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 is a timestamped infection probability where $s \in [0, 1]$ and $t \in \mathbb{N}$ 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 [22]; and *exposure scores*, or posterior infection probabilities, which incorporate the risk of direct and indirect contact (i.e., sustained, proximal interaction) with others.

Given their recent symptom 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 nodes (i.e., contacts) in the graph.

Formally, let $G = (V, F, E)$ be a *factor graph* where V is the set of variable nodes, F is the set of factor nodes, and E is the set of edges incident between them [18]. A *variable node*

$$v : \Omega \to \{0, 1\}$$

is a random variable that represents the infection status of an individual, where the sample space is $\Omega = \{\textit{healthy}, \textit{infected}\}$ and

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

Thus, $p_t(v_i) = s_t$ is a risk score of the i-th individual. A *factor node*

$$f : V \times V \to [0, 1]$$

defines the transmission of infection risk between two contacts. Specifically, contact between the i-th and j-th individual is represented by the factor node $f(v_i, v_j) = f_{ij}$, which is adjacent to the variable nodes v_i, v_j. This work and previous work [4] assume risk transmission is a symmetric function, $f_{ij} = f_{ji}$. However, it may be extended to account for an individual's susceptibility and transmissibility such that $f_{ij} \neq f_{ji}$. Figure 1.1 depicts a factor graph that reflects the domain constraints.

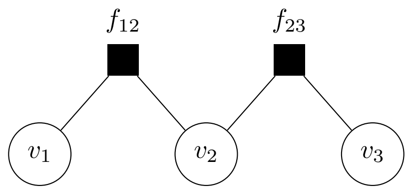

Figure 1.1: A factor graph of 3 variable nodes and 2 factor nodes.

1.1 Synchronous Risk Propagation

Risk propagation was first proposed as a synchronous, iterative message-passing algorithm that uses the factor graph to compute exposure scores [4]. The inputs to Synchronous-Risk-Propagation are the family of risk score sets S, where

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

is the set of symptom scores and exposure scores of the i-th individual; and the contact set

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

where (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 $t + \delta$. To remain consistent with prior work [4], the *minimum contact duration* $\delta \in \mathbb{N}$ is assumed to be $\delta = 15$ minutes[1].

[1]While this and prior work require contact to occur over a contiguous period of δ time, the Centers for Disease Control and Prevention only requires that the duration of contact is at least δ *over a 24-hour period* [7].

Risk scores and contacts are assumed to have finite relevance. As such, (1.1) and (1.2) are constrained by the *risk score time-to-live* $T_s \in \mathbb{N}$ and the *contact time-to-live* $T_c \in \mathbb{N}$, respectively, where $\tau \in \mathbb{N}$ is the time at which SYNCHRONOUS-RISK-PROPAGATION is invoked.

1.1.1 Initialization (lines 1–4)

The factor graph G is created from the contact set C. The data structure R is used to record the risk scores associated with each variable node. Specifically, $R_i^{(n)}$ is the set of risk scores associated with the variable node v_i in the n-th iteration. For the i-th individual, $R_i^{(1)}$ is assigned the top K risk scores in S_i.

1.1.2 Computing variable messages (lines 6–7)

The current exposure score of the i-th individual is defined as $\max S_i$, which assumes that interacting with others has a nondecreasing effect on their infection risk. Consequently, a *variable message* $m_{v_i \to f_{ij}}^{(n)}$ from the variable node v_i to the factor node 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].

1.1.3 Computing factor messages (lines 8–9)

A *factor message* $m^{(n)}_{f_{ij}\to v_j}$ from the factor node f_{ij} to the variable node v_j during the n-th iteration describes the infection risk of interacting with those at most n degrees separated from the j-th individual through the i-th individual. This population is defined by the subgraph induced in the factor graph G by

$$\{\, v \in V \cap F \setminus \{v_j, f_{ij}\} \mid d(v_i, v) \leq 2n \,\},$$

where $d(u,v)$ is the distance between the nodes u, v such that $d(u,v) = \infty$ if u, v are disconnected.

The computation of the factor message $m^{(n)}_{f_{ij}\to v_j}$ comprises of filtering and transforming the risk scores contained in the variable message $m^{(n)}_{v_i\to f_{ij}}$.

1. The i-th individual is defined by their highest recent risk score, $\max S_i$.
2. Risk transmission decays exponentially at a rate of $\log \alpha$, where $\alpha \in (0,1)$ is the *transmission rate*. A value of $\alpha = 0.8$ is assumed, unless specified otherwise [13].
3. A risk score is *relevant* to the contact (i,j,t) if it is computed at most β time after the time of contact t.

Unlike previous work [4], the contact set C only contains *valid* contacts.

A factor message $m^{(n)}_{f_{ij}\to v_j}$ is the maximum *relevant* risk score contained in the variable message $m^{(n)}_{v_i\to f_{ij}}$ The set of risk scores in the variable message $m^{(n)}_{v_i\to f_{ij}}$ is used to derive the factor message $m^{(n)}_{f_{ij}\to v_j}$.

is the maximum risk score derived by variable node v_i, computed at most β time after individuals i, j interacted, and scaled by the transmission rate α. If the variable message $m^{(n)}_{v_i \to f_{ij}}$ is empty or none of the risk scores contained therein satisfy the temporal constraint, then a risk score of 0 is assigned.

1.1.4 Detecting termination (line 5)

Assigned to $R_i^{(n)}$ are the top K risk scores from the first n iterations and derived from the neighboring factor nodes N_i of the variable node v_i.

Synchronous-Risk-Propagation terminates either after N iterations or when the normed difference between individuals' maximum exposure score of the previous and current iteration is less than the tolerance δ. Note that R_{max} is the vector of maximum risk scores such that $(R_{max})_i = \max R_i$. The ℓ_1-norm and the ℓ_∞-norm are sensible choices for detecting convergence. The ℓ_1-norm was assumed in previous work [4], which ensures

Synchronous-Risk-Propagation(S, C)

$(V, F, E) \leftarrow$ Factor-Graph(C)

$n \leftarrow 1$

for each $v_i \in V$

  $R_i^{(n-1)} \leftarrow$ top K of S_i

while $n \leq N$ **and** $\|R_{max}^{(n)} - R_{max}^{(n-1)}\| > \delta$

  for each $\{v_i, f_{ij}\} \in E$

    $m_{v_i \to f_{ij}}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\, m_{f_{ij} \to v_i}^{(k)} \mid k \in [1\,..\,n-1] \,\}$

  for each $\{v_i, f_{ij}\} \in E$

    $m_{f_{ij} \to v_j}^{(n)} \leftarrow \max(\{0\} \cup \{\, \alpha \cdot s_t \mid s_t \in m_{v_i \to f_{ij}}^{(n)}, t \leq t_{ij} + \beta \,\})$

  for each $v_i \in V$

    $R_i^{(n)} \leftarrow$ top K of $\{\, m_{f_{ij} \to v_i}^{(n)} \mid f_{ij} \in N_i \,\}$

  $n \leftarrow n + 1$

return $R_{max}^{(n)}$

1.2 Asynchronous Risk Propagation

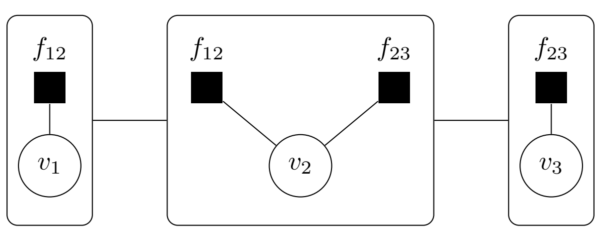

Figure 1.2: One-mode projection onto the variable nodes in Figure 1.1.

The only purpose of a factor vertex is to compute and relay messages between variable vertices. Thus, one-mode projection onto the variable vertices can be applied such that variable vertices $v_i, v_j \in V$ are adjacent if the factor vertex $f_{ij} \in F$ exists [26]. Figure 1.2 shows the modified topology.

To send a message to variable vertex v_i, variable vertex v_j applies the computation associated with the factor vertex f_{ij}. This modification differs from the distributed extension of risk propagation [4] in that we do not create duplicate factor vertices and messages in each user's PDS. By storing the contact time between users on the edge incident to their variable vertices, this modified topology is identical to the *contact-sequence* representation of a *contact network*, a kind of *time-varying* or *temporal network* in which a vertex represents a person and an edge indicates that two persons came in contact:

$$\{ (v_i, v_j, t) \mid v_i, v_j \in V; v_i \neq v_j; t \in \mathbb{N} \}, \tag{1.3}$$

where a triple (v_i, v_j, t) is called a *contact* [15]. Specific to risk propagation, t is the time at which users u and v *most recently* came in contact (see Section 1.2).

The usage of a temporal network in this work differs from its typical usage in epidemiology which focuses on modeling and analyzing the spreading dynamics of disease [9, 10, 17, 20, 24, 25, 27]. In contrast, this work uses a temporal network to infer a user's MPPI. As a result, Section 1.3 extends temporal reachability to account for both the message-passing semantics and temporal dynamics of the network. As noted by [15], the transmission graph provided by [25] "cannot handle edges where one vertex manages to not catch the disease." Notably, the usage of a temporal network in this work allows for such cases by modeling the possibility of infection as a continuous outcome. Factor graphs are useful for decomposing complex probability distributions and allowing for efficient inference algorithms.

However, as with risk propagation, and generally any application of a factor graph in which the variable vertices represent entities of interest (i.e., of which the marginal probability of a variable is desired), applying one-mode projection is a .

1.2.1 ShareTrace Actor System

As a distributed algorithm, risk propagation is specified from the perspective of an *actor*, which is equivalent to the paradiagm of *think- like-a-vertex* graph-processing algorithms [21]. Some variation exists on exactly how actor behavior is defined [1, 3, 11]. Perhaps the simplest definition is that the *behavior of an actor* is both its *interface* (i.e., the types of messages it can receive) and *state* (i.e., the internal data it uses to process messages) [11]. An *actor system*[2] is defined as the set of actors it contains and the set of unprocessed messages[3] in the actor mailboxes. An expanded definition of an actor system also includes a *local states function* that maps mail addresses to behaviors, the set of *receptionist actors* that can receive communication that is external to the actor system, and the set of *external actors* that exist outside of the actor system [1, 3]. Practically, a local states function is unnecessary to specify, so the narrower definition of an actor system is used. The remainder of this section describes the components of the ShareTrace actor system.

[2]This is technically referred to as an *actor system configuration*.

[3]Formally, a *message* is called a *task* and is defined by a *tag*, a unique identifier; a *target*, the mail address to which the message is delivered; and a *communication*, the message content [1].

1.2.1.1 User Actor Behavior

Each user corresponds to an actor that participates in the message-passing protocol of risk propagation. Herein, the user of an actor will only be referred to as an *actor*. The following variant of the concurrent, object-oriented actor model is assumed to define actor behavior [2, 3].

- An actor follows the *active object pattern* [11, 19] and the *Isolated Turn Principle* [11]. Specifically, the state change of an actor is carried out by instance- variable assignment, instead of the canonical BECOME primitive that provides a functional construct for pipelining actor behavior replacement [1–3]. The interface of a user actor is fixed in risk propagation, so the more general semantics of BECOME is unnecessary.

- The term "name" [1, 14] is preferred over "mail address" [1–3] to refer to the sender of a message. Generally, the mail address that is included in a message need not correspond to the actor that sent it. Risk propagation, however, requires this actor is identified in a risk-score message. Therefore, to emphasize this requirement, "name" is used to refer to both the identity of an actor and its mail address.

- An actor is allowed to include a loop with finite iteration in its behavior definition; this is traditionally prohibited in the actor model [1, 2].

- Because the following pseudocode to describe actor behavior mostly follows the conventions of [8] (see Appendix B), the behavior definition is implied by all procedures that take as input an actor.

The Create-Actor operation initializes an actor, which is equivalent to the *new expression* [1] or Create primitive [2, 3] with the exception that it only specifies the attributes (i.e., state) of an actor. As mentioned earlier, the behavior description of an actor is implied by the procedures that require an actor as input. Every actor A has the following attributes.

- $A.name$: an identifier that enables actors to communicate with it [1, 14].
- $A.contacts$: a dictionary (see Appendix B) that maps an actor name to a timestamp. That is, if user i of actor A_i comes in contact with user j of actor A_j, then the *contacts* of A_i contains the name of A_j along with the most recent contact time for users i and j. The converse holds for user j. This is an extension of the *acquaintance vector* [14] or *acquaintance list* [1, 3].
- $A.score$: the user's current exposure score. This attribute is either a default risk score (see Default-Risk-Score), a risk score sent by an actor of some contacted user[4], or a symptom score of the user.
- $A.cache$: a dictionary that maps a time interval to a risk score; used to tolerate synchronization delays between a user's device and actor (see Section 1.2.1.2).

Risk scores and contacts have finite relevance, which is parametrized by a *liveness* or *relevance duration* $T_s, T_c > 0$, respectively. The relevance of risk scores and contacts is important, because it influences how actors pass

[4] As discussed later (see Handle-Contact-Message in Section 1.2.1.1), it possible for an actor i to send a message to an actor j but not be a contact of actor j.

Create-Actor

$A.name \leftarrow$ Create-Name

$A.contacts \leftarrow \emptyset$

$A.score \leftarrow$ Default-Risk-Score(A)

$A.cache \leftarrow \emptyset$

return A

Default-Risk-Score(A)

$s.value \leftarrow 0$

$s.t \leftarrow 0$

$s.sender \leftarrow A.name$

return s

messages. For example, actors do not send irrelevant risk scores or relevant risk scores to irrelevant contacts. The *time-to-live* (TTL) of a risk score or contact is the remaining duration of its relevance. In the following operations, $s.t$ denotes the time at which the risk score was originally computed, $c.t$ is the contact time, and Get-Time returns the current time. The interface of a

Score-Ttl(s)

return $\max\{0, T_s - (\tau - s.t)\}$

Contact-Ttl(c)

return $\max\{0, T_c - (\tau - c.t)\}$

user actor is defined by two types of messages: contact messages and risk-score messages. A *contact message* c contains the name $c.name$ of the actor whose user was contacted and the contact time $c.t$. A *risk-score message* s is simply a risk score along with the actor's name $s.sender$ that sent it. A risk score previously defined as the ordered pair (r, t) (see Section 1.2) is represented as the attributes r and $s.t$. The following sections discuss how a contact message and risk-message are processed by an actor.

1.2.1.1.1 Handling Contact Messages There are two ways in which a user actor can receive a contact message. The first, technically correct approach is for a receptionist actor to mediate the communication between the user actor and the PDS so that the user actor can retrieve its user's contacts. The second approach is to relax this formality and allow the user actor to communicate with the PDS directly[5].

The HANDLE-CONTACT-MESSAGEoperation defines how a user actor processes a contact message. A contact is assumed to have finite relevance which is parametrized by the *contact time-to-live* $T_c > 0$. A contact whose contact time occurred no longer than T_c time ago is said to be *alive*. Thus, a user actor only adds a contact if it is alive. Regardless of whether the contact

HANDLE-CONTACT-MESSAGE(A, c)

```
1: if CONTACT-TTL(c) > 0
2:     c.key ← c.name
3:     c.threshold ← 0
4:     INSERT(A.contacts, c)
5:     START-CONTACTS-REFRESH-TIMER(A)
6:     SEND-CURRENT-OR-CACHED(A, c)
```

[5]If the PDS itself is an actor, then a push-oriented dataflow could be implemented, where the user actor receives contact messages (and symptom-score messages). This would be more efficient and timely than a pull-oriented dataflow in which the PDS is a passive data store that requires the user actor or receptionist to poll it for new data.

```
START-CONTACTS-REFRESH-TIMER(A)
  oldest ← MINIMUM(A.contacts)
  if oldest ≠ NIL
      x.key ← "contacts"
      START-TIMER(tr, CONTACT-TTL(oldest))
```

```
HANDLE-CONTACTS-REFRESH-TIMER(tr)
  for each c ∈ A.contacts
      if CONTACT-TTL(c) ≤ 0
          DELETE(A.contacts, c)
  START-CONTACTS-REFRESH-TIMER(A)
```

is alive, the user actor attempts to send a risk-score message that is derived from its current exposure score or a cached risk score (see Section 1.2.1.2):

Like contacts, each risk score is assumed to have finite relevance

```
SEND-CURRENT-OR-CACHED(A, c)
  if SHOULD-RECEIVE(c, A.score)
      SEND(A, c, A.score)
  else
      s ← CACHE-MAX(A.cache, c.t + β)
      if s ≠ NIL and SCORE-TTL(s) > 0
          SEND(A, c, s)
```

that is parametrized by the *score time-to-live* $T_s > 0$ and evaluated by the operation IS-SCORE-ALIVE. To send an actor's current exposure score, the contact must be sufficiently recent. It is assumed that risk scores computed after a contact occurred have no effect on the user's exposure score.

SEND(A, c, s)

$s'.value \leftarrow \alpha \cdot s.value$

$s'.t \leftarrow s.t$

$s'.sender \leftarrow A.name$

SEND(c, s')

UPDATE-THRESHOLD(c, s')

SHOULD-RECEIVE(c, s)

return $c.threshold < s.value$ **and** $c.t + \beta \geq s.t$ **and** SCORE-TTL$(s) > 0$ **and** $c.name \neq s.sender$

UPDATE-THRESHOLD(c, s)

$threshold \leftarrow \gamma \cdot s.value$

if $threshold > c.threshold$

    $c.threshold \leftarrow threshold$

    START-THRESHOLD-TIMER(c, s)

START-THRESHOLD-TIMER(c, s)

$x.key \leftarrow c.name +$ " threshold"

$x.name \leftarrow c.name$

START-TIMER$(tr,$ SCORE-TTL$(s))$

To account for the disease incubation period, a delay in reporting symptoms, or a delay in establishing actor communication, a time buffer $\beta \geq 0$ is considered. That is, a risk score is not sent to a contact if IS-CONTACT-RECENT returns FALSE.

The TRANSMITTED operation is used to generate risk scores that are sent to other actors. It is assumed that contact only implies an incomplete transmission of risk between users. Thus, when sending a risk score to another

```
HANDLE-THRESHOLD-TIMER(A, tr)
c ← SEARCH(A.contacts, x.name)
if c ≠ NIL
    s ← CACHE-MAX(A.cache, c.t + β)
    if s ≠ NIL and SCORE-TTL(s) > 0
        c.threshold ← γ · s.value
        START-THRESHOLD-TIMER(c, s)
    else
        c.threshold ← 0
```

actor, the value of the risk score is scaled by the *transmission rate* $\alpha \in (0, 1)$. Notice that the time of the risk score is left unchanged; the act of sending a risk-score message is independent of when the risk score was first computed.

If line 1 of SEND-CURRENT-OR-CACHED evaluates to FALSE, then the actor attempts to retrieve the maximum cached risk-score message based on the buffered contact time. If such a message exists and is alive, a risk-score message is derived and sent to the contact. The SEND operation follows the semantics of the SEND-TO primitive [1, 2] or the *send command* [3].

1.2.1.1.2 Handling Risk-Score Messages Upon receiving a risk-score message, an actor executes the following operation. The UPDATE-ACTOR operation is responsible for updating an actor's state, based on a received risk-score message. Specifically, it stores the message inside the actor's interval cache *A.cache*, assigns the actor a new exposure score and send coefficient (discussed below) if the received risk-score value exceeds that of the current exposure score, and removes expired contacts. In previous work

```
Handle-Risk-Score-Message(A, s)
Cache-Insert(A.cache, s)
if s.value > A.score.value
    previous ← A.score
    A.score ← s
    if previous ≠ Default-Risk-Score(A)
        Start-Score-Refresh-Timer(A)
Propagate(A, s)

Start-Score-Refresh-Timer(A)
x.key ← "score"
Start-Timer(tr, Score-Ttl(A.score))

Handle-Score-Refresh-Timer(A, tr)
s ← Cache-Max(Get-Time)
if s = nil
    s ← Default-Risk-Score(A)
A.score ← s
if s ≠ Default-Risk-Score(A)
    Start-Score-Refresh-Timer(A)
```

[4], risk propagation assumes synchronous message passing, so the notion of an iteration or inter-iteration difference threshold can be used as stopping conditions. However, as a streaming algorithm that relies on asynchronous message passing, such stopping criteria are unnatural. Instead, the following heuristic is applied and empirically optimized to minimize accuracy loss and maximize efficiency. Let $\gamma > 0$ be the *send coefficient* such that an actor only sends a risk-score message if its value exceeds the actor's *send threshold*

A.threshold (see line **??** in Propagate).

Assuming a finite number of actors, any positive send coefficient γ guarantees that a risk-score message will be propagated a finite number of times. Because the value of a risk score that is sent to another actor is scaled by the transmission rate α, its value exponentially decreases as it propagates away from the source actor with a rate constant $\log \alpha$.

As with the Send-Current-Or-Cached operation, a risk-score message must be alive and relatively recent for it to be propagated. As in previous work [4], factor marginalization is achieved by not propagating the received message to the actor who sent it. The logic of Propagate differs, however, in two ways. First, it is possible that no message is not propagated to a contact. The intent of sending a risk-score message is to update the exposure score of other actors. However, previous work [4] required that a "null" message with a risk-score value of 0 is sent. Sending such ineffective messages incurs additional communication overhead. The second difference is that only the *most recent* contact time is used to determine if a message should be propagated to a contact. Contact times determine what messages are relevant. Given two contact times t_1, t_2 such that $t_1 \leq t_2$, then any risk score with time $t \leq t_1 + \beta$ also satisfies $t \leq t_2 + \beta$. Thus, storing multiple contact times is unnecessary.

Propagate(A, s)

1: **for each** $c \in A.contacts$
2:     **if** Should-Receive(c, s)
3:         Send(A, c, s)

1.2.1.2 Risk Score Caching

For two actors to communicate, each must have the other actor in their contacts (see Section 1.2.1.1). Recall that an actor must retrieve these contacts from the user's PDS, which subsequently requires synchronization with the user's mobile device (see **??**). While the user's device can locally store contacts from proximal devices and symptoms of the user, an internet connection is needed to synchronize with the PDS and thus the user's actor. Therefore, it is not only possible but a reality that the user's mobile device and actor will not always be synchronized.

In the best case, this "lag" may only be a few seconds; in the worst case, the user's device is offline for several days. If δ_i (δ_j) is the delay between when the device of user i (ref. j) records a contact with user j (ref. i) and when its actor receives the corresponding contact message, then the delay between when actors A_i and A_j can communicate bidirectionally and when the contact actually occurred is $\delta_{ij} = \max(\delta_i, \delta_j)$. Such dissonance between the "true" state of the world (i.e., when users actually came in contact) and that known to the network of actors could impact the accuracy of risk propagation, which assumes such delays are nonexistent. To address this issue, each actor maintains a cache of received risk-score messages such that it can still send a message that reflects its previous state to a contact that was significantly delayed.

An *interval cache* C is a dictionary that maps a finite time interval (key) to a data element (value). In a typical cache, the *time-to-live* (TTL) of an element is a fixed duration after which the element is removed or *evicted*. In an interval cache, however, the TTL of an element is determined by its associated interval and the current time. That is, an interval cache is like a series of sliding windows, where each window corresponds to an interval that can hold a single element. Thus, the TTL of an element is the duration between the start of its interval and the start of the earliest interval in the cache.

An interval cache maintains N contiguous, half-closed (start-inclusive) intervals, each of duration T. An interval cache contains N_a *look-ahead intervals* and N_b *look-back intervals* such that $N = N_b + N_a$. Look- ahead (resp. look-back) intervals allow elements to be associated with intervals whose start times are later (resp. earlier) than the current time t. The *look-back duration* T_b and *look-ahead duration* T_a are defined as

$$T_b = N_b \cdot T$$
$$T_a = N_a \cdot T.$$

The *range* of the interval cache is $[C.start, C.end)$, where

$$C.start = C.\mathit{refresh} - T_b$$
$$C.end = C.\mathit{refresh} + T_a,$$

and $C.\mathit{refresh}$ is the time at which the cache was last refreshed.

An interval cache is a “live” data structure, so the range must be updated periodically to reflect the advancement of time. Furthermore, intervals and their associated elements that are no longer contained in the range must be evicted. This process of updating the range and evicting cached elements is called *refreshing the cache*. The *refresh period* $T > 0$ of an interval cache is the duration until the range is updated, based on the current time t. Depending on the interval duration T and the nature of the data that is being cached, the refresh period may be on the order of seconds or days. To recognize when a refresh is necessary, the cache maintains the attribute $C.refresh$, which is the time of the previous refresh. The operation CACHE-REFRESHupdates the range if at least T time has elapsed since the previous refresh and then removes all expired elements.

CACHE-REFRESH(C)

$t \leftarrow$ GET-TIME
if $t - C.refresh > T$
    $C.start \leftarrow t - T_b$
    $C.end \leftarrow t + T_a$
    $C.refresh \leftarrow t$
    for each $x \in C$
        if $x.key < C.start$
            DELETE(C, x)

The operation CACHE-INSERTrefreshes the cache, if necessary, and merges into the cache the element pointed to by x if its timestamp $x.t$ is in the range. Keys in the interval cache are interval start times and are lazily computed (line 2) to avoid storing intervals with no associated element. By not storing all intervals explicitly, the interval cache only achieves $O(N)$ space complexity when each interval has an associated element. The MERGEoperation (line 7) can be as trivial as replacing the existing value. For risk propagation, the interval cache associates with each interval the newest risk score of maximum value.

CACHE-KEY(C, x)

return $C.start + \left\lfloor \frac{x.t - C.start}{T} \right\rfloor \cdot T$

CACHE-INSERT(C, x)

if $x.t \in [C.start, C.end)$
    $x.key \leftarrow$ CACHE-KEY(C, x)
    $x_{old} \leftarrow$ SEARCH(C, x)
    if $x_{old} =$ NIL
        $x_{new} \leftarrow x$
    else
        $x_{new} \leftarrow$ MERGE(x_{old}, x)
    INSERT(C, x_{new})

The intention of sending a cached risk score to a contacted user actor is to account for the delay between when the contact occurred and when the actors establish communication. Therefore, the cached risk score that should be sent is that which would have been the current exposure score at the time the users came into contact. That is, each user actor should send the maximum risk score whose cache interval ends at or before the time of contact, accounting for the time buffer β (see line 4 of SEND-CURRENT-OR-CACHEDin Section 1.2.1.1). The operation CACHE-MAXis used to carry out this query.

CACHE-MAX(C, t)

return MAXIMUM$(\{x \in C \mid x.key <$ CACHE-KEY$(t)\})$

An interval cache is implemented by augmenting a hash table [**?** , pp. 253–285] with the aforementioned attributes and parameters. By using a hash table, the interval cache offers $\Theta(1)$ average-case insert, search, and delete operations. Reference [**?** , pp. 348–354] implements an *interval tree* by augmenting a red-black tree. However, insert, delete, and search operations on a red-black tree require $\Theta(\log N)$ in the average case.

1.3 Message Reachability

A fundamental concept of reachability in temporal networks 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, denoted $u \rightarrow v$ [23]. The following quantities are derived from the notion of a time-respecting path and help quantify reachability in a time-varying network [15].

- The *influence set* $I(v)$ *of vertex* v is the set of vertices that v can reach by a time-respecting path.
- The *source set* $S(v)$ *of vertex* v is the set of vertices that can reach v by a time-respecting path.
- The *reachability ratio* $f(G)$ *of a temporal network* G is the average influence-set cardinality of a vertex v.

Generally, a message-passing algorithm defines a set of constraints that determine when and what messages are sent from one vertex to another. 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 more general definition of reachability that can account for network topology *and* message-passing semantics [5].

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

$$m(u,v) = \sum_{(i,j)\in P} f(u,i,j,v),$$

where

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

Vertex v is *message reachable* from vertex u if there exists a shortest path such that $m(u,v) > 0$. The *message reachability of vertex u* is the maximum message reachability from vertex u:

$$m(u) = \max\{m(u,v) \mid v \in V\}. \tag{1.4}$$

The temporal reachability metrics previously defined can be extended to message reachability by only considering the message-reachable vertices:

$$I_m(u) = \{v \in V \mid m(u,v) > 0\}$$
$$S_m(v) = \{u \in V \mid m(u,v) > 0\}$$
$$f_m(G) = \sum_{v\in V} |I_m(v)| \cdot |V|^{-1}.$$

Let $\mathbf{M}$ be the $|V| \times |V|$ *message-reachability matrix* of the temporal network G such that vertices are enumerated $1, 2, \ldots, |V|$ and for each $m_{ij} \in \mathbf{M}$,

$$m_{ij} = \begin{cases} 1 & \text{if } m(i,j) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then the cardinality of the influence set for vertex i is the number of nonzero elements in the ith row of $\mathbf{M}$:

$$|I_m(i)| = \sum_{j=1}^{|V|} m_{ij}. \tag{1.5}$$

Similarly, the cardinality of the source set for vertex j is the number of nonzero elements in the jth column of $\mathbf{M}$:

$$|S_m(j)| = \sum_{i=1}^{|V|} m_{ij}. \tag{1.6}$$

For risk propagation, let $H(x)$ be the *Heaviside step function*,

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then message reachability is defined as

$$m(u,v) = \sum_{(i,j) \in P} f_r(u,i) \cdot f_c(u,i,j) \tag{1.7}$$

where P is the set of edges along the shortest path $u \to v$ such that the actors are enumerated $0, 1, \ldots, |P| - 1$ and

$$f_r(u, i) = H(\alpha^i \cdot r_u - \gamma \cdot r_i) \tag{1.8}$$

$$f_c(u, i, j) = H(t_{ij} - t_u + \beta) \tag{1.9}$$

are the value and contact-time constraints in the Should-Receive operation (see Section 1.2.1.1), where (r_i, t_i) is the current exposure score for actor i and t_{ij} is the most recent contact time between actors i and j.

The value of (1.7) can be found by associating with each symptom score a unique identifier. If each actor maintains a log of the risk scores it receives, then the set of actors that receive the symptom score or a propagated risk score thereof can be identified. This set of actors defines the induced subgraph on which to compute (1.7) using a shortest-path algorithm [16].

Regarding efficiency, (1.4) to (1.6) provide the means to quantify the communication overhead of a given message-passing algorithm on a temporal network. Moreover, because such metrics capture the temporality of message passing, they can better quantify complexity than traditional graph metrics.

By relaxing the constraint (1.9), it is possible to estimate (1.7) with (1.8). The *estimated message reachability of vertex u to vertex v*, denoted $\hat{m}(u, v)$, is defined as follows. Based on (1.8),

$$\alpha^{\hat{m}(u,v)} \cdot r_u \leq \gamma \cdot r_v,$$

where the left-hand side is the value of the propagated symptom score for actor u when $\hat{m}(u, v) = 1$, and the right-hand side is the value required by some message-reachable actor v to propagate the message received by actor u or some intermediate actor. Solving for $\hat{m}(u,v)$,

$$\hat{m}(u,v) \leq f(u,v), \tag{1.10}$$

where

$$f(u,v) = \begin{cases} 0 & \text{if } r_u = 0 \\ |P| & \text{if } r_v = 0 \\ \log_\alpha \gamma + \log_\alpha r_v - \log_\alpha r_u & \text{otherwise.} \end{cases}$$

Equation (1.10) indicates that a lower send coefficient γ will generally result in higher message reachability, at the cost of sending possibly ineffective messages (i.e., risk scores that do not update the exposure score of another actor). Equation (1.10) also quantifies the effect of the transmission rate α. Unlike the send coefficient, however, the transmission rate is intended to be derived from epidemiology to quantify disease infectivity and should not be optimized to improve performance.

Given the multivariate nature of message reachability, it is helpful to visualize how it with various combinations of parameter values. Figure 1.3 includes several line plots of estimated message reachability $\hat{m}(u,v)$ with respect to the initial risk score magnitude of actor u.

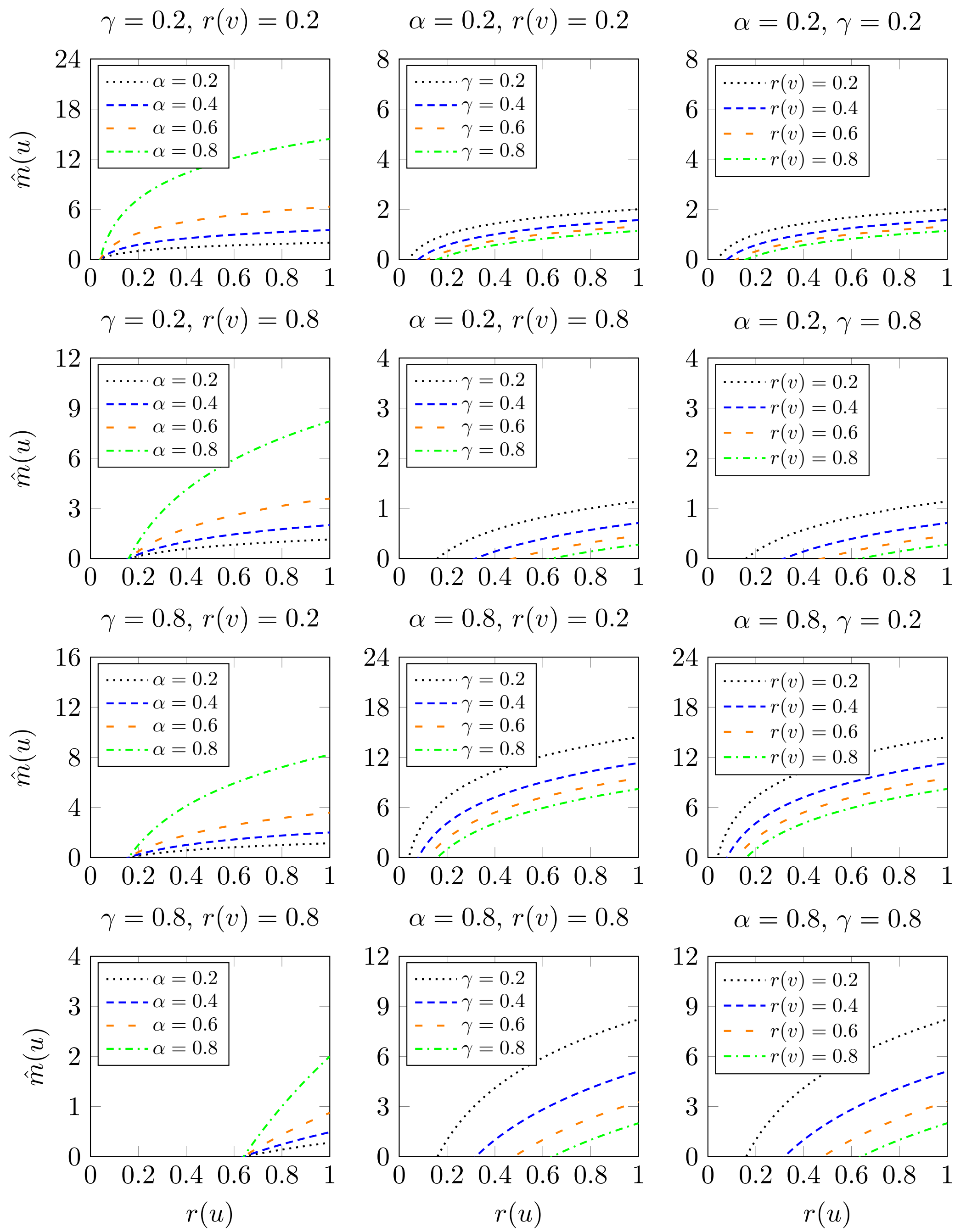

Figure 1.3: Estimated risk propagation message reachability $\hat{m}(u)$ for different values of the transmission rate α, the send tolerance γ, and the initial magnitude $r(v) = r_0(v)/\alpha$ of the destination user v with respect to the initial magnitude $r_u = r_0(u)/\alpha$ of the source user u.

Appendix A

Data Structures

Let a *dynamic set* S be a mutable collection of distinct elements, and a *dictionary* be a dynamic set that supports insertion, deletion, and membership querying. Each element of S is represented as an object x with attributes such that $x.key$ is a unique identifier for the object x [8, p. 249]. The operations SEARCH, INSERT, DELETE, MINIMUM, and MAXIMUM are mostly consistent with [8, p. 250].

- SEARCH(S, k) returns a pointer x to an element in the set S such that $x.key = k$, or NIL if no such element belongs to S.
- INSERT(S, x) adds the element pointed to by x to the set S.
- DELETE(S, x) removes the element pointed to by x from the set S.
- MINIMUM(S) and MAXIMUM(S) return a pointer x to the minimum and maximum element, respectively, of the totally ordered set S, or NIL if S is empty. Reference [8] only considers the *key* attribute of the

pointers when finding the minimum or maximum element of S. This work, however, will allow other attributes to be used.

Appendix B

Typographical Conventions

B.1 Mathematics

Mathematical typesetting follows the guidance of [12].

B.2 Pseudocode

The pseudocode conventions used in this work mostly follow [8, pp. 21–24].

- 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*. That is, 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 object's attributes are not" [8, p. 23]. Thus, object attribute assignment "is visible if the calling procedure has a pointer to the same object" [8, p. 24].

- A **return** statement "immediately transfers control back to the point of call in the calling procedure" [8, p. 24].

- 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.

Bibliography

[1] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. PhD thesis, MIT, Cambridge, MA, 1985.

[2] Gul Agha. Concurrent object-oriented programming. *Commun. ACM*, 33(9):125–141, 1990.

[3] Gul Agha and Carl Hewitt. Concurrent programming using actors: Exploiting large-scale parallelism. In S. N. Maheshwari, editor, *Foundations of Software Technology and Theoretical Computer Science*, pages 19–41, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[4] 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.

[5] Alain Barrat and Ciro Cattuto. Temporal networks of face-to-face human interactions. In Petter Holme and Jari Saramäki, editors, *Temporal Netw.*, Underst. Complex Syst. Springer, 2013.

[6] Christopher M. Bishop. Pattern recognition and machine learning. In M. I. Jordan, Robert Nowak, and Bernhard Schoelkopf, editors, *Inf. Sci. Stat.* Springer, 2006.

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

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, fourth edition, 2022.

[9] Meggan E. Craft. Infectious disease transmission and contact networks in wildlife and livestock. *Phil. Trans. R. Soc. B*, 370, 2015.

[10] Leon Danon, Ashley P. Ford, Thomas House, Chris P. Jewell, Gareth O. Roberts, Joshua V. Ross, and Matthew C. Vernon. Networks and the epidemiology of infectious disease. *Interdiscip. Perspect. Infect. Dis.*, 2011, 2011.

[11] Joeri De Koster, Tom Van Cutsem, and Wolfgang De Meuter. 43 years of actors: A taxonomy of actor models and their key properties. In *Proceedings of the 6th International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE 2016, pages 31–40, New York, NY, 2016. Association for Computing Machinery.

[12] Chris Dyer, Kevin Gimpel, and Noah A. Smith. A short guide to

typesetting math in NLP papers. http://demo.clab.cs.cmu.edu/cdyer/short-guide-typesetting.pdf.

[13] Lea Hamner, Polly Dubbel, Ian Capron, Andy Ross, Amber Jordan, Jaxon Lee, Joanne Lynn, Amelia Ball, Simranjit Narwal, Sam Russell, Dale Patrick, and Howard Leibrand. High SARS-CoV-2 attack rate following exposure at a choir practice – Skagit County, Washington, March 2020. *MMWR Surveill. Summ.*, 69, 2020.

[14] Carl Hewitt and Henry Baker. Laws for communicating parallel processes. Technical report, Massachusetts Institute of Technology, 1977. http://hdl.handle.net/1721.1/41962.

[15] Petter Holme and Jari Saramäki. Temporal networks. *Phys. Rep.*, 519, 2012.

[16] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24, 1977.

[17] Andreas Koher, Hartmut H. K. Lentz, James P. Gleeson, and Philipp Hövel. Contact-based model for epidemic spreading on temporal networks. *Phys. Rev. X*, 9, 2019.

[18] Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47, 2001.

[19] R. Greg Lavender and Douglas C. Schmidt. Active object – an object

behavioral pattern for concurrent programming, 1996. `https://csis.pace.edu/~marchese/CS865/Papers/Act-Obj.pdf`.

[20] Andrey Y. Lokhov, Marc Mézard, Hiroki Ohta, and Lenka Zdeborová. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Phys. Rev. E*, 90, 2014.

[21] Robert McCune, Tim Weninger, and Greg Madey. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Comput. Surveys*, 48, 2015.

[22] Cristina Menni, Ana M Valdes, Maxim B Freidin, Carole H Sudre, Long H Nguyen, David A Drew, Sajaysurya Ganesh, Thomas Varsavsky, M Jorge Cardoso, Julia S El-Sayed Moustafa, Alessia Visconti, Pirro Hysi, Ruth C E Bowyer, Massimo Mangino, Mario Falchi, Jonathan Wolf, Sebastien Ourselin, Andrew T Chan, Claire J Steves, and Tim D Spector. Real-time tracking of self-reported symptoms to predict potential COVID-19. *Nat. Med.*, 26, 2020.

[23] James Moody. The importance of relationship timing for diffusion. *Soc. Forces.*, 81, 2002.

[24] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Rev. of Mod. Phys.*, 87, 2015.

[25] Christopher S. Riolo, James S. Koopman, and Stephen E. Chick. Methods

and measures for the description of epidemiologic contact networks. *J. Urban Health*, 78, 2001.

[26] Tao Zhou, Jie Ren, Matú š Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 76, 2007.

[27] Lorenzo Zino and Ming Cao. Analysis, prediction, and control of epidemics: A survey from scalar to dynamic network models. *IEEE Circuits Syst. Mag.*, 21, 2021.