimation
##' Estimates MLE and confidence interval for R and k
##'      @param ls likelihood surface data obtained from
   surflike() function
##'      @param ls_max logical likelihood surface data
   identifying max  (ls_max <- ls==max(ls))
##'      @param conf.interval Desired confidence interval (as
   decimal, i.e. 0.95)
##'      - - - - - - - - - - - - - -
##'      @return Point, lower, and upper bound estimates for R
```

```
    and k
##'
   ------------------------------------------------------------------------


calc_profile <- function(ls, ls_max, Rrange, krange, conf.
   interval){
  chiV <- qchisq(conf.interval/100, df = 1) / 2
  prfk <- apply(ls,2,function(x){max(x)})
  prfk2 <- krange[prfk - max(prfk) >- chiV]
  prfR <- apply(ls,1,function(x){max(x)})
  prfR2 <- Rrange[prfR - max(prfR) >- chiV]

  output <- rbind(cbind(Rrange[sum(seq(1, length(Rrange)) %*%
     ls_max)], min(prfR2),max(prfR2)),
                  cbind(krange[sum(ls_max %*% seq(1, length(
                     krange)))], min(prfk2),max(prfk2)))
  colnames(output) <- c("point_est","lower_ci","upper_ci")
  rownames(output) <- c("R","k")
  return(output)
}
```

7.6 Applying Methods to External Datasets

```
##' This code will explain how to use these methods with
   external surveillance cluster data
##'
#
   ########################################################################

## Preparing the data
#
   ########################################################################


##' First, the data need to be prepared with three columns:
##'     1: cluster size (with each row a unique cluster)
##'     2: number of subclusters/index cases
##'     3: censorship status
##'
```

```
##' Typically, the second and third columns are 1 and 0,
   respectively

##' Each row in the dataset needs to be an individual cluster,
    so the
##' length of the dataset is the number of clusters in the
   surveillance
##' system. However, often data come in the form of a table, i
   .e.:
##' ----------------------------
##' Cluster size | Frequency
##' ----------------------------
##'            1 | 4003
##'            2 | 362
##'            3 | 174
##'            4 | 92
##'            5 | 75
##'            6 | 41
##'            7 | 44
##'            8 | 19
##'            .....
##'            52 | 1
##'            68 | 1
##'            77 | 1
##'
##' The below code will help you convert table data of cluster
    sizes to the
##' format appropriate for this code

##' First, we will create example table data for demonstration
    purposes using the branching
##' process function to generate a surveillance system

bp <- function(gens = 100, init.size = 1, offspring, ...){
  Z <- list() #initiate the list
  Z[[1]] <- init.size #set the first position of the list as
     the number of index cases
  i <- 1
  while(sum(Z[[i]]) > 0 && i <= gens) {
    Z[[i+1]] <- offspring(sum(Z[[i]]), ...)
    i <- i+1
```

```
  }
  return(Z)
}
##' We can now generate our example table data. Here we will
   simulate a
##' surveillance system originating with 1000 chains, with an
   offspring
##' distributed with mean R = 0.50 and dispersion k = 0.15

num_chains <- 1000
R <- 0.50
k <- 0.15

set.seed(05062020)
table_data <- data.frame(table(unlist(lapply(replicate(num_
   chains, bp(offspring = rnbinom, mu = R, size = k)),function
   (x) sum(unlist(x))))))
names(table_data) <- c("cluster_size","frequency") # rename
   columns
table_data[ , "cluster_size"] <- as.numeric(as.character(table
   _data[ , "cluster_size"])) # convert to numeric

##' In these data, the first column is the cluster size, the
   second column is the frequency.
##' Ensure your data are in this format.
##' To convert it to the proper long format (each individual
   cluster is a row) for use in these methods,
##' run the following step on your table data. This assumes no
    subclustering or censoring.

Y_data <- data.frame(clust_sizes = table_data[rep(seq_len(nrow
   (table_data)), table_data[, "frequency"]), ][ , 1],
                     index_cases = rep(1, times = sum(table_
                        data[, "frequency"])),
                     cens_status = rep(0, times = sum(table_
                        data[, "frequency"])))

##' If you want to assume some definition for censoring, i.e.,
    clusters larger than some threshold are censored,
##' you can easily do this:
##'
```

```
# cens_threshold <- 10
# Y_data[Y_data[ , "clust_sizes"] >= cens_threshold, "cens_
   status"] <- 1 # all clusters Y >= 10

#
   ##########################################################################

## Using the likelihood and parameter estimation functions
#
   ##########################################################################

##' Now that the data are prepared and in the proper format,
   parameter estimation is quite easy
##' Pull in the functions from the file, "Likelihood and
   Parameter Estimation Functions.R"
##' Note: the functions are also pasted below for convenience,
    in case the additional file is unavailable

# source("~Insert_Filepath/Likelihood and Parameter Estimation
    Functions.R")

##' If not yet defined, define search grid, which will be an R
    x k matrix with each cell containing
##' the likelihood of each R/k combination

resolution <- 0.01 # set resolution (increased resolution
   increases computer time needed)

# R range
R.min <- 0.01
R.max <- 1.00
Rrange <- seq(R.min, R.max, by = resolution)

# k range
k.min <- 0.01
k.max <- 1.00
krange <- seq(k.min, k.max, by = resolution)

# Calculate grid of likelihoods
Y_surflikes <- surflike(data = Y_data, Rrange = Rrange, krange
    = krange)
```

```
# find maximum in the x,y grid
Y_maxlikes <- Y_surflikes == max(Y_surflikes)

# estimate parameters
Y_MLE <- calc_profile(ls = Y_surflikes, ls_max = Y_maxlikes,
   Rrange = Rrange, krange = krange, conf.interval = 95)
Y_MLE

#
   ##############################################################################

## Creating a figure
#
   ##############################################################################


# Set values to calculate 90 and 95% confidence regions
CI95 <- qchisq(0.95, df = 1) / 2
CI90 <- qchisq(0.90, df = 1) / 2

# Calculate contour lines for 90 and 95% confidence regions
ctlns_95 <- contourLines(x = Rrange, y = log10(krange), as.
   matrix(Y_surflikes - max(Y_surflikes)), levels = c(-CI95,
   CI95+0.001))
ctlns_90 <- contourLines(x = Rrange, y = log10(krange), as.
   matrix(Y_surflikes - max(Y_surflikes)), levels = c(-CI90,
   CI90+0.001))

# Set some general figure parameters, i.e. axes, colors, etc
xtick <- seq(0, 1.4, 0.1)
ytick <- c(0.01, 0.05, 0.1, 0.15, 0.3, 0.5, 0.75, 1.0)

# Make figure
# pdf(paste0("~filepath/MLE_90_and_95_Estimates_",Sys.Date()
   ,".pdf"), width=10, height=8) # if you want to save PDF,
   uncomment this and "dev.off()" below
filled.contour(x = Rrange, y = log10(krange), as.matrix(Y_
   surflikes - max(Y_surflikes)), col='white', levels = seq(-3
   ,0,0.5),
               xlab = 'Reproduction␣number,␣R', ylab = "
```

```
                    Dispersion␣parameter,␣k␣(log␣scale)",
                  xlim=c(0.01, 1),
                  ylim=log10(c(0.01,max(krange))),
                  plot.axes = {
                    axis(1, at = xtick, label = xtick)
                    axis(2, at = log10(ytick), label = ytick)

                    points(Y_MLE["R","point_est"], log10(Y_MLE["k
                       ","point_est"]), pch = 19, cex = 0.5, col=
                       "black")
                    lines(ctlns_95[[1]][[2]], ctlns_95[[1]][[3]],
                        lty = 1, col = "black")
                    lines(ctlns_90[[1]][[2]], ctlns_90[[1]][[3]],
                        lty = 3, col = "black")
                  }
)
legend('topleft', c("95%␣CI", "90%␣CI"), lty = c(1,3), lwd = 2
   , col = "black", bty='n')
# dev.off()
```