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 (Menni et al., 2020); 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 nodes (i.e., contacts).

Let $G = (X, F, E)$ be a *factor graph* where X is the set of variable nodes, F is the set of factor nodes, and E is the set of edges incident between them (Kschischang et al., 2001).

A *variable node* $x : \Omega \to \{0, 1\}$ 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 node* $f : X \times X \to [0, 1]$ defines the transmission of infection risk between two contacts. Specifically, contact between the i-th and j-th individual is represented by the factor node $f(x_i, x_j) = f_{ij}$, which is adjacent to the variable nodes x_i, x_j. This work and Ayday et al. (2021) 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.

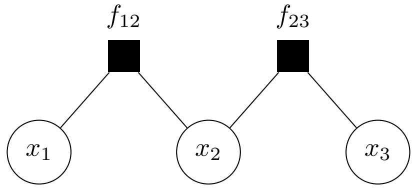

Figure 1.1: A factor graph of 3 variable nodes and 2 factor nodes.

1.1 Synchronous Risk Propagation

Ayday et al. (2021) 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 et al. (2021) 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 node x_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 (Bishop, 2006, pp. 411–415).

1.1.2 Factor Messages

A *factor message* $\lambda_{ij}^{(n)}$ from the factor node f_{ij} to the variable node 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 nodes 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. The expression

$t_{ij} + \beta$ is called the *buffered contact time*.

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.

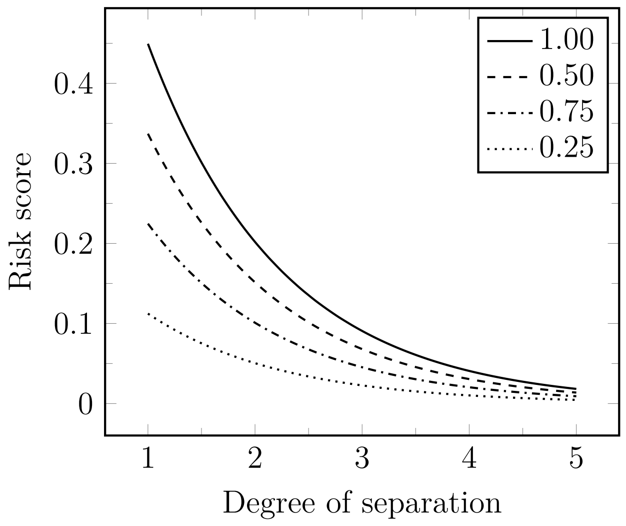

Figure 1.2: Exponential decay of risk scores.

To summarize, 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 et al. (2021) 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 node 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 nodes 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 et al. (2021) use the ℓ^1 norm, which ensures that an individual's exposure score changed by at most ϵ after the penultimate iteration.

```
Risk-Propagation$(S, C)$
 1: $(X, F, E) \leftarrow$ Create-Factor-Graph$(C)$
 2: $n \leftarrow 1$
 3: for each $x_i \in X$
 4:     $R_i^{(n-1)} \leftarrow$ top $K$ of $S_i$
 5:     $r_i^{(n-1)} \leftarrow \max R_i^{(n-1)}$
 6:     $r_i^{(n)} \leftarrow \infty$
 7: while $\|\mathbf{r}^{(n)} - \mathbf{r}^{(n-1)}\| > \epsilon$
 8:     for each $\{x_i, f_{ij}\} \in E$
 9:         $\mu_{ij}^{(n)} \leftarrow R_i^{(n-1)} \setminus \{\, \lambda_{ji}^{(k)} \mid k \in [1 \mathinner{..} n-1] \,\}$
10:     for each $\{x_i, f_{ij}\} \in E$
11:         $\lambda_{ij}^{(n)} \leftarrow \max \{\, \alpha s_t \mid s_t \in \mu_{ij}^{(n)}, t < t_{ij} + \beta \,\}$
12:     for each $x_i \in X$
13:         $R_i^{(n)} \leftarrow$ top $K$ of $\{\, \lambda_{ji}^{(n)} \mid f_{ij} \in N_i \,\}$
14:     for each $x_i \in X$
15:         $r_i^{(n-1)} \leftarrow r_i^{(n)}$
16:         $r_i^{(n)} \leftarrow \max R_i^{(n)}$
17:     $n \leftarrow n + 1$
18: return $\mathbf{r}^{(n)}$
```

1.2 Asynchronous Risk Propagation

While straightforward to specify, Risk-Propagation, is not a viable implementation for real-world application, because it is an *offline algorithm* that requires all individuals' recent contacts and risk scores to run. As Ayday et al.

(2021) note, the centralization of health and contact data is not privacy preserving. An offline design is also computational inefficient and risks human safety. Specifically, most exposure scores may not change across invocations of Risk-Propagation, which implies communication overhead and computational redundancy. As a mitigation, Ayday et al. (2020) suggest running Risk-Propagation only once or twice daily. However, this causes substantial delay in reporting to individuals their exposure scores; and in the face of a pandemic, timely information is essential for individual and collective health.

To address the privacy limitations of Risk-Propagation, Ayday et al. (2021) propose decentralizing the factor graph such that the processing entity (e.g., mobile device or "personal cloud") associated with the i-th individual maintains the state of the i-th variable node and its neighboring factor nodes. But for real-world application, the proposal leaves key questions unanswered:

1. Is message passing synchronous or asynchronous?
2. How does message passing terminate?
3. Are any optimizations utilized to reduce communication cost?
4. How do processing entities exchange messages over the network?
5. How private is decentralized risk propagation?

1.2.1 Actor Behavior

The Create-Actor operation initializes an actor (Agha, 1985). An actor a has the following attributes.

- *a.exposure*: the current exposure score of the individual that this actor represents. 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 B) of contacts. In the context of an actor, a contact is a *proxy* (Gamma et al., 1995) 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 $c.t$ is the most recent time of contact. This attribute extends the concept of *actor acquaintances* (Agha, 1985; Hewitt, 1977; Hewitt and Baker, 1977) 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 with the given query interval. Figure 1.3 depicts a hypothetical step function that *a.scores* represents.

```
NULL-RISK-SCORE
1: $s.value \leftarrow 0$
2: $s.t \leftarrow 0$
3: $s.sender \leftarrow$ NIL
4: return $s$
```

Note that CREATE-ACTOR does not specify a name for the actor. This

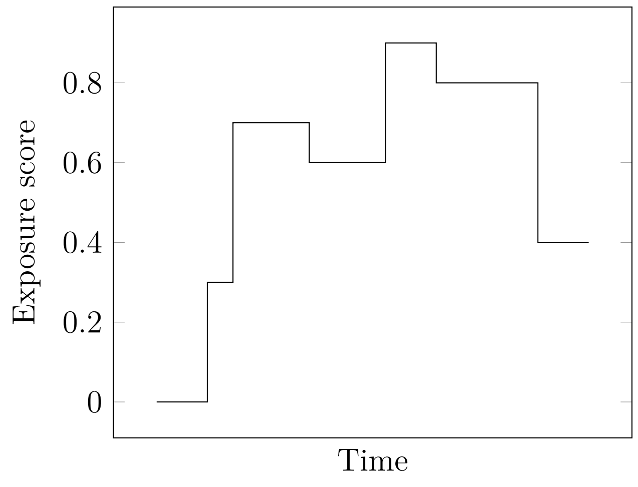

Figure 1.3: Historical exposure scores of an actor.

```
CREATE-ACTOR
1: $a.contacts \leftarrow \emptyset$
2: $a.scores \leftarrow \emptyset$
3: $a.exposure \leftarrow$ NULL-RISK-SCORE
4: return $a$
```

allows the actor to have multiple names and for them to be specified "out-of-band."

The interface of an actor is primarily defined by two types of messages: contacts and risk scores. As with Section 1.1, contacts and risk scores have finite relevance. 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($s$)
1: return $T_s - (\tau - s.t)$
```

The HANDLE-RISK-SCORE operation defines the actions an actor performs

```
CONTACT-TTL($c$)
1: return $T_c - (\tau - c.t)$
```

upon receiving a risk score. The key initially associated with the risk score is the time interval for 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, $s.key \subseteq [s.t, s.t + T_s)$ for all exposure scores in *a.scores*. The UPDATE-EXPOSURE-SCORE operation describes how *a.exposure* is updated. The dictionary *a.contacts* is assumed to only contain unexpired contacts. Additional context is needed before specifying APPLY-RISK-SCORE in detail. For now, it is sufficient to know that the operation uses the risk score to update the state of a contact.

```
HANDLE-RISK-SCORE($a, s$)
1: if RISK-SCORE-TTL($s$) $> 0$
2:     $s.key \leftarrow [s.t, s.t + T_s)$
3:     MERGE($a.scores, s$)
4:     UPDATE-EXPOSURE-SCORE($a, s$)
5:     for each $c \in a.contacts$
6:         APPLY-RISK-SCORE($a, c, s$)
```

```
UPDATE-EXPOSURE-SCORE($a, s$)
1: if $a.exposure.value < s.value$
2:     $a.exposure \leftarrow s$
3: else if RISK-SCORE-TTL($a.exposure$) $\leq 0$
4:     $a.exposure \leftarrow$ MAXIMUM($a.scores$)
```

For the moment, assume that Apply-Risk-Score is equivalent to computing a factor message (see Section 1.1). Line 2 indicates that the risk score s is copied and updated.

```
Apply-Risk-Score($a, c, s$)
1: if $c.t + \beta > s.t$
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.* For asynchronous risk propagation to be scalable and cost-efficient, actors should only send risk scores that offer new information to other actors. 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.

The objective of sending a risk score to another actor is to update its exposure score. Based on Handle-Risk-Score, it is only necessary to send an actor risk scores with values greater than its current exposure score. However, an actor is only privy to the risk scores that it sends. Thus, an actor can associate a *send threshold* for a contact that must be exceeded for a risk score to be sent to the target actor. To permit a trade-off between accuracy and efficiency of asynchronous risk propagation, let the *send coefficient* $\gamma \in \mathbb{R}$ be a scaling factor that is applied to a risk score upon setting the send threshold.

Below is the definition of Apply-Risk-Score that incorporates this new aspect of message passing. Assuming a finite number of actors, a positive send

```
SET-SEND-THRESHOLD($c, s$)
1: $s'.value \leftarrow \gamma \cdot s.value$
2: $c.threshold \leftarrow s'$
```

coefficient guarantees that a risk score has finite propagation.

```
APPLY-RISK-SCORE($a, c, s$)
1: if $c.threshold.value < s.value$ and $c.t + \beta > s.t$
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 this update behavior. The latter predicate of Line 1 stems from the fact that the send threshold is a risk score and thus subject to expiry. The former predicate is more subtle and will be revisited shortly. The MAXIMUM-OLDER-THAN is the same as MAXIMUM with the additional restriction that the interval key intersects with the query interval $(-\infty, c.t + \beta)$. In this way, the returned risk score is always relevant to the contact. As with APPLY-RISK-SCORE, the risk score retrieved from *a.scores* is also scaled by the transmission rate and set as the new send threshold.

The UPDATE-SEND-THRESHOLD is invoked during APPLY-RISK-SCORE, so another modification to the operation is defined below.

Returning to the first predicate on Line 1 of UPDATE-SEND-THRESHOLD, there are two cases in which the send threshold has a value of 0:

```
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($a.scores, c.t + \beta$)
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
```

```
Apply-Risk-Score($a, c, s$)
1: Update-Send-Threshold($a, c$)
2: if $c.threshold.value < s.value$ and $c.t + \beta > s.t$
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
5:     Send($c.name, s'$)
```

1. when initially assigning the send threshold to be the null risk score; and
2. when no key interval in *a.scores* intersects the query interval, and thus the null risk score again is assigned the send threshold.

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 contact a relevant risk score. In the worst case, this prevents *any* risk score from being sent to the target actor, thus providing the individual associated with target actor a false sense of security that they are not at risk of being infected. From a broader message-passing perspective, updating the send threshold to a non-null risk score *before* applying the risk score received by the actor may introduce a non-trivial amount of inaccuracy. To summarize, updating the send threshold only when its value is nonzero ensures correct message-passing behavior between actors.

Up until this point, the refinements to Apply-Risk-Score have focused on ensuring that message passing terminates and correctly adjusts over time, as risk scores expire. Prior to concluding this section, a message-passing optimization is introduced. Over a given period of time, an actor may receive several risk scores that are then propagated to multiple contacts. Intuitively, rather than sending multiple risk scores, it would be more efficient to send just the final risk score. To increase the likelihood of that this occurs, Apply-Risk-Score can be modified so that a contact "buffers" a single risk score that is intended to be sent to the target actor. Upon receiving a *flush timeout* message, the actor then "flushes" all contacts by sending the buffered message of each contact to the respective target actor. This is also known as *sender-side aggregation* in which the contact is an *aggregator* of risk scores.

The final iteration Apply-Risk-Score integrates sender-side aggregation. Moreover, Handle-Flush-Timeout clarifies the concept of "flushing" a contact. A flush timeout is assumed to be a periodically occurring "self" message.

```
Apply-Risk-Score($a, c, s$)
1: Update-Send-Threshold($a, c$)
2: if $c.threshold.value < s.value$ and $c.t + \beta > s.t$
3:     $s'.value \leftarrow \alpha \cdot s.value$
4:     Set-Send-Threshold($c, s'$)
5:     if $c.name \neq s.sender$
6:         $c.buffered \leftarrow s'$
```

```
HANDLE-FLUSH-TIMEOUT($a$)
1: for each $c \in a.contacts$
2:     if $c.buffered \neq$ NIL
3:         SEND($c.name, c.buffered$)
4:         $c.buffered \leftarrow$ NIL
```

To conclude the specification of actor behavior, the HANDLE-CONTACT operation indicates how an actor responds when a new contact or contact with a newer contact time is received. Similar to HANDLE-RISK-SCORE, expired contacts are not processed. The MERGE operation for *a.contacts* differs from how its used for *a.scores*. Specifically, if a contact with the same key already exists, its contact time is updated to that of the new contact; all other state of the previous contact is maintained.

```
HANDLE-CONTACT($a, c$)
1: if CONTACT-TTL($c$) $> 0$
2:     $c.threshold \leftarrow$ NULL-RISK-SCORE
3:     $c.buffered \leftarrow$ NIL
4:     $c.key \leftarrow c.name$
5:     MERGE($a.contacts, c$)
6:     $s \leftarrow$ MAXIMUM-OLDER-THAN($a.scores, c.t + \beta$)
7:     APPLY-RISK-SCORE($a, c, s$)
```

1.2.2 Message Reachability

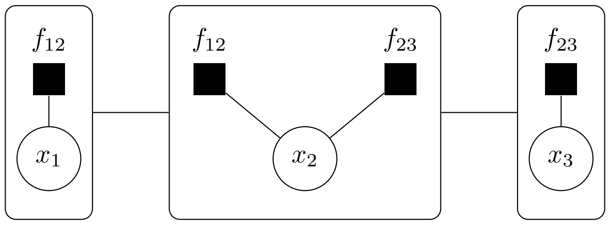

Figure 1.4: One-mode projection onto the variable nodes in Figure 1.1.

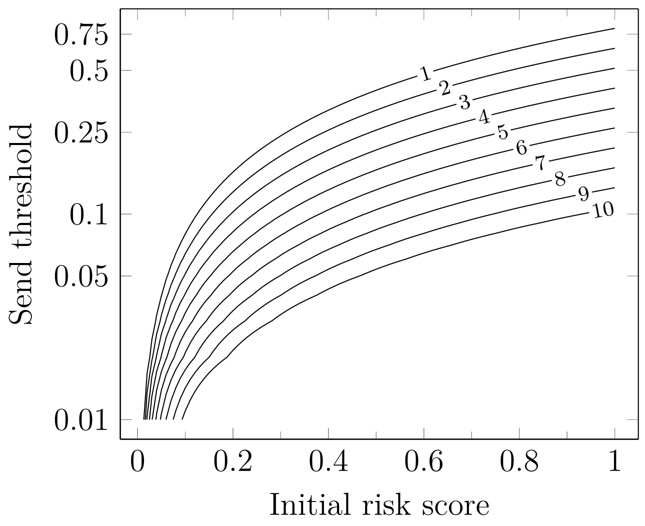

Figure 1.5: Estimated message reachability. Contour lines are shown for integral reachability values. Given an initial risk score and message reachability, a contour line indicates an upper bound on the permissible send threshold.

Appendix A

Pseudocode Conventions

Pseudocode conventions are mostly consistent with Cormen et al. (2022).

- 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 B

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 (Cormen et al., 2022).

- 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.

Bibliography

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

Erman Ayday, Fred Collopy, Taehyun Hwang, Glenn Parry, Justo Karell, James Kingston, Irene Ng, Aiden Owens, Brian Ray, Shirley Reynolds, Jenny Tran, Shawn Yeager, Youngjin Yoo, and Gretchen Young. Sharetrace: A smart privacy-preserving contact tracing solution by architectural design during an epidemic. Technical report, Case Western Reserve University, 2020. URL `https://github.com/cwru-xlab/sharetrace-papers/blob/main/sharetace-whitepaper.pdf`.

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.

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

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, fourth edition, 2022.

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

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, 1995.

Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, 1977. doi:10.1016/0004-3702(77)90033-9.

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`.

Frank R. Kschischang, Brendan J. Frey, and Hans A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, 47, 2001. doi:10.1109/18.910572.

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. doi:10.1038/s41591-020-0916-2.