Chapter 1

Risk Propagation

Risk propagation is a message-passing algorithm that estimates an individual's infection risk by considering their demographics, symptoms, diagnosis, and contact with others. Formally, a *risk score* $s_t \in [0, 1]$ is a timestamped infection probability where $t \in \mathbb{N}$ is the time of its computation. Thus, an individual with a high risk score is likely to test positive for the infection and poses a significant health risk to others. There are two types of risk scores: *symptom scores*, or prior infection probabilities, which account for an individual's demographics, symptoms, and diagnosis [42]; and *exposure scores*, or posterior infection probabilities, which incorporate the risk of direct and indirect contact with others.

Given their recent risk scores and contacts, an individual's exposure score is derived by marginalizing over the joint infection probability distribution. Naively computing this marginalization scales exponentially with the number of variables (i.e., individuals). To circumvent this intractability, the joint distribution is modeled as a factor graph, and an efficient message-passing procedure is employed to compute the marginal probabilities with a time complexity that scales linearly in the number of factor vertices (i.e., contacts).

Let $G = (X, F, E)$ be a *factor graph* where X is the set of variable vertices, F is the set of factor vertices, and E is the set of edges incident between them [33]. A *variable vertex* $x : \Omega \to \{0, 1\}$ is a random variable that represents the infection status of an individual, where the sample space is $\Omega = \{\textit{healthy}, \textit{infected}\}$ and

$$x(\omega) = \begin{cases} 0 & \text{if } \omega = \textit{healthy} \\ 1 & \text{if } \omega = \textit{infected}. \end{cases}$$

Thus, $p_t(x_i) = s_t$ is a risk score of the i-th individual. A *factor vertex* $f : X \times X \to [0, 1]$ defines the transmission of infection risk between two contacts. Specifically, contact between the i-th and j-th individual is represented by the factor vertex $f(x_i, x_j) = f_{ij}$, which is adjacent to the variable vertices x_i, x_j. Figure 1.1 depicts a factor graph that reflects the domain constraints.

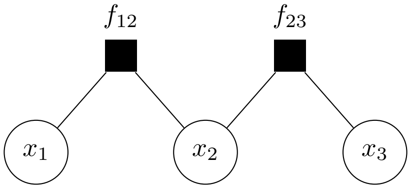

Figure 1.1: A factor graph of 3 variable vertices and 2 factor vertices.

1.1 Synchronous Risk Propagation

Ayday, Yoo, and Halimi [2] first proposed risk propagation as a synchronous, iterative message-passing algorithm that uses the factor graph to compute exposure scores. The first input to Risk-Propagation is the set family S, where

$$S_i = \{\, s_t \mid \tau - t < T_s \,\} \in S \tag{1.1}$$

is the set of recent risk scores of the i-th individual. The second input to Risk-Propagation is the contact set

$$C = \{\, (i, j, t) \mid i \neq j, \tau - t < T_c \,\} \tag{1.2}$$

such that (i, j, t) is the *most recent* contact between the i-th and j-th individual that occurred from time t until at least time $t + \delta$, where $\delta \in \mathbb{N}$ is the *minimum contact duration*[1]. Naturally, risk scores and contacts have finite relevance, so (1.1) and (1.2) are constrained by the *risk score expiry* $T_s \in \mathbb{N}$ and the *contact expiry* $T_c \in \mathbb{N}$, respectively. The *reference time* $\tau \in \mathbb{N}$ defines the relevance of the inputs and is assumed to be the time at which Risk-Propagation is invoked. For notational simplicity in Risk-Propagation, let X be a set. Then $\max X = 0$ if $X = \emptyset$.

[1] While Ayday, Yoo, and Halimi [2] require contact over a δ-contiguous period of time, the Centers for Disease Control and Prevention (2021) account for contact over a 24-hour period.

1.1.1 Variable Messages

The current exposure score of the i-th individual is defined as $\max S_i$. Hence, a *variable message* $\mu_{ij}^{(n)}$ from the variable vertex x_i to the factor vertex f_{ij} during the n-th iteration is the set of maximal risk scores $R_i^{(n-1)}$ from the previous $n-1$ iterations that were not derived by f_{ij}. In this way, risk propagation is reminiscent of the max-sum algorithm; however, risk propagation aims to maximize *individual* marginal probabilities rather than the joint distribution [6, pp. 411–415].

1.1.2 Factor Messages

A *factor message* $\lambda_{ij}^{(n)}$ from the factor vertex f_{ij} to the variable vertex x_j during the n-th iteration is an exposure score of the j-th individual that is based on interacting with those at most $n-1$ degrees separated from the i-th individual. This population is defined by the subgraph induced in G by

$$\{\, v \in X \cap F \setminus \{x_j, f_{ij}\} \mid d(x_i, v) \leq 2(n-1) \,\},$$

where $d(u, v)$ is the distance between the vertices u, v. The computation of a factor message assumes the following.

1. Contacts have a nondecreasing effect on an individual's exposure score.
2. A risk score s_t is *relevant* to the contact (i, j, t_{ij}) if $t < t_{ij} + \beta$, where $\beta \in \mathbb{N}$ is a *time buffer* that accounts for the incubation period, along with the delayed reporting of symptom scores and contacts.

3. Risk transmission between contacts is incomplete. Thus, a risk score decays exponentially along its transmission path in G at a rate of $\log \alpha$, where $\alpha \in (0, 1)$ is the *transmission rate*. Figure 1.2 visualizes this decay, assuming a transmission rate of $\alpha = 0.8$ [23].

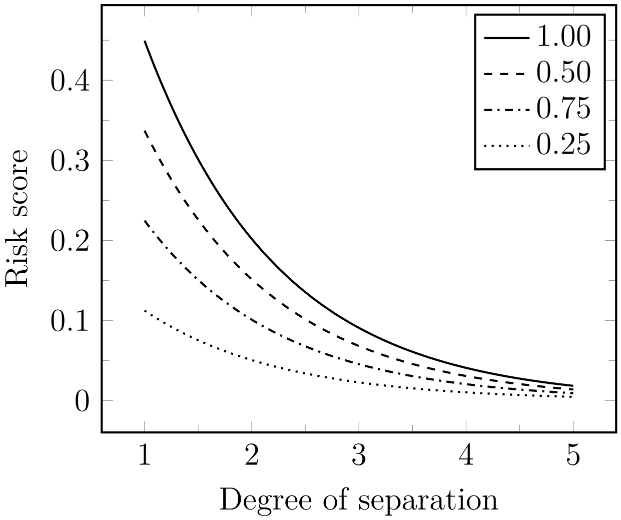

Figure 1.2: Exponential decay of risk scores.

To reiterate, a factor message $\lambda_{ij}^{(n)}$ is the maximum relevant risk score in the variable message $\mu_{ij}^{(n)}$ (or 0) that is scaled by the transmission rate α.

Ayday, Yoo, and Halimi [2] assume that the contact set C may contain (1) multiple contacts between the same two individuals and (2) *invalid* contacts, or those lasting less than δ time. However, these assumptions introduce unnecessary complexity. Regarding assumption 1, suppose the i-th and j-th individual come into contact m times such that $t_k < t_\ell$ for $1 \leq k < \ell \leq m$. Let Λ_k be the set of relevant risk scores, according to the contact time t_k, where

$$\Lambda_k = \{\, \alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_k + \beta \,\}.$$

Then $\Lambda_k \subseteq \Lambda_\ell$ if and only if $\max \Lambda_k \leq \max \Lambda_\ell$. Therefore, only the most recent contact time t_m is required to compute the factor message $\lambda_{ij}^{(n)}$. With respect to assumption 2, there are two possibilities.

1. If an individual has at least one valid contact, then their exposure score is computed over the subgraph induced in G by their contacts that define the neighborhood N_i of the variable vertex x_i.

2. If an individual has no valid contacts, then their exposure score is $\max S_i$ or 0, if all of their previously computed risk scores have expired.

In either case, a set C containing only valid contacts implies fewer factor vertices and edges in the factor graph G. Consequently, the complexity of Risk-Propagation is reduced by a constant factor since fewer messages must be computed.

1.1.3 Termination

To detect convergence, the normed difference between the current and previous exposure scores is compared to the threshold $\epsilon \in \mathbb{R}$. Note that $\mathbf{r}^{(n)}$ is the vector of exposure scores in the the n-th iteration such that $r_i^{(n)}$ is the i-th component of $\mathbf{r}^{(n)}$. The L_1 and L_∞ norms are sensible choices for detecting convergence. Ayday, Yoo, and Halimi [2] use the L_1 norm, which ensures that an individual's exposure score changed by at most ϵ after the penultimate iteration.

```
Risk-Propagation($S, C$)
$(X, F, E) \leftarrow$ Factor-Graph($C$)
$n \leftarrow 1$
for each $x_i \in X$
    $R_i^{(n-1)} \leftarrow$ top $k$ of $S_i$
    $r_i^{(n-1)} \leftarrow \max R_i^{(n-1)}$
    $r_i^{(n)} \leftarrow \infty$
while $\|\mathbf{r}^{(n)} - \mathbf{r}^{(n-1)}\| > \epsilon$
    for each $\{x_i, f_{ij}\} \in E$
        $\mu_{ij}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\, \lambda_{ji}^{(\ell)} \mid \ell \in [1 \,..\, n-1] \,\}$
    for each $\{x_i, f_{ij}\} \in E$
        $\lambda_{ij}^{(n)} \leftarrow \max \{\, \alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_{ij} + \beta \,\}$
    for each $x_i \in X$
        $R_i^{(n)} \leftarrow$ top $k$ of $\{\, \lambda_{ji}^{(n)} \mid f_{ij} \in N_i \,\}$
    for each $x_i \in X$
        $r_i^{(n-1)} \leftarrow r_i^{(n)}$
        $r_i^{(n)} \leftarrow \max R_i^{(n)}$
    $n \leftarrow n + 1$
return $\mathbf{r}^{(n)}$
```

1.2 Asynchronous Risk Propagation

While Risk-Propagation offers proof of concept, it is not viable for real-world application. Risk-Propagation is an *offline algorithm* [12], because it requires the contact and health information of all individuals as input. As

Ayday, Yoo, and Halimi [2] note, this centralization of personal data is not privacy-preserving. RISK-PROPAGATION is also inefficient. Most exposure scores are not likely to change across frequent invocations, which implies communication overhead and computational redundancy. To mitigate this inefficiency, Ayday et al. [3] suggest running RISK-PROPAGATION once or twice per day. Unfortunately, this cadence introduces substantial delay in updating individuals' exposure scores. In the midst of a pandemic, timely information is essential for individual and collective health.

To address the limitations of RISK-PROPAGATION, Ayday, Yoo, and Halimi [2] propose decentralizing the factor graph such that the processing entity associated with the i-th individual maintains the state of the i-th variable vertex and the neighboring factor vertices. Applying one-mode projection onto the variable vertices [54], Figure 1.3 illustrates how each entity corresponds to a portion of the factor graph. More generally, Ayday, Yoo, and Halimi [2] envision risk propagation as a decentralized communication protocol for informing individuals about their infection risk. Such a message-passing protocol naturally aligns with the *actor model* which describes concurrent computation as the sending and processing of messages amongst *actors* (Appendix D). Since communication is asynchronous in the actor model, risk propagation defined in this way is called *asynchronous risk propagation*.

1.2.1 Actor Behavior

An actor in the ShareTrace actor system corresponds to an individual. The CREATE-ACTOR operation [1] initializes an actor a with the following at-

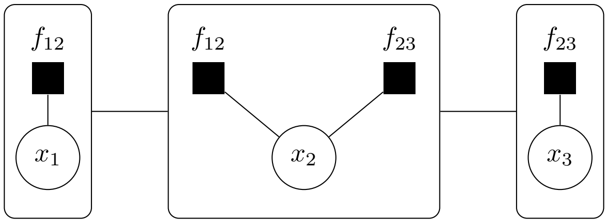

Figure 1.3: One-mode projection onto the variable vertices in Figure 1.1.

tributes.

- *a.exposure*: the current exposure score of the individual. This attribute is either a symptom score, a risk score sent by another actor, or the null risk score (see Null-Risk-Score).

- *a.contacts*: a *dictionary* (Appendix C) of contacts. In the context of an actor, a contact is a *proxy* [20] of the actor that represents an individual with which the individual represented by this actor was physically proximal. That is, if the i-th individual interacted with the j-th individual, then $a_i.contacts$ contains a contact c such that $c.key = c.name$ is a name of the j-th actor and $c.time$ is the most recent time of contact. This attribute extends the concept of *actor acquaintances* [25, 26, 1] to be time-varying.

- *a.scores*: a dictionary of exposure scores such that $s.key$ for an exposure score s is the time interval during which $a.exposure = s$. The null risk score is returned for queries in which the dictionary does not contain a risk score with a key that intersects the given query interval. Figure 1.4 depicts a hypothetical step function that *a.scores* represents.

```
Null-Risk-Score
$s.value \leftarrow 0$
$s.time \leftarrow 0$
$s.sender \leftarrow$ nil
**return** $s$
```

```
Create-Actor
$a.contacts \leftarrow \emptyset$
$a.scores \leftarrow \emptyset$
$a.exposure \leftarrow$ Null-Risk-Score
**return** $a$
```

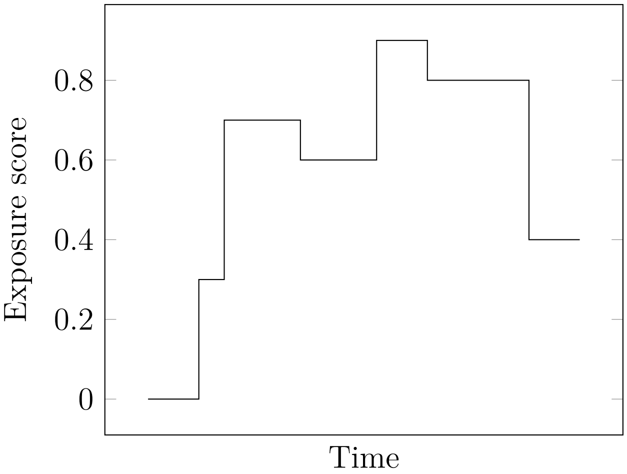

Figure 1.4: Hypothetical history of an individual's exposure score.

The interface of an actor is primarily defined by two types of messages: contacts and risk scores. Based on Section 1.1, let the *time to live* (TTL) of a message be the remaining time of its relevance. The reference time τ is assumed to be the current time.

```
Risk-Score-Ttl($s$)
    return $T_s - (\tau - s.time)$
```

```
Contact-Ttl($c$)
    return $T_c - (\tau - c.time)$
```

The Handle-Risk-Score operation defines how an actor behaves upon receiving a risk score. The key *s.key* initially associated with the risk score s is the time interval during which it is relevant. For the dictionary *a.scores*, Merge preserves the mapping invariant defined above such that risk scores are ordered first by value and then by time. Thus, $s.key \subseteq [s.time, s.time + T_s)$ for all exposure scores in *a.scores*. The Update-Exposure-Score operation is the online equivalent of updating an individual's exposure score in Risk-Propagation.

```
Handle-Risk-Score($a, s$)
    if Risk-Score-Ttl($s$) $> 0$
        $s.key \leftarrow [s.time, s.time + T_s)$
        Merge($a.scores, s$)
        Update-Exposure-Score($a, s$)
        for each $c \in a.contacts$
            Apply-Risk-Score($a, c, s$)
```

```
Update-Exposure-Score($a, s$)
if $a.exposure.value < s.value$
    $a.exposure \leftarrow s$
else if Risk-Score-Ttl($a.exposure$) $\leq 0$
    $a.exposure \leftarrow$ Maximum($a.scores$)
```

For the moment, assume that Apply-Risk-Score is the online equivalent of computing a factor message (see Section 1.1). Line 2 indicates that a copy s' of the risk score s is updated using the transmission rate α. The Send operation enables actor communication [1].

```
Apply-Risk-Score($a, c, s$)
1: if $c.time + \beta > s.time$
2:     $s'.value \leftarrow \alpha \cdot s.value$
3:     Send($c.name, s'$)
```

The problem with Apply-Risk-Score is that it causes risk scores to propagate *ad infinitum*. Unlike Risk-Propagation, a global convergence test is not available to terminate message passing, so it is necessary to define a local condition that determines if a risk score should be sent to another actor. Practically, "self-terminating" message-passing is necessary for asynchronous risk propagation to be scalable and cost-efficient.

The intent of sending a risk score to an actor is to update its exposure score. According to Handle-Risk-Score, it is only necessary to send an actor risk scores with values that exceed its current exposure score. Thus, an actor can associate a *send threshold* with a contact such that the target actor only receives risk scores that exceed the threshold. To permit a trade-off between accuracy and efficiency, let the *send coefficient* $\gamma \in \mathbb{R}$ be a scaling factor that is applied to a risk score upon setting the send threshold.

```
Set-Send-Threshold($c, s$)
1: $s'.value \leftarrow \gamma \cdot s.value$
2: $c.threshold \leftarrow s'$
```

The Apply-Risk-Score operation that incorporates the concept of a send threshold is defined below. Assuming a finite number of actors, a positive send coefficient guarantees that a risk score has finite propagation.

```
Apply-Risk-Score($a, c, s$)
1: if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
2:     $s'.value \leftarrow \alpha \cdot s.value$
3:     Set-Send-Threshold($c, s'$)
4:     Send($c.name, s'$)
```

The Set-Send-Threshold operation defines *how* the send threshold is updated, but not *when* it should be updated; the Update-Send-Threshold operation encapsulates the latter. The second predicate on Line 1 stems from the fact that the send threshold is a risk score and thus subject to expiry. The first predicate, however, is more subtle and will be revisited shortly. The Maximum-Older-Than operation is the same as Maximum, with the constraint that the key of the risk score intersects the query interval $(-\infty, c.time + \beta)$. Thus, the returned risk score is always relevant to the contact. Consistent with Apply-Risk-Score, the risk score retrieved from *a.scores* is scaled by the transmission rate and set as the new send threshold.

```
UPDATE-SEND-THRESHOLD($a, c$)
1: if $c.threshold.value > 0$ and RISK-SCORE-TTL($c.threshold$) $\leq 0$
2:     $s \leftarrow$ MAXIMUM-OLDER-THAN($a.scores, c.time + \beta$)
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     SET-SEND-THRESHOLD($c, s'$)
```

The send threshold should be current when evaluating Line 1 of APPLY-RISK-SCORE above, so the UPDATE-SEND-THRESHOLD operation should be invoked beforehand.

```
APPLY-RISK-SCORE($a, c, s$)
1: UPDATE-SEND-THRESHOLD($a, c$)
2: if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     SET-SEND-THRESHOLD($c, s'$)
5:     SEND($c.name, s'$)
```

Returning to the first predicate on Line 1 of UPDATE-SEND-THRESHOLD, the send threshold has a value of 0 when initially assigning the send threshold to be the null risk score; and when no key in *a.scores* intersects the query interval, so the send threshold is again assigned the null risk score. Suppose the first predicate is omitted from Line 1. Given that UPDATE-SEND-THRESHOLD is the first statement in APPLY-RISK-SCORE, it is possible that the send threshold is set prior to sending the target actor a relevant risk score. In the worst case, this prevents *all* risk scores from being sent to that actor, and the corresponding individual is inaccurately provided a low risk of infection. For correct message-passing behavior, the send threshold is updated only when its value is nonzero.

The aforementioned refinements to Apply-Risk-Score have focused on ensuring that message passing terminates and behaves correctly over time. To conclude the definition of Apply-Risk-Score, a message-passing optimization, called *sender-side aggregation* [41], will be introduced. Over a given period of time, an actor may receive several risk scores that are subsequently sent to multiple target actors. Rather than sending multiple risk scores, it would be more efficient to just send the final risk score. As a heuristic, Apply-Risk-Score can be modified so that a contact "buffers" a risk score that the target actor should receive.

```
Apply-Risk-Score($a, c, s$)
Update-Send-Threshold($a, c$)
if $c.threshold.value < s.value$ and $c.time + \beta > s.time$
    $s'.value \leftarrow \alpha \cdot s.value$
    Set-Send-Threshold($c, s'$)
    if $c.name \neq s.sender$
        $c.buffered \leftarrow s'$
```

When the actor receives a periodic *flush timeout* message, all contacts are "flushed" by sending the buffered risk scores to the target actors. For convenience, the Handle-Flush-Timeout operation also removes all expired contacts from $a.contacts$.

```
Handle-Flush-Timeout($a$)
for each $c \in a.contacts$
    if $c.buffered \neq$ nil
        Send($c.name, c.buffered$)
        $c.buffered \leftarrow$ nil
    if Contact-Ttl($c$) $\leq 0$
        Delete($a.contacts, c$)
```

The Handle-Contact operation concludes this section on actor behavior. Similar to Handle-Risk-Score, expired contacts are not processed. The Merge operation for *a.contacts* differs from its usage with *a.scores*. That is, if a contact with the same key already exists, its contact time is updated to that of the newer contact; all other state of the previous contact is maintained. A risk score from *a.scores* is also applied to the contact to ensure that, if the actor receives no other risk score before the contact expires, the target actor is sent at least one relevant risk score.

```
Handle-Contact($a, c$)
if Contact-Ttl($c$) $> 0$
    $c.threshold \leftarrow$ Null-Risk-Score
    $c.buffered \leftarrow$ nil
    $c.key \leftarrow c.name$
    Merge($a.contacts, c$)
    $s \leftarrow$ Maximum-Older-Than($a.scores, c.time + \beta$)
    Apply-Risk-Score($a, c, s$)
```

1.2.2 Message Reachability

The ShareTrace actor system is a *contact network*, a type of *temporal network* in which vertices represent individuals and edges indicate contact between them. The contact set defined in (1.2) is the *contact sequence* representation of a contact network [29]. Contact networks are typically used in epidemiological studies that aim to model and analyze the spreading dynamics of infection [49, 15, 36, 13, 47, 32, 55]. The ShareTrace actor network, however, models the spreading of infection *risk*. Holme and Saramäki [29] note that the transmission graph proposed by Riolo, Koopman, and Chick [49] "cannot handle edges where one node manages to not catch the disease." By framing the spreading phenomenon as continuous, rather than discrete, it is possible to model partial transmission.

A primitive of temporal reachability analysis is the *time-respecting path*: a contiguous sequence of contacts with nondecreasing time. Thus, vertex v is *temporally reachable* from vertex u if there exists a time-respecting path from u to v [44]. The following derivatives of a time-respecting path help quantify reachability in temporal networks [29].

- *influence set* $I(v)$: vertices that v can reach by a time-respecting path.
- *source set* $S(v)$: vertices that can reach v by a time-respecting path.
- *reachability ratio* $\rho(G)$: the average influence set cardinality in G.

Generally, a message-passing algorithm specifies constraints that determine when and what messages are sent between vertices. Even if operating on a temporal network, those constraints may be more or less strict than requiring temporal reachability. As a dynamic process, message passing on a time-varying network requires a broader definition of reachability that accounts for network topology *and* message-passing semantics [4].

Formally, the *reachability of a message m from vertex u to vertex v* is the number of edges along the shortest path P that satisfy the message-passing constraints,

$$r_m(u, v) = \sum_{(i,j) \in P} f_m(u, i, j, v),$$

where

$$f_m(u, i, j, v) = \begin{cases} 1 & \text{if all constraints are satisfied} \\ 0 & \text{otherwise.} \end{cases}$$

The vertex v is *reachable from vertex u for the message m* if there exists a shortest path P such that $r_m(u, v) = |P|$. The *reachability of a message m from vertex u* is

$$r_m(u) = \max_{v \in V} r_m(u, v). \tag{1.3}$$

Measures of temporal reachability can be extended to message reachability,

$$I_m(u) = \{\, v \in V \mid r_m(u, v) = |P| \,\}$$

$$S_m(v) = \{\, u \in V \mid r_m(u, v) = |P| \,\}$$

$$\rho(G, M) = \sum_{v \in V,\, m \in M} |I_m(v)| \cdot |V|^{-1},$$

where M is the set of messages associated with the vertices in V.

Asynchronous Risk Propagation

Let P be the set of contact edges along the shortest path from actor u to actor v such that the actors are enumerated $1, \ldots, |P|$. Let (s_u, t_u) be a symptom score of actor u. Let s_{ij} be the send threshold of the i-th actor for the j-th actor. Then message reachability for asynchronous risk propagation is defined as

$$r(u, v) = \sum_{(i,j) \in P} [\alpha^i s_u > \gamma \alpha s_{ij}] \cdot [t_u < t_{ij} + \beta], \tag{1.4}$$

where $[\cdot]$ is the Iverson bracket[2].

By relaxing the temporal constraint in (1.4), an upper bound on the reachability of a symptom score can be defined. Reversing the inequality of the first term in (1.4) and solving for i,

$$\hat{r}(u, v) \leq \begin{cases} 0 & \text{if } s_u = 0 \\ |P| & \text{if } s_v = 0 \\ \log_\alpha \gamma + \log_\alpha s_v - \log_\alpha s_u & \text{otherwise,} \end{cases} \tag{1.5}$$

where s_v is the send threshold for actor v of the preceding actor along the path P. Assuming a transmission rate of $\alpha = 0.8$ [23] and a send coefficient of $\gamma = 1$, Figure 1.5 provides a visual interpretation of (1.5).

Equation (1.5) indicates that a lower send coefficient will generally result in higher message reachability, which increases the likelihood of redundant communication (i.e., sending risk scores that do not update the exposure scores of

[2] The *Iverson bracket* $[x]$ is the indicator function of the set values for which x is true.

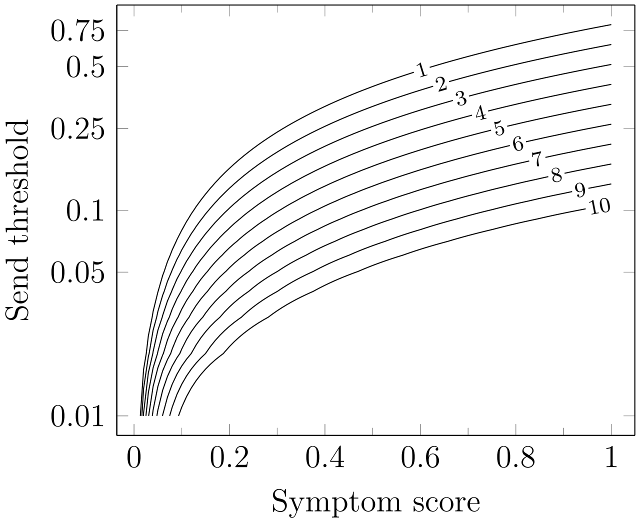

Figure 1.5: Message reachability for asynchronous risk propagation. Contour lines are shown for integral reachability values. Given a symptom score, the contour lines indicate the maximum send threshold that permits sending the risk score.

other actors). Equation (1.5) also quantifies the effect of the transmission rate. However, the transmission rate should be derived from epidemiology in order to quantify infectivity, so it should not be optimized to improve performance.

1.3 Mobile Crowdsensing

Mobile crowdsensing (MCS) is a "sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices" that is aggregated "in the cloud for crowd intelligence extraction and human-centric service delivery" [22]. Over the past decade, substantial research has been conducted on defining and classifying MCS applications [10, 22, and references therein]. While not discussed in previous work [3, 2], ShareTrace is a MCS application. The following characterization of ShareTrace assumes the four-layered architecture of a MCS application [10], which offers a comprehensive set of classification criteria that To offer a clear comparison, Table 1.1 follows the same structure as Capponi et al. [10]. Some aspects of the architecture, namely sampling frequency and sensor activity, are marked according to how ShareTrace is described in previous work, rather than how it would function to optimize for energy efficiency. More detail is provided below in this regard. When classifying ShareTrace as an MCS application, the following description is helpful:

> *ShareTrace is a decentralized, delay-tolerant contact-tracing application that estimates infection risk from proximal user interactions and user symptoms.*

1.3.1 Application Layer

1.3.1.1 Application Tasks

Task Scheduling *Proactive scheduling* allows users to decide when and where they contribute sensing data, while *reactive scheduling* requires that "a user receives a request, and upon acceptance, accomplishes a task" [10]. ShareTrace follows proactive scheduling where the sensing task is to detect proximal interactions with other users. Naturally, the scheduling of this task is at the discretion of the ShareTrace user.

Task Assignment *Centralized assignment* assumes that "a central authority distributes tasks among users." Conversely, with *decentralized assignment*, "each participant becomes an authority and can either perform the task or forward it to other users. This approach is very useful when users are very interested in specific events or activities" [10]. The latter naturally aligns with ShareTrace in which each user is responsible for their own interactions that are both temporally and spatially specific.

Task Execution With *single-task execution*, MCS applications assign one type of task to users, while *multi-task execution* assigns multiple types of tasks [10]. ShareTrace only involves the single task of sensing proximal interactions. Alternatively, ShareTrace could be defined more abstractly as sensing infection risk through interactions and user symtpoms. In this case, ShareTrace would follow a multi-tasked execution model, where the task of sensing user symptoms is achieved through user reporting or bodily sensors.

1.3.1.2 Application Users

User Recruitment *Volunteer-based recruitment* is when citizens can "join a sensing campaign for personal interests…or willingness to help the community," while *incentive-based recruitment* promotes participation and offers control over the recruitment rate…These strategies are not mutually exclusive and users can first volunteer and then be rewarded for quality execution of sensing tasks" [10]. ShareTrace assumes volunteer-based recruitment. However, as a decentralized application (dApp) that aligns with the principles of self-sovereignty, an incentive structure that rewards users with verifiable, high-quality data is plausible.

User Selection *User-centric selection* is when “contributions depend only on participants[’] willingness to sense and report data to the central collector, which is not responsible for choosing them.” *Platform-centric selection* is “when the central authority directly decides data contributors...Platform centric decisions are taken according to the utility of data to accomplish the targets of the campaign” [10]. ShareTrace employs user-centric selection, because the purpose of the application is to passively sense the user’s interactions and provide them with the knowledge of their infection risk.

User Type A *contributor* “reports data to the MCS platform with no interest in the results of the sensing campaign” and “are typically driven by incentives or by the desire to help the scientific or civil communities.” A *consumer* joins “a service to obtain information about certain application scenario[s] and have a direct interest in the results of the sensing campaign” [10]. For ShareTrace, users can be either a consumer or a contributor but are likely biased toward the former.

1.3.2 Data Layer

1.3.2.1 Data Management

Data Storage *Centralized storage* involves data being “stored and maintained in a single location, which is usually a database made available in the cloud. This approach is typically employed when significant processing or data aggregation is required.” *Distributed storage* “is typically employed for delay-tolerant applications, i.e., when users are allowed to deliver data with a delayed reporting" [10]. For sensing human interaction, ShareTrace relies on distributed storage in the form of Dataswift Personal Data Accounts. Moreover, ShareTrace is delay-tolerant, so distributed storage is most appropriate. However, for reporting the population-level risk distribution, it is likely that centralized storage would be used.

Data Format *Structured data* is standardized and readily analyzable. *Unstructured data*, however, requires significant processing before it can be used [10]. ShareTrace deals with structured data (e.g., user symptoms, actor URIs).

Data Dimensionality *Single-dimension data* typically occurs when a single sensor is used, while *multi-dimensional data* arises with the use of multiple sensors. ShareTrace data is one-dimensional because it only uses Bluetooth to sense nearby users.

1.3.2.2 Data Processing

Data Pre-processing *Raw data output* implies that no modification is made to the sensed data. *Filtering and denoising* entail "removing irrelevant and redundant data. In addition, they help to aggregate and make sense of data while reducing at the same time the volume to be stored" [10]. ShareTrace only retains the actor URIs that correspond to valid contacts (i.e., lasting at least 15 minutes).

Data Analytics *Machine learning* (ML) *and data mining analytics* are not real-time. They "aim to infer information, identify patterns, or predict future trends." On the contrary, *real-time analytics* consist of "examining collected data as soon as it is produced by the contributors" [10]. ShareTrace aligns with the former category since it aims to infer the infection risk of users.

Data Post-processing *Statistical post-processing* "aims at inferring proportions given quantitative examples of the input data. *Prediction post-processing* aims to determine "future outcomes from a set of possibilities when given new input in the system" [10]. ShareTrace applies predictive post-processing via risk propagation.

1.3.3 Communication Layer

1.3.3.1 Communication Technology

Infrastructured Technology *Cellular* connectivity "is typically required from sensing campaign[s] that perform real-time monitoring and require data reception as soon as it is sensed." *WLAN* "is used mainly when sensing organizers do not specify any preferred reporting technologies or when the application domain permits to send data" at "a certain amount of time after the sensing process" [10]. Infrastructured technology is also referred to as the *infrastructured transmission paradigm* [38]. ShareTrace does not require cellular infrastructure, because it is delay-tolerant and thus only requires WLAN.

Infrastructure-less Technology *Infrastructure-less technologies* "consists of device-to-device (D2D) communications that do not require any infrastructure...but rather allow devices in the vicinity to communicate directly." Technologies include *WiFi-Direct*, *LTE-Direct*, and *Bluetooth* [10]. Infrastructureless technology is also called the *opportunistic transmission paradigm* [38]. ShareTrace uses Bluetooth because of its energy efficiency and short range.

1.3.3.2 Data Reporting

Upload mode With *relay uploading*, "data is delivered as soon as collected." *Store-and-forward* "is typically used in delay-tolerant applications when campaigns do not need to receive data in real-time" [10]. Because ShareTrace is delay-tolerant, it uses store-and-forward uploading.

Metholodgy *Individualized sensing* is "when each user accomplishes the requested task individually and without interaction with other participants." *Collaborative sensing* is when "users communicate with each other, exchange data[,] and help themselves in accomplishing a task or delivering information to the central collector. Users are typically grouped and exchange data exploiting short-range communication technologies, such as WiFi-[D]irect or Bluetooth...Note that systems that create maps merging data from different users are considered individual because users do not interact between each other to contribute" [10]. The methodology of sensing is similar to the *sensing scale* which is typically dichotomized as *personal* [35, 21] (i.e., individualized) and *community* [21] or *group* [35] (i.e., collaborative). ShareTrace is inherently collaborative, relying on mobile devices to exchange actor URIs to estimate infection risk. Thus, collaborative sensing is used.

Timing *Timing* is based on whether devices "should sense in the same period or not." *Synchronous timing* "includes cases in which users start and accomplish at the same time the sensing task. For synchronization purposes, participants communicate with each other." *Asynchronous timing* occurs "when users perform sensing activity not in time synchronization with other users" [10]. ShareTrace requires synchronous timing, because contact sensing inherently requires synchronous communication between the involved devices.

1.3.4 Sensing Layer

1.3.4.1 Sensing Elements

Sensor Deployment *Dedicated deployment* involves the use of "non-embedded sensing elements," typically for a specific task. *Non-dedicated deployment* utilizes sensors that "do not require to be paired with other devices for data delivery but exploit the communication capabilities of mobile devices" [10]. ShareTrace relies on non-dedicated deployment since it relies on Bluetooth that is ubiquitous in modern-day mobile devices.

Sensor Activity *Always-on sensors* "are required to accomplish mobile devices['] basic functionalities, such as detection of rotation and acceleration... Activity recognition [i.e., context awareness]...is a very important feature that accelerometers enable." *On-demand sensors* "need to be switched on by users or exploiting an application running in the background. Typically, they serve more complex applications than always-on sensors and consume a higher amount fo energy" [10]. ShareTrace uses Bluetooth, which may be considered on-demand. While energy efficient, users do control when it is enabled. Ideally, ShareTrace would also use always-on sensors to enable Bluetooth with context awareness (i.e., that the user is carrying or nearby the device).

Acquisition *Homogeneous acquisition* “involves only one type of data and it does not change from one user to another one,” while *heterogeneous acquisition* “involves different data types usually sampled from several sensors” [10]. ShareTrace is homogeneous, because all users sense the same data from one type of sensor.

1.3.4.2 Data Sampling

Sampling Frequency *Continuous sensing* “indicates tasks that are accomplished regularly and independently [of] the context of the smartphone or the user['s] activities.” *Event-based sensing* is “data collection [that] starts after a certain event has occurred. In this context, an event can be seen as an active action from a user or the central collector, but also a given context awareness” [10]. ShareTrace sensing is continuous but would ideally be event-based to conserve device energy.

Sensing Responsibility When the *mobile device* is responsible, “devices or users take sampling decisions locally and independently from the central authority...When devices take sampling decisions, it is often necessary to detect the context [of the] smartphones and wearable devices...The objective is to maximize the utility of data collection and minimize the cost of performing unnecessary operations.” When the *central collector* is responsible, they make "decisions about sensing and communicate them to the mobile devices" [10]. Given the human-centric nature of the ShareTrace sensing task, mobile devices are responsible.

User involvement *Participatory involvement* "requires active actions from users, who are explicitly asked to perform specific tasks. They are responsible to consciously meet the application requests by deciding when, what, where, and how to perform sensing tasks." *Opportunistic involvement* means that "users do not have direct involvement, but only declare their interest in joining a campaign and providing their sensors as a service. Upon a simple handshake mechanism between the user and the MCS platform, a MCS thread is generated on the mobile device (e.g., in the form of a mobile app), and the decisions of what, where, when, and how to perform the sensing are delegated to the corresponding thread. After having accepted the sensing process, the user is totally unconscious with no tasks to perform and data collection is fully automated...The smartphone itself is context-aware and makes decisions to sense and store data, automatically determining when its context matches the requirements of an application. Therefore, coupling opportunistic MCS systems with context-awareness is a crucial requirement" [10]. Earlier works on MCS refer to user involvement as the *sensing paradigm* [35, 21, 38]. ShareTrace is opportunistic, ideally with context-awareness.

1.4 System Model

The system model is similar to that in previous work [3, 2]. **??** illustrates the corresponding data flow[3].

- Each user owns a *personal data store* (PDS), a form of cloud storage that empowers the user with ownership and access control over their data.
- Symptom scores are computed in a user's PDS to support integrating multiple streams of personal data [3]. While local symptom-score computation [3, 2] is more privacy-preserving, it is assumed that the user's PDS is a trusted entity.
- User device interactions serve as a proxy for proximal human interactions. This work does not assume a specific protocol, but does assume that the protocol can approximate the duration of contact with relative accuracy and that communication with the actors of those contacted users can be established in a privacy-preserving manner.
- No geolocation data is collected [3]. As a decentralized, proximity-based solution, it is not necessary to collect user geolocation data. See Appendix A.5 for a discussion of a geolocation-based design that was considered.

[3] A *data-flow diagram* consists of data processors (circles), directed data flow (arrows), data stores (parallel lines), and external entities (rectangles) [19, pp. 437–438].

Application	Task	Scheduling	Proactive	●
			Reactive	
		Assignment	Centralized	
			Decentralized	●
		Execution	Single task	●
			Multi-tasking	
	User	Recruitment	Voluntary	●
			Incentivized	
		Selection	Platform-centric	
			User-centric	●
		Type	Consumer	●
			Contributor	●
Data	Management	Storage	Centralized	
			Distributed	●
		Format	Structured	●
			Unstructured	
		Dimension	Single dimension	●
			Multi-dimensional	
	Processing	Pre-processing	Raw data	
			Filtering and denoising	●
		Analytics	ML and data mining	●
			Real-time	
		Post-processing	Statistical	
			Prediction	●
Communication	Technologies	Infrastructured	Cellular	●
			WLAN	●
		Infrastructure-less	LTE-Direct	
			WiFi-Direct	
			Bluetooth	●
	Reporting	Upload mode	Relay	
			Store and forward	●
		Methodology	Individual	
			Collaborative	●
		Timing	Synchronous	●
			Asynchronous	
Sensing	Elements	Deployment	Dedicated	
			Non-dedicated	●
		Activity	Always-on	●
			On-demand	○
		Acquisition	Homogeneous	●
			Heterogeneous	
	Sampling	Frequency	Continuous	●
			Event-based	○
		Responsibility	Mobile device	●
			Central collector	
		User involvement	Participatory	
			Opportunistic	●

Table 1.1: ShareTrace classification using the four-layered architecture of a mobile crowdsensing application [10]. Always (●); with context-awareness (○).

Appendix A

Previous Designs and Implementations

Before working on ShareTrace, I did not have experience developing distributed algorithms. The approach proposed in Chapter 1 is my *fifth* attempt at defining a performant implementation of risk propagation that is also decentralized and online. The prior four attempts offered valuable learnings that guided me toward the proposed approach; however, only the latter supports truly decentralized, privacy-preserving contact tracing. To document my efforts in developing this thesis, prior designs and implementations are provided in this appendix.

A.1 Thinking Like a Vertex

The first iteration of risk propagation[1] utilized Apache Giraph[2], an open-source version of the iterative graph-processing library, Pregel [40], which is based on the bulk synchronous parallel model of distributed computing [53]. Giraph follows the *"think like a vertex" paradigm* in which the algorithm is specified in terms of the local information available to a graph vertex [41].

Risk propagation was implemented as defined by Ayday et al. [3] and Ayday, Yoo, and Halimi [2], using the factor graph representation of the contact network. Moreover, the implementation assumed the use of Dataswyft Personal Data Accounts[3], which provide a data-oriented interface to self-sovereign identity [48, pp. 98–99]. However, because the Exposure Notification API developed by Apple[4] and Google[5] does not permit remotely persisting ephemeral identifiers, the implementation assumed that user geolocation data would be analyzed to generate the factor vertices in the factor graph (Appendix A.5). Figure A.1 describes the high-level architecture. Callouts 1, 2, and 4 were implemented using a fan-out design in which a *ventilator* Lambda function divides the work amongst *worker* Lambda functions.

[1] `https://github.com/cwru-xlab/sharetrace-giraph`

[2] `https://giraph.apache.org`

[3] `https://www.dataswyft.io`

[4] `https://covid19.apple.com/contacttracing`

[5] `https://www.google.com/covid19/exposurenotifications`

[6] `https://aws.amazon.com/lambda`

[7] `https://aws.amazon.com/s3`

[8] `https://aws.amazon.com/emr`

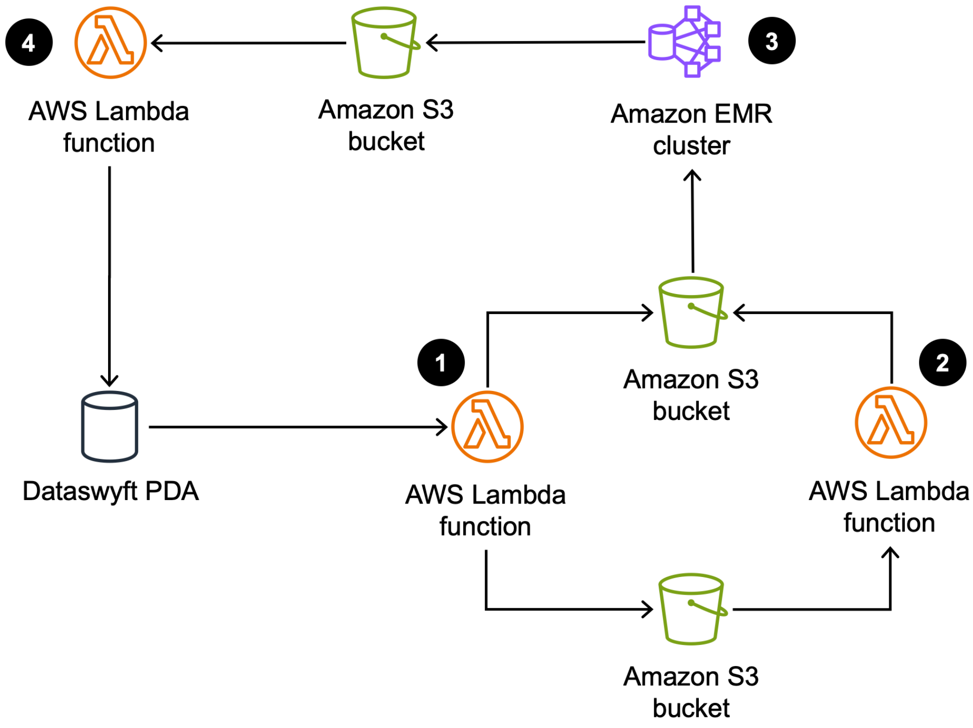

Figure A.1: ShareTrace batch-processing architecture. ❶ An AWS Lambda function[6] retrieves the recent risk scores and location data from the Dataswyft Personal Data Accounts (PDAs) of ShareTrace users. Risk scores are formatted as Giraph vertices and stored in an Amazon Simple Storage Service[7] (S3) bucket. Location data is stored in a separate S3 bucket. ❷ A Lambda function performs a contact search over the location data and stores the contacts as Giraph edges in the same bucket that stores the Giraph vertices. ❸ Amazon Elastic MapReduce[8] (EMR) runs risk propagation as a Giraph job and stores the exposure scores in an S3 bucket. ❹ A Lambda function stores the exposure score of each user in their respective PDA.

A couple factors prompted me to search for an alternative implementation.

1. *Dependency management incompatibility.* The primary impetus for reimplementation was the dependency conflicts between Giraph and other libraries. Despite several attempts (e.g., using different library versions, using different versions of Giraph, and forcing specific versions of transitive dependencies) to resolve the conflicts, a lack of personal development experience and stalled progress prompted me pursue alternative implementations.

2. *Implementation complexity.* For a relatively straightforward data flow, the architecture in Figure A.1 corresponded to over 4,000 lines of source code. In retrospect, AWS Step Functions[9] could have been used to orchestrate the workflow, including the fan-out design pattern, which would have simplified the Lambda function implementations. Regarding the implementation of risk propagation, one-mode projection (first used in Appendix A.4) would have simplified the implementation since it avoids types of vertices and messages.

A.2 Factor Subgraph Actors

In an attempt to simplify the design in Appendix A.1, I rewrote risk propagation using the Ray Python library[10]. While it claims to support actor-based programming, Ray only offers coarse-grained concurrency, with each actor being mapped to a physical core. To achieve parallelism, the factor graph was

[9] `https://aws.amazon.com/step-functions`

[10] `https://www.ray.io`

partitioned amongst the actors such that each actor maintained a subset of variable vertices *or* factor vertices. The graph topology was stored in shared memory since it was immutable. The lifetime of this design was brief for the following reasons.

1. *Poor performance.* Communication between Ray actors requires message serialization. Moreover, partitioning the factor graph into subsets of factor vertices and variable vertices results in maximal interprocess communication. Unsurprisingly, this choice of partitioning manifested in poor runtime performance.

2. *Design complexity.* Not using a framework, like Giraph, meant that this implementation required more low-level code to implement actor functionality and message passing. Regardless of the performance, the overall design of this implementation was poorly organized and overthought.

A.3 Driver-Monitor-Worker Framework

Based on the poor runtime performance and complexity of the previous approach, I speculated that centralizing the mutable aspects of risk propagation (i.e., the iterative exposure scores of each variable vertex) would improve both metrics. With this is mind, I designed the *monitor-worker-driver* (MWD) *framework*, which draws inspiration from the *tree of actors* design pattern[11]. Figure A.2 describes the framework.

[11] `https://docs.ray.io/en/latest/ray-core/patterns/tree-of-actors.html`

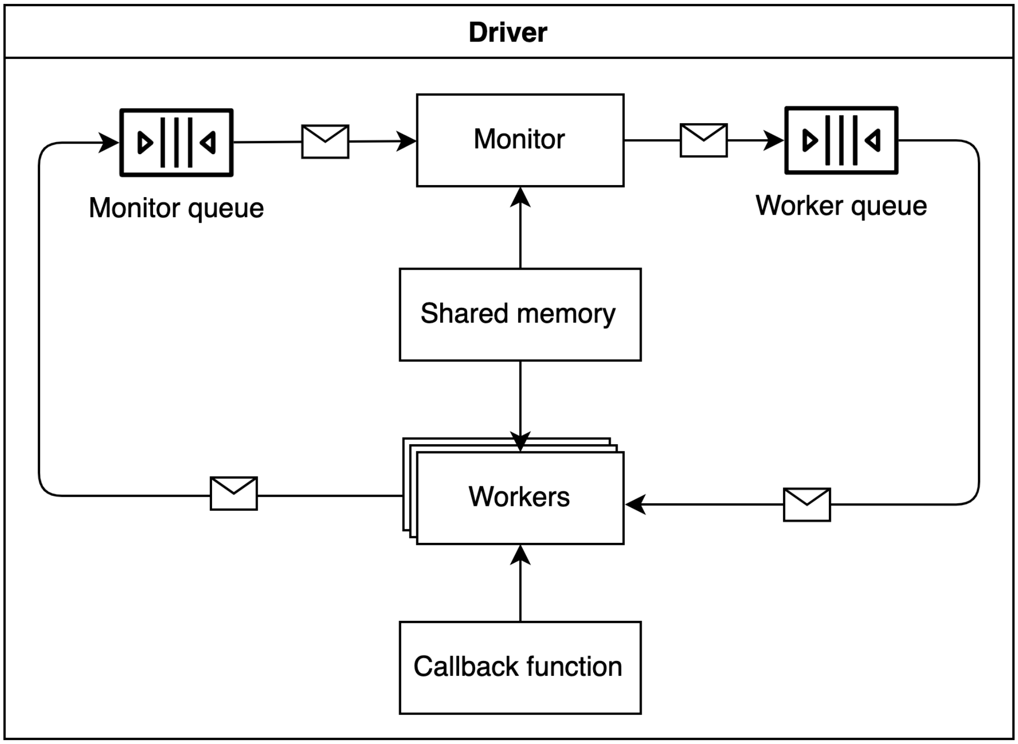

Figure A.2: Monitor-worker-driver framework. The *monitor* encapsulates the mutable state of the program and decides which messages are processed by workers. A *worker* is a stateless entity that processes the messages that the monitor puts in the *worker queue*. Worker behavior is defined by a *callback function*, which typically depends on the message contents. The side effects of processing a message are recorded as new messages and put in the *monitor queue*. This cycle repeats until some termination condition is satisfied. Any immutable state of the program can be stored in *shared memory* for efficient access. The *driver* is the entry point into the program. It initializes the monitor and workers, and then waits for termination.

For risk propagation, the driver creates the factor graph from the set of risk scores S and contacts C, stores the factor graph in shared memory, sets the initial state of the monitor to be the maximum risk score of each individual, and puts all risk scores in the monitor queue. During message passing, the monitor maintains the exposure score for each variable vertex.

The MWD framework was the first approach that utilized the send coefficient to ensure the convergence and termination of message passing. However, because the MWD-based implementation assumed the factor graph representation of the contact network, the send coefficient was applied to both variable and factor messages.

Compared to Appendix A.2, this implementation provided a cleaner design and less communication overhead. However, what prompted (yet another) an alternative implementation was its scalability. Because the monitor processes messages serially, it is a bottleneck for algorithms in which the workers perform fine-grained tasks. Indeed, the Ray documentation[12] notes that the parallelization of small tasks is an anti-pattern because the interprocess communication cost exceeds the benefit of multiprocessing. Unfortunately, the computation performed by factor vertices and variable vertices was fine-grained, so the scalability of the MWD framework was demonstrably poor.

[12] `https://docs.ray.io/en/latest/ray-core/patterns/too-fine-grained-tasks.html`

A.4 Projected Subgraph Actors

The last alternative design of risk propagation is the subject of published work [52] and preprint [51]. Since those works were written during the coarse of my graduate studies, algorithmic details and experimental results are included below.

A.4.1 Risk Propagation

Like all previous designs, risk propagation is an offline algorithm. Let n be the number of actors, where G_i represents the i-th subgraph of contacts that is associated with the i-th actor. It is assumed that actor communication is expensive, so a partitioning algorithm [9] that minimizes communication complexity between actors is key to maximizing performance. Each actor is associated with a *remote mailbox* and a *local mailbox*. The former is designated for messages sent by other actors, while the latter is designated for messages associated with individuals within the same subgraph. For an actor to send a message to another actor, it must know the address of the target actor's remote mailbox and an identifier of the individual within the target actor's subgraph. The Risk-Propagation-Main operation describes the main steps.

```
Risk-Propagation-Main($S, C$)
$G \leftarrow$ Contact-Network($C$)
for each $i \in [1 \,..\, n]$
    Risk-Propagation-Actor($G_i, S_i$)
Collect the exposure scores from all actors
```

The Risk-Propagation-Actor operation defines the behavior of an actor. According to Tatton et al. [51, 52], an actor terminates when no message has been received after a set period of time. Tatton et al. [51] also allows an actor to terminate after passing messages for a maximum duration; or if a certain number of messages have been received and none caused an individual's exposure score to be updated.

```
Risk-Propagation-Actor$(G, S)$
for each $v_i \in V$
    $\tilde{s}_{t,i}, \dot{s}_{t,i}, \leftarrow \max S_i$
    for each $v_j \in N_i$
        $s_t \leftarrow \arg\max \left\{ s_t^{\lambda(t - t_{ij})} \;\middle|\; s_t \in S_i, t < t_{ij} + \beta \right\}$
        Send $\alpha s_t$ to $v_j$
while termination condition is not satisfied
    Receive $s_t$ for $v_i$ from $v_j$
    $\tilde{s}_{t,i} \leftarrow \max\{\tilde{s}_{t,i}, s_t\}$
    for each $v_k \in N_i \setminus \{v_j\}$
        if $t < t_{ik} + \beta$ and $s_t \geq \gamma \dot{s}_i$ and $t \leq \dot{t}_i$
            Send $\alpha s_t$ to $v_k$
```

Compared to asynchronous risk propagation (Section 1.2), actor behavior differs in two significant ways:

1. The send threshold is applied *across* contacts and discriminates against risk scores that are newer than the initial exposure score $\dot{s}_t$. The latter is based on the assumption that newer risk scores are less likely to be propagated by other actors.

2. The initial risk score to sent to each contact accounts for recency, which is assumed to exponentially decay with a rate constant of $\lambda > 0$.

As such, message reachability differs from (1.4):

$$r(u,v) = \sum_{(i,j)\in P} [\alpha^i \dot{s}_u \geq \gamma\alpha\dot{s}_i] \cdot [\dot{t}_u \leq \dot{t}_i] \cdot [\dot{t}_u < t_{ij} + \beta],$$

A.4.2 Experiment Design

The implementation of risk propagation is available on GitHub[13].

Unless stated otherwise, it is assumed that the transmission rate is $\alpha = 0.8$; the send coefficient is $\gamma = 0.6$; the time buffer is $\beta = 2$ days; the risk score and contact expiries are $T_s = T_c = 14$ days; and the rate constant is $\lambda = 1$. Actors were configured to timeout after 3 seconds of not receiving any messages; or after receiving $10n$ messages that did not prompt an update to an individual's exposure score, where n is the number of vertices in the contact network.

To partition the contact network, the METIS algorithm[14] [30] was configured to use k-way partitioning with a load imbalance factor of 0.2; to attempt contiguous partitions that have minimal inter-partition connectivity; to apply 10 iterations of refinement during each stage of the uncoarsening process; and to use the best of 3 cuts.

The Case Western Reserve University high-performance computing cluster was utilized. For efficiency experiments, 4 CPUs and 8GB RAM were used. For scalability experiments, 8–12 CPUs and 16–64GB RAM were used. All

[13] `https://github.com/cwru-xlab/sharetrace-ray`

[14] `https://github.com/inducer/pymetis`

experiments were run on a single cluster node.

A.4.2.1 Synthetic Graphs

We evaluate the scalability and efficiency of risk propagation on three types of graphs: a random geometric graph [RGG] [14], a benchmark graph [LFRG] [34], and a clustered scale-free graph [CSFG] [28]. Together, these graphs demonstrate some aspects of community structure [17] which allows us to more accurately measure the performance of risk propagation. When constructing a RGG, we set the radius to $r(n) = \min\left(1, 0.25^{\log_{10}(n)-1}\right)$, where n is the number of users. This allows us to scale the size of the graph while maintaining reasonable density. We use the following parameter values to create LFRGs: mixing parameter $\mu = 0.1$, degree power-law exponent $\gamma = 3$, community size power-law exponent $\beta = 2$, degree bounds $(k_{\min}, k_{\max}) = (3, 50)$, and community size bounds $(s_{\min}, s_{\max}) = (10, 100)$. Our choices align with the suggestions by Lancichinetti, Fortunato, and Radicchi [34] in that $\gamma \in \mathbb{R}_{[2,3]}$, $\beta \in \mathbb{R}_{[1,2]}$, $k_{\min} < s_{\min}$, and $k_{\max} < s_{\max}$. To build CSFGs, we add $m = 2$ edges for each new user and use a triad formulation probability of $P_t = 0.95$. For all graphs, we remove self-loops and isolated users.

The following defines our data generation process. Let p be the probability of a user being "high risk" (i.e., $r \geq 0.5$) Then, with probability $p = 0.2$, we sample $L + 1$ values from the uniform distribution $\mathbb{U}_{[0.5,1)}$. Otherwise, we sample from $\mathbb{U}_{[0,0.5)}$. This assumes symptom scores and exposure scores are computed daily and includes the present day. We generate the times of these risk scores by sampling a time offset $t_{\text{off}} \sim \mathbb{U}_{[0\text{s};86{,}400\text{s}]}$ for each user such that

$t_d = t_{\text{now}} + t_{\text{off}} - d$ days, where $d \in \mathbb{N}_{[0,L]}$. To generate a contact times, we follow the same procedure for risk scores, except that we randomly sample one of the $L + 1$ times and use that as the contact time.

We evaluate various transmission rates and send coefficients:

$$(\gamma, \alpha) \in \{0.1, 0.2, \ldots, 1\} \times \{0.1, 0.2, \ldots, 0.9\}.$$

For all γ, α, we set $n = 5,000$ and $K = 2$.

To measure the scalability of risk propagation, we consider $n \in \mathbb{N}_{[10^2,10^4]}$ users in increments of 100 and collect 10 iterations for each n. The number of actors we use depends on n such that $K(n) = 1$ if $n < 10^3$ and $K(n) = 2$ otherwise. Increasing K for our choice of n did not offer improved performance due to the communication overhead.

A.4.2.2 Real-World Graphs

We analyze the efficiency of risk propagation on three real-world contact networks that were collected through the SocioPatterns collaboration. Specifically, we use contact data in the setting of a high school [Thiers13] [18], a workplace [InVS15], and a scientific conference [SFHH] [**Genois2018**]. Because of limited availability of large-scale contact networks, we do not use real-world contact networks to measure the scalability of risk propagation.

To ensure that all risk scores are initially propagated, we shift all contact times forward by τ and use ($\tau - 1$ day) when generating risk scores times. In this way, we ensure the most recent risk score is still older than the first contact time. Risk score values are generated in the same manner as described in Section A.4.2.1 with the exception that we only generate one score. Lastly, we perform 10 iterations over each data set to obtain an average performance.

A.4.3 Experimental Results

A.4.3.1 Efficiency

Prior to measuring scalability and real-world performance, we observed the effects of send coefficient and transmission rate on the efficiency of risk propagation. As ground truth, we used the maximum update count for a given transmission rate. Fig. A.3 indicates that a send coefficient of $\gamma = 0.6$ permits 99% of the possible updates. Beyond $\gamma = 0.6$, however, the transmission rate has considerable impact, regardless of the graph. As noted in Section 1.2.2, send coefficient quantifies the trade-off between completeness and efficiency. Thus, $\gamma = 0.6$ optimizes for both criteria.

Unlike the update count, Fig. A.3 shows a more variable relationship with respect to runtime and message count. While, in general, transmission rate (send coefficient) has a direct (resp. inverse) relationship with runtime and message count, the graph topology seems to have an impact on this fact. Namely, the LFRG displayed less variability across send coefficient and transmission rate than the RGG and CSFG, which is the cause for the large interquartile ranges. Therefore, it is useful to consider the lower quartile Q_1, the median Q_2, and the upper quartile Q_3. For $\alpha = 0.8$ and $\gamma = 0.6$, risk propagation is more efficient with $(Q_1, Q_2, Q_3) = (0.13, 0.13, 0.46)$ normalized runtime and $(Q_1, Q_2, Q_3) = (0.13, 0.15, 0.44)$ normalized message count.

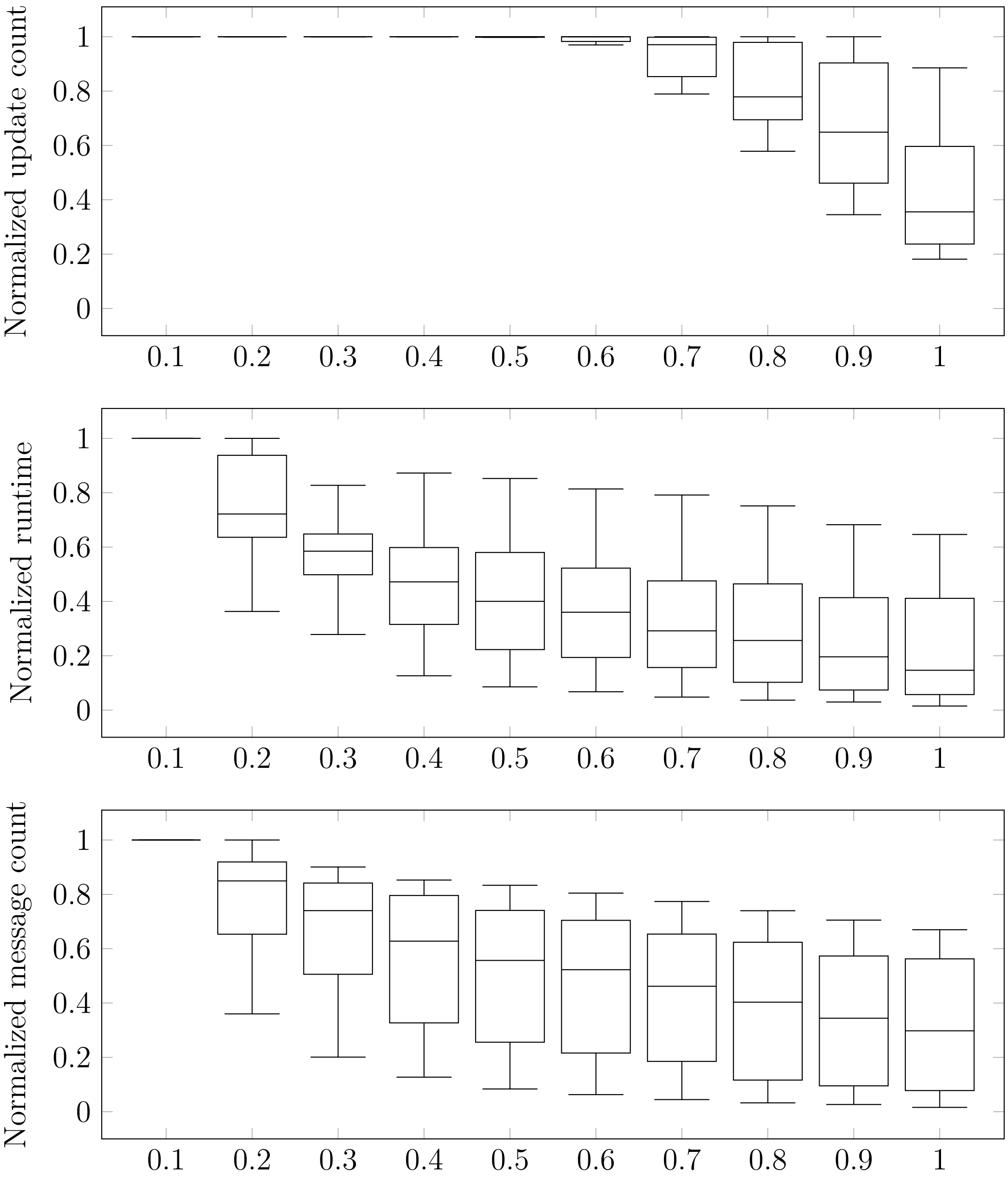

Figure A.3: Effects of send coefficient on efficiency. All dependent variables are normalized across graphs and transmission rates.

A.4.3.2 Message Reachability

To validate the accuracy of Equation (1.5), we collected values of Equation (1.4) and Equation (1.5) for real-world and synthetic graphs. For the latter set of graphs, we observed reachability while sweeping across values of γ and α.

To measure the accuracy of Equation (1.5), let the *message reachability ratio* (MRR) be defined as

$$\mathrm{mrr}(u) := \frac{r(u)}{\hat{r}(u)}. \tag{A.1}$$

Overall, Equation (1.5) is a good estimator of Equation (1.4). Across all synthetic graphs, Equation (1.5) modestly underestimated Equation (1.4) with quartiles $(Q_1, Q_2, Q_3) = (0.71, 0.84, 0.98)$ for the Equation (A.1). For $\alpha = 0.8$ and $\gamma = 0.6$, the quartiles of Equation (A.1) were $(Q_1, Q_2, Q_3) = (0.52, 0.77, 1.12)$ and $(Q_1, Q_2, Q_3) = (0.79, 0.84, 0.93)$, respectively. Table A.1 provides mean values of Equation (A.1) for both synthetic and real-world graphs. Fig. A.4 indicates that moderate values of γ tend to result in a more stable MRR, with lower (higher) γ underestimating (resp. overestimating) Equation (1.4). With regard to transmission rate, Equation (A.1) tends to decrease with increasing α, but also exhibits larger interquartile ranges.

Because Equation (1.5) does not account for the temporality constraints **??** and **??**, it does not perfectly estimate Equation (1.4). With lower γ and higher α, Equation (1.5) suggests higher MR. However, because a message is only passed under certain conditions (see Algorithm **??**), this causes Equation (1.5) to overestimate Equation (1.4). While Equation (1.5) theoretically is an upper bound on Equation (1.4), it is possible for Equation (1.5) to underestimate Equation (1.4) if the specified value of $s_0(v)$ overestimates the true value of $s_0(v)$. When computing Equation (A.1) for Fig. A.4, we used the mean $s_0(v)$ across all users v, so $\mathrm{mrr}(u) > 1$ in some cases.

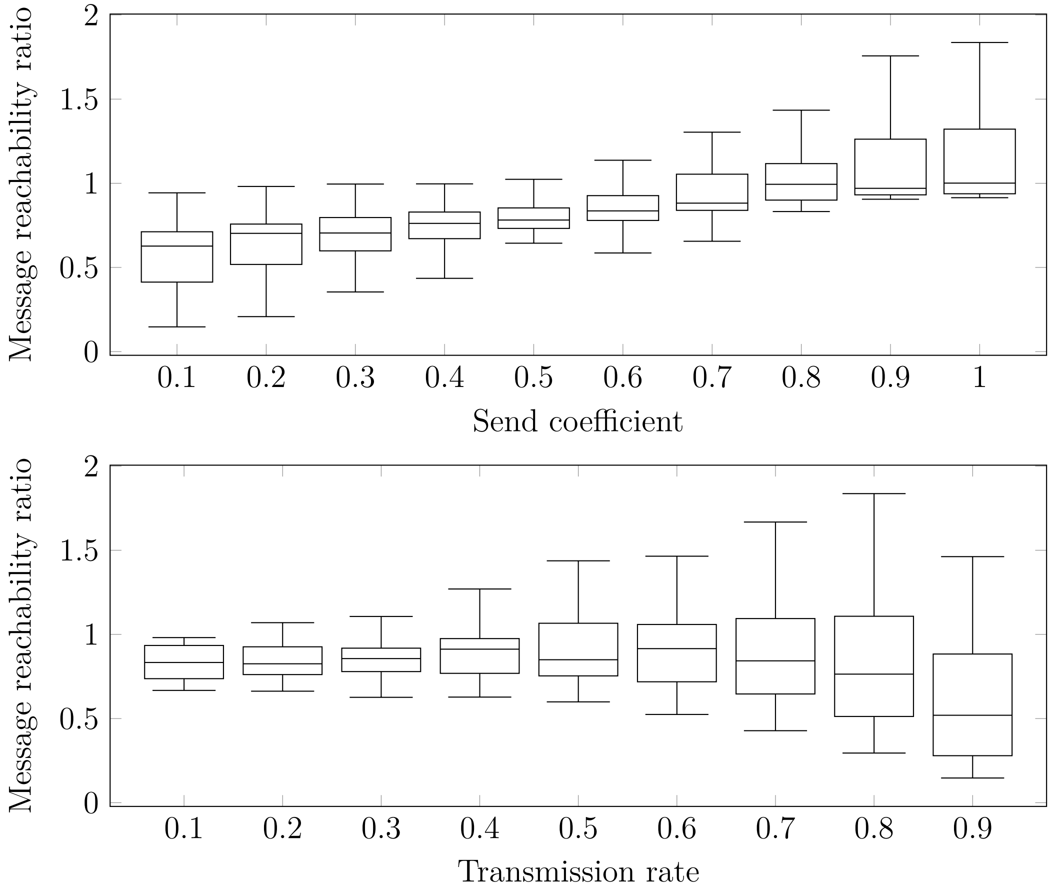

Figure A.4: Effects of send coefficient and transmission rate on the message reachability ratio. Independent variables are grouped across graphs.

Setting	$\mathrm{mrr}(u) \pm 1.96 \cdot \mathrm{SE}$
Synthetic	
LFR	0.88 ± 0.14
RGG	0.74 ± 0.12
CSFG	0.90 ± 0.14
	$\mathbf{0.85 \pm 0.08}$
Real-world	
Thiers13	0.58 ± 0.01
InVS15	0.63 ± 0.01
SFHH	0.60 ± 0.01
	$\mathbf{0.60 \pm 0.01}$

Table A.1: Message reachability ratio for synthetic and real-world graphs ($\alpha = 0.8$, $\gamma = 0.6$). Synthetic (real-world) ratios are averaged across parameter combinations (resp. runs).

A.4.3.3 Scalability

Fig. A.5 describes the runtime behavior of risk propagation. The runtime of CSFGs requires further investigation. A linear regression fit explains ($R^2 = 0.52$) the runtime of LFRGs and RGGs with a slope $m = (1.1 \pm 0.1) \cdot 10^{-3}$ s/contact and intercept $b = 4.3 \pm 1.6$s ($\pm 1.96 \cdot \mathrm{SE}$).

A.5 Location-Based Contact Tracing

- Motivation: Google/Apple API prevents exporting Bluetooth EphIDs
- Other location-based contact tracing approaches

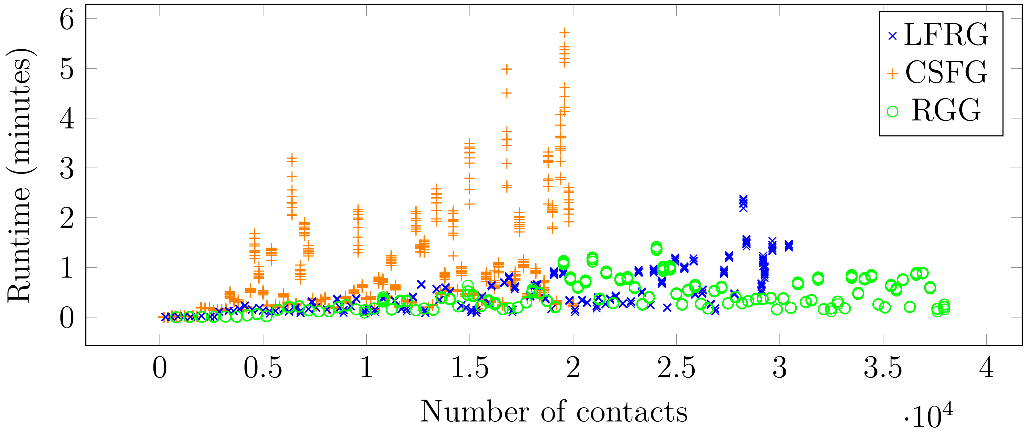

Figure A.5: Runtime of risk propagation on synthetic graphs containing 100–10,000 users and approximately 200–38,000 contacts.

A.5.1 System Model

The system model is very similar to Ayday et al. [3] and Ayday, Yoo, and Halimi [2] and designs. The only difference is that user geolocation data is collected instead of Bluetooth ephemeral identifiers.

Geohashing is a public-domain encoding system that maps *geographic coordinates* (i.e., latitude-longitude ordered pairs [50, p. 5]) to alphanumeric strings called *geohashes*, where the length of a geohash is correlated with its geospatial precision [45]. To offer some basic privacy, a user's precise geolocation history is obfuscated on-device by encoding geographic coordinates as geohashes with 8-character precision which corresponds to a region of 730m^2.

A.5.2 Contact Search

a temporally ordered sequence of timestamped geolocations. It is assumed that

1. geolocation histories are not recorded on a fixed schedule, and
2. a user remains at a geolocation until the next geolocation is recorded.

Finding the most recent contact between two users from their aligned geolocation histories is similar to finding the last k-length common substring between two strings, where each symbol represents a timestamped geolocation. The difference lies in how the start and end of the contact time interval is defined. By assumption 2, the start (end) of a contact time interval is defined as the earlier (ref. later) timestamp of the two first (ref. last) timestamped geolocations in the sequence where the two histories differ. Figure A.6 provides a visual example.

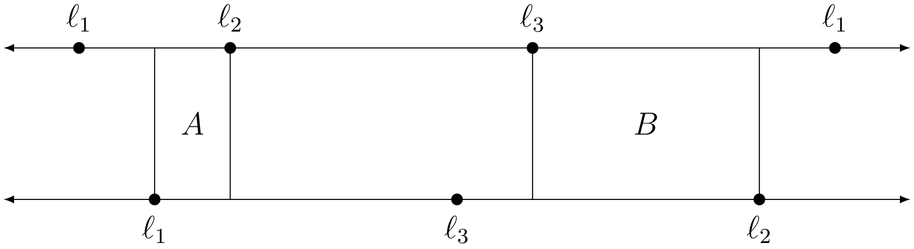

Figure A.6: Contact search with two geolocation histories. Each line denotes time, increasing from left to right. A point ℓ_i is a geolocation and occurs relative in time with respect to the placement of other points. Region B defines the contact interval as it is of sufficient duration and occurs after A.

A.5.2.1 Naive Contact Search

A naive approach to finding all contacts amongst a set of geolocation histories H is to compare all unordered pairs. For a given pair of aligned geolocation histories, the idea is to maintain a pointer to the previous and current index in each history, advancing the pair of pointers whose geolocation occurs later in time. Once a common geolocation is found, all pointers are advanced together until the geolocations differ. If the sequence is δ-contiguous, where a sequence of timestamped geolocations S is *δ-contiguous* if $S.length \geq \delta$ and then it is recorded. The latest such sequence is used to define the contact between the two users. Because only the most recent time of contact is of interest, the procedure can be improved by iterating in reverse and then terminating once a sequence is found. Regardless, this approach takes $\Theta(n^2)$ time, where $n = |H|$, because all $\frac{n(n-1)}{2}$ unique pairs must be considered.

A.5.2.2 Indexed Contact Search

While the **for** loop in Naive-Contact-Search is *embarrassingly parallel* [24, p. 14], the naive approach is neither scalable nor efficient. It can be improved by observing that it is necessary, but not sufficient, that a pair of ϵ-proximal geolocations exists between two geolocation histories for a contact to exist. Therefore, the geolocation histories H can be indexed into a spatial data structure I [**Dinh2010**, 43, 39] and then only consider the geolocation-history pairs that share at least one ϵ-proximal geolocation pair. This approach is described by the Indexed-Contact-Search operation.

Line 2 executes a fixed-radius near-neighbors search (FR-NNS) [5, 7] for each geolocation in the spatial index I. Formally, given a set of geolocations $L \subseteq \mathbb{L}$, a metric d, and a distance ϵ, the *fixed-radius near-neighbors* of a geolocation $\ell \in L$ is defined as the subset of ϵ-proximal geolocations [7],

$$N(\ell) = \{\ell' \in L \mid d(\ell, \ell') \leq \epsilon\}$$

Note that the set of neighbors $N(i)$ of user i corresponds to the geolocations that are ϵ-proximal to *any* of the geolocations in their geolocation history H_i,

$$N(i) = \bigcup_{\ell \in H_i} N(\ell).$$

On line 3, the operation Unique-Users maps these near-neighbors back to the associated users, removing any duplicates that may arise from mapping multiple geolocations to the same user. Finally, line 4 maps the set of users U back to their geolocation histories and runs Naive-Contact-Search on the resultant subset.

```
Indexed-Contact-Search($H$)
1: $I \leftarrow$ Spatially-Index($H$)
2: $N \leftarrow$ Fixed-Radius-Near-Neighbors($I, \epsilon$)
3: $U \leftarrow$ Unique-Users($N, H$)
4: return Naive-Contact-Search($\{H_i \in H \mid i \in U\}$)
```

To carry out FR-NNS, one approach is to use a *ball tree*, a complete binary tree that associates with each node a hypersphere that contains a subset of the data [**Neeraj2008**, 46, 31]. Any metric can be used to perform FR-NNS on a ball tree. However, because geolocation is represented as geographic coordinates, metrics that assume a Cartesian coordinate system may be unsuitable. One of the simplest geometric models of the Earth is that of a sphere. Given two geographic coordinates, the problem of finding the length of the geodesic[15] between them is known as the *inverse geodetic problem* [**Sjoberg2012**]. As-

[15] The *geodesic* is the shortest segment between two points on an ellipsoid [37].

suming a spherical Earth, the solution to the inverse problem is to find the length of the segment that joins the two points on a great circle[16].

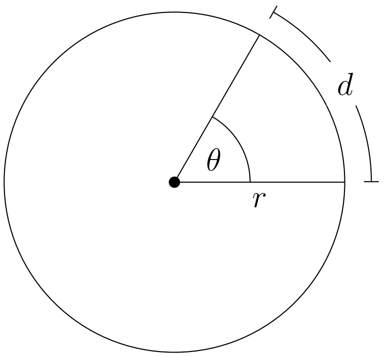

Figure A.7: Central angle of a great circle.

The *haversine*, or the half "versed" (i.e., reversed) sine, of a central angle θ is defined as

$$\operatorname{hav}\theta = \frac{\operatorname{vers}\theta}{2} = \frac{1-\cos\theta}{2} = \sin^2\frac{\theta}{2}. \tag{A.2}$$

The great-circle distance d between two points can be found by inversing (A.2) and solving,

$$d(\ell_i, \ell_j) = 2 \cdot \arcsin\sqrt{\sin^2\frac{\phi_i - \phi_j}{2} + \cos\phi_i \cdot \cos\phi_j \cdot \sin^2\frac{\lambda_i - \lambda_j}{2}},$$

where $\ell_i = (\phi_i, \lambda_i)$ is a latitude-longitude coordinate in radians [8, pp. 157–162].

The choice of the great-circle distance was primarily driven by its readily available usage in the scikit-learn [**sklearn2013**] implementation of a ball tree. If such an approach for discovering contacts were to be used in practice, more advanced *geodetic datum* [37, pp. 71–130] could be used to provide better geospatial accuracy. Moreover, by projecting geodetic coordinates onto

[16] The *great circle* is the cross-section of a sphere that contains its center [37]

the plane, metrics that assume a Cartesian coordinate system could be used instead [37, pp. 265–326].

Appendix B

Pseudocode Conventions

Pseudocode conventions are mostly consistent with Cormen et al. [12].

- Indentation indicates block structure.

- Looping and conditional constructs have similar interpretations to those in standard programming languages.

- Composite data types are represented as *objects*. Accessing an *atttribute* a of an object o is denoted $o.a$. A variable representing an object is a *pointer* or *reference* to the data representing the object. The special value NIL refers to the absence of an object.

- Parameters are passed to a procedure *by value*: the "procedure receives its own copy of the parameters, and if it assigns a value to a parameters, the change is *not* seen by the calling procedure. When objects are passed, the pointer to the data representing the object is copied, but the attributes of the object are not." Thus, attribute assignment "is visible if the calling procedure has a pointer to the same object."

- A **return** statement "immediately transfers control back to the point of call in the calling procedure."

- Boolean operators **and** and **or** are *short circuiting*.

The following conventions are specific to this work.

- Object attributes may be defined *dynamically* in a procedure.

- Variables are local to the given procedure, but parameters are global.

- The "$\leftarrow$" symbol is used to denote assignment, instead of "$=$".

- The "$=$" symbol is used to denote equality, instead of "$==$", which is consistent with the use of "$\neq$" to denote inequality.

- The "$\in$" symbol is used in **for** loops when iterating over a collection.

- Set-builder notation $\{\, x \in X \mid \textsc{Predicate}(x) \,\}$ is used to create a subset of a collection X in place of constructing an explicit data structure.

Appendix C

Data Structures

Let a *dynamic set* X be a mutable collection of distinct elements. Let x be a pointer to an element in X such that $x.key$ uniquely identifies the element in X. Let a *dictionary* be a dynamic set that supports insertion, deletion, and membership querying [12].

- Insert(X, x) adds the element pointed to by x to X.

- Delete(X, x) removes the element pointed to by x from X.

- Search(X, k) returns a pointer x to an element in the set X such that $x.key = k$; or nil, if no such element exists in X.

- Merge(X, x) adds the element pointed to by x, if no such element exists in X; otherwise, the result of applying a function to x and the existing element is added to X.

- Maximum(X) returns a pointer x to the maximum element of the totally ordered set X; or nil if X is empty.

Appendix D

Actor Model

The *actor model* is a local model of concurrent computing that defines computation as a strict partial ordering of events [25, 26]. An *event* is defined as “a transition from one local state to another” [26]; or, more concretely, “a *message* arriving at a computational agent called an *actor*.” The actor that receives the message of an event is called the *target* of the event.

Upon receipt, an actor may send messages to itself or other actors, update its local state, or create other actors.

May send messages to other actors (Clinger) Update its local state (Clinger) Create other actors

Events caused

An event may *activate* subsequent events

Ordering laws:

Passing messages between actors is the only means of communication. Thus, control and data flow are inseparable in the actor model [27].

A minimal specification of an actor includes its *name* and *behavior*, or how it acts upon receiving a message. Clinger [11] explains,

> An actor's name is a necessary part of its description because two different actors may have the same behavior. An actor's behavior is a necessary part of its description because the same actor may have different behaviors at different times.

acquaintances, or a finite collection of names of the actors that it knows about [27].

Thus, the specification of an actor system consists of the following [25].

- Deciding on the natural kinds of actors (objects) to have in the system to be constructed.
- Deciding for each kind of actor what kind of messages it should receive.
- Deciding for each kind of actor what it should do when it receives each kind of message.

Bibliography

[1] Gul Agha. “Actors: A model of concurrent computation in distributed systems”. PhD thesis. MIT, 1985. URL: `http://hdl.handle.net/1721.1/6952`.

[2] Erman Ayday, Youngjin Yoo, and Anisa Halimi. “ShareTrace: An Iterative Message Passing Algorithm for Efficient and Effective Disease Risk Assessment on an Interaction Graph”. In: *Proc. 12th ACM Con. Bioinformatics, Comput. Biology, Health Inform.* BCB 2021. 2021.

[3] Erman Ayday et al. *ShareTrace: A Smart Privacy-Preserving Contact Tracing Solution by Architectural Design During an Epidemic.* Tech. rep. Case Western Reserve University, 2020. URL: `https://github.com/cwru-xlab/sharetrace-papers/blob/main/sharetace-whitepaper.pdf`.

[4] Alain Barrat and Ciro Cattuto. “Temporal Networks of Face-to-Face Human Interactions”. In: *Temporal Netw.* Ed. by Petter Holme and Jari Saramäki. Underst. Complex Syst. Springer, 2013. DOI: `10.1007/978-3-642-36461-7\_10`.

[5] Jon Louis Bentley. *A Survey of Techniques for Fixed Radius Near Neighbor Searching.* Tech. rep. Stanford University, 1975.

[6] Christopher M. Bishop. “Pattern Recognition and Machine Learning”. In: *Inf. Sci. Stat.* Ed. by M. I. Jordan, Robert Nowak, and Bernhard Schoelkopf. Springer, 2006.

[7] Sergey Brin. “Near Neighbor Search in Large Metric Spaces”. In: *Proceedings of the 21th International Conference on Very Large Data Bases.* VLDB ’95. 1995, pp. 574–584.

[8] Glen Van Brummelen. *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry.* Princeton University Press, 2013.

[9] Aydın Buluç et al. “Recent Advances in Graph Partitioning”. In: *Algorithm Eng.: Sel. Results Surveys.* Ed. by Lasse Kliemann and Peter Sanders. Springer, 2016. DOI: 10.1007/978-3-319-49487-6_4.

[10] Andrea Capponi et al. “A Survey on Mobile Crowdsensing Systems: Challenges, Solutions, and Opportunities”. In: *IEEE Commun. Surv. Tut.* 21.3 (2019), pp. 2419–2465. DOI: 10.1109/COMST.2019.2914030.

[11] William Clinger. “Foundations of actor semantics”. PhD thesis. MIT, 1981. URL: http://hdl.handle.net/1721.1/6935.

[12] Thomas H. Cormen et al. *Introduction to Algorithms.* Fourth. The MIT Press, 2022.

[13] Meggan E. Craft. “Infectious disease transmission and contact networks in wildlife and livestock”. In: *Phil. Trans. R. Soc. B* 370 (1669 2015). DOI: 10.1098/rstb.2014.0107.

[14] Jesper Dall and Michael Christensen. "Random geometric graphs". In: *Phys. Rev. E* 66 (1 2002). DOI: `10.1103/PhysRevE.66.016121`.

[15] Leon Danon et al. "Networks and the Epidemiology of Infectious Disease". In: *Interdiscip. Perspect. Infect. Dis.* 2011 (2011). DOI: `10.1155/2011/284909`.

[16] Centers for Disease Control and Prevention. *Quarantine and Isolation.* `https://www.cdc.gov/coronavirus/2019-ncov/your-health/quarantine-isolation.html`. 2021.

[17] Santo Fortunato. "Community detection in graphs". In: *Phys. Rep.* 486 (3–5 2010). DOI: `10.1016/j.physrep.2009.11.002`.

[18] Julie Fournet and Alain Barrat. "Contact Patterns among High School Students". In: *PLoS ONE* 9 (9 2014). DOI: `10.1371/journal.pone.0107878`.

[19] Susan Fowler and Victor Stanwick. *Web Application Design Handbook: Best Practices for Web-Based Software.* Morgan Kaufmann Publishers, 2004.

[20] Erich Gamma et al. *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley, 1995.

[21] Raghu K. Ganti, Fan Ye, and Hui Lei. "Mobile crowdsensing: current state and future challenges". In: *IEEE Commun. Mag.* 49.11 (2011), pp. 32–39. DOI: `10.1109/MCOM.2011.6069707`.

[22] Bin Guo et al. "Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm". In: *ACM Comput. Surv.* 48.1 (2015), pp. 1–31. DOI: `10.1145/2794400`.

[23] Lea Hamner et al. "High SARS-CoV-2 Attack Rate Following Exposure at a Choir Practice – Skagit County, Washington, March 2020". In: *MMWR Surveill. Summ.* 69 (19 2020). DOI: `10.15585/mmwr.mm6919e6`.

[24] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming.* Morgan Kaufmann, 2012.

[25] Carl Hewitt. "Viewing control structures as patterns of passing messages". In: *Artificial Intelligence* 8.3 (1977), pp. 323–364. DOI: `10.1016/0004-3702(77)90033-9`.

[26] Carl Hewitt and Henry Baker. *Laws for communicating parallel processes.* Working paper. MIT Artificial Intelligence Laboratory, 1977. URL: `http://hdl.handle.net/1721.1/41962`.

[27] Carl Hewitt, Peter Bishop, and Richard Steiger. "A universal modular ACTOR formalism for artificial intelligence". In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence.* IJCAI'73. 1973, pp. 235–245. URL: `https://dl.acm.org/doi/10.5555/1624775.1624804`.

[28] Petter Holme and Beom Jun Kim. "Growing scale-free networks with tunable clustering". In: *Phys. Rev. E* 65 (2 2002). DOI: `10.1103/PhysRevE.65.026107`.

[29] Petter Holme and Jari Saramäki. "Temporal networks". In: *Phys. Rep.* 519 (3 2012). DOI: 10.1016/j.physrep.2012.03.001.

[30] George Karypis and Vipin Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". In: *SIAM J. Sci. Comput.* 20 (1 1998). DOI: 10.1137/S1064827595287997.

[31] Ashraf M. Kibriya and Eibe Frank. "An Empirical Comparison of Exact Nearest Neighbour Algorithms". In: *Knowledge Discovery in Databases: PKDD 2007*. 2007, pp. 140–151.

[32] Andreas Koher et al. "Contact-based model for epidemic spreading on temporal networks". In: *Phys. Rev. X* 9 (3 2019). DOI: 10.1103/PhysRevX.9.031017.

[33] Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. "Factor graphs and the sum-product algorithm". In: *IEEE Trans. Inf. Theory* 47 (2 2001). DOI: 10.1109/18.910572.

[34] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. "Benchmark graphs for testing community detection algorithms". In: *Phys. Rev. E* 78 (4 2008). DOI: 10.1103/PhysRevE.78.046110.

[35] Nicholas D. Lane et al. "A survey of mobile phone sensing". In: *IEEE Commun. Mag.* 48.9 (2010), pp. 140–150. DOI: 10.1109/MCOM.2010.5560598.

[36] Andrey Y. Lokhov et al. "Inferring the origin of an epidemic with a dynamic message-passing algorithm". In: *Phys. Rev. E* 90 (1 2014). DOI: 10.1103/PhysRevE.90.012801.

[37] Zhiping Lu, Yunying Qu, and Shubo Qiao. *Geodesy: Introduction to Geodetic Datum and Geodetic Systems*. Springer, 2014.

[38] Huadong Ma, Dong Zhao, and Peiyan Yuan. "Opportunities in mobile crowd sensing". In: *IEEE Commun. Mag.* 52.8 (2014), pp. 29–35. DOI: 10.1109/MCOM.2014.6871666.

[39] Ahmed R. Mahmood, Sri Punni, and Walid G. Aref. "Spatio-temporal access methods: a survey (2010 - 2017)". In: *Geoinformatica* 23 (1 2019), pp. 1–36.

[40] Grzegorz Malewicz et al. "Pregel: A System for Large-Scale Graph Processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[41] Robert McCune, Tim Weninger, and Greg Madey. "Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing". In: *ACM Comput. Surveys* 48 (2 2015). DOI: 10.1145/2818185.

[42] Cristina Menni et al. "Real-time tracking of self-reported symptoms to predict potential COVID-19". In: *Nat. Med.* 26 (7 2020). DOI: 10.1038/s41591-020-0916-2.

[43] Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. "Spatio-temporal Access Methods". In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 26 (2003), pp. 1–7.

[44] James Moody. "The Importance of Relationship Timing for Diffusion". In: *Soc. Forces.* 81 (1 2002).

[45] G.M. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. Tech. rep. International Business Machines, 1966.

[46] Stephen M. Omohundro. *Five Balltree Construction Algorithms*. Tech. rep. International Computer Science Institute, 1989.

[47] Romualdo Pastor-Satorras et al. "Epidemic processes in complex networks". In: *Rev. of Mod. Phys.* 87 (3 2015). DOI: 10.1103/RevModPhys.87.925.

[48] Alex Preukschat and Drummond Reed. *Self-Sovereign Identity: Decentralized digital identity and verifiable credentials*. Manning, 2021.

[49] Christopher S. Riolo, James S. Koopman, and Stephen E. Chick. "Methods and measures for the description of epidemiologic contact networks". In: *J. Urban Health* 78 (3 2001). DOI: 10.1093/jurban/78.3.446.

[50] Jan Van Sickle. *Basic GIS coordinates*. CRC Press, 2004.

[51] Ryan Tatton et al. *ShareTrace: Contact Tracing with Asynchronous, Parallel Message Passing on a Temporal Graph*. 2022. arXiv: 2203.12445 [cs.DC].

[52] Ryan Tatton et al. "ShareTrace: Contact Tracing with the Actor Model". In: *2022 IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. ©2022 IEEE. 2022, pp. 13–18. DOI: 10.1109/HealthCom54947.2022.9982762.

[53] Leslie G. Valiant. "A Bridging Model for Parallel Computation". In: *Communications of the ACM* 33.8 (1990). DOI: 10.1145/79173.79181.

[54] Tao Zhou et al. “Bipartite network projection and personal recommendation”. In: *Phys. Rev. E* 76 (4 2007).

[55] Lorenzo Zino and Ming Cao. “Analysis, Prediction, and Control of Epidemics: A Survey from Scalar to Dynamic Network Models”. In: *IEEE Circuits Syst. Mag.* 21 (4 2021). DOI: 10.1109/mcas.2021.3118100.