Human Mobility and Location Privacy in Wireless Sensor Networks

by

Ryan Andrew Vogt

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

By eavesdropping on a user's query in a sensor network, an adversary can deduce both the user's current location and his/her location of interest. Issuing k queries instead of one (our "k-query" scheme) protects the privacy of the user's location of interest, but facilitates the adversary determining the user's current location. We propose a formal method for measuring how well issuing k queries to locations dispersed throughout the network protects the privacy of the user's location of interest, as well as a quantitative measure of how much information the k queries leak about the user's current location. Experiments reveal that how physically dispersed the k queries are has no meaningful effect on the user's privacy. However, there is a direct trade-off between the user's location-of-interest privacy and his/her current-location privacy, controlled by the value of k the user chooses.

User interactions with sensor networks do not occur in featureless, uniform environments. To facilitate the study of our k-query scheme in a rich environment characterized by realistically mobile users, we developed a new generative mobility model to produce mobility data for simulated agents. Existing generative mobility models suffer from a number of limitations. Most significantly, existing models are not representative of actual human movement. Our new mobility model is based on state-of-the-art work in understanding pedestrian mobility patterns in urban areas, known as Space Syntax. Under our model, agents move in a meaningful fashion in terms of destination selection and pathfinding, constrained by their surroundings in an outdoor urban environment. Results obtained from our publicly available Destination-Based Space Syntax Simulator (DBS3), independent from our k-query experiments, demonstrate which mobility model parameters affect wireless network simulations in general: the pathfinding metric in grid-based urban centres and centrality bias in other urban centres.

We combined DBS3 with our k-query scheme in order to study how long in advance a user should issue the k queries if travelling from some current location to his/her location

of interest. While the exact threshold depends on the urban environment and speed of the agents in question, the typical threshold is very low, e.g., 10 minutes when using $k = 3$ in downtown Edmonton, Canada.

Preface

Chapter 2 is based on a previously published paper: Ryan Vogt, Ioanis Nikolaidis, and Pawel Gburzynski. A realistic outdoor urban pedestrian mobility model. *Simulation Modelling Practice and Theory*, 26:113–134, 2012. Ryan Vogt was responsible for the design of the mobility simulation system described therein, the implementation of that system, experimental design, analysis, and manuscript composition. Ioanis Nikolaidis and Pawel Gburzynski were supervisory authors, and were involved with concept formation and manuscript composition.

Chapter 3 is a significant extension of a previously published paper: Ryan Vogt, Mario Nascimento, and Janelle Harms. On the trade-off between user-location privacy and queried-location privacy in wireless sensor networks. In *Proceedings of the 8th International Conference on Ad-Hoc Networks and Wireless*, pages 241–254, 2009. Ryan Vogt was responsible for the design and implementation of the experiments, analysis, and manuscript composition. Mario Nascimento and Janelle Harms were supervisory authors, and were involved with concept formation and manuscript composition.

Dedication

To my beautiful and brilliant wife, Stefanie.

Acknowledgements

Thank you to the Natural Sciences and Engineering Research Council of Canada, and iCORE and Alberta Advanced Education & Technology for funding this research in-part. Thank you also to Ioanis Nikolaidis, Mario Nascimento, and Pawel Gburzynski for supervising me through the arduous journey that was this degree.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1 Introduction

Privacy is an important challenge in many wireless sensor network applications. Consider a sensor network where a user with a portable device interacts with the sensor nodes for the purpose of querying a sensor's data at a remote location. An often-used practical example is that of a sensor network in a battlefield deployment being queried by troops. Alternately, the user could be a pedestrian with a handheld device, querying an urban deployment of sensors regarding, e.g., crowd densities. The user's query would be received by a nearby node and routed through the sensor network to the location of interest, then processed and returned. Given an assumed sensitive nature of the information returned by the query, encryption should be used to protect against eavesdropping. Either symmetric encryption or public-key encryption that is sufficiently inexpensive to be performed by a low-power sensor node [42] could be used.

However, while cryptography can ensure data confidentiality, encryption alone cannot provide anonymity against an adversary capable of compromising some of the nodes in a sensor network. By reading the headers of traffic in the sensor network at compromised nodes, an adversary could learn: (a) *where the queried sensor is*, possibly providing significant insight into the user's intentions; and (b) *where the query originated*, thus revealing the user's current location.

One contribution of this dissertation, presented in Chapter 3, is to present an effective method to obfuscate the user's location of interest by having the user issue not just one

query to the location of interest, but instead k queries to a diverse set of locations (our "k-query" scheme). The trade-off to this approach, aside from the energy cost of processing an additional $k-1$ queries, is that the additional traffic from the user's current location facilitates an adversary determining that location. That chapter will study this trade-off between protecting the privacy of the user's current location and that of their location of interest.

Initially, when we start studying the k-query approach, we will consider the sensor network to be in a featureless environment. However, real sensor networks do not exist in featureless environments, devoid of context. People move in their environments. Is it the case, then, that a user's current location is indicative of the location in which they are interested — potentially, to which they are travelling? To facilitate studying the k-query approach in this richer context, we will actually begin this dissertation with its other key contribution: the generation of natural patterns of human mobility in an outdoor urban environment. In Chapter 2, we will introduce the concept of Space Syntax [26], which seeks to understand how people perceive distance and move in their environment. Drawing from the theory of Space Syntax, we have built the Destination-Based Space Syntax Simulation (DBS3), a human-mobility simulator capable of generating realistic movement patterns in outdoor urban environments.

After we have presented both DBS3, and the k-query approach (both in a featureless environment, and in the more interesting context generated by DBS3), we will conclude in Chapter 4.

1.2 Key Contributions to Mobility Simulation

DBS3, using formulas inspired by Space Syntax, realistically simulates the movement of people in an outdoor urban environment. The biggest advantage of DBS3 over existing approaches is that it does so with a small set of input parameters.

DBS3 comes with two maps, Edmonton, Canada and Fira, Greece, which form the only non-trivial input to DBS3. While agent speeds and pause times can be changed from their default values, there is no need to do so — agents already choose their speed at random, normally distributed over average walking speed, and choose their pause times at locations with a realistic log-normal distribution. Agents choose their destinations realistically, tend-

ing to head to well-connected locations that are nearby, but there is no need to tag the map or input any sort of metadata about the map or agents involved. Agents also move realistically to their destination, minimizing the number of turns they take or the magnitude of those turns, even at the cost of additional Euclidean distance (as real people in urban environments tend to do). In order to compute this kind of pathfinding, we developed the MEA* pathfinding algorithm, which is an extension to the classical A* algorithm [23] that minimizes Euclidean distance travelled. We also developed an improvement to MEA* called StreetCut, that reduces its running time with no cost to accuracy.

The only additional inputs to DBS3 are two tuning parameters, which control the strength of centrality bias — that is, the tendency of people to travel to well-connected locations — and distance decay — the tendency of people not to want to travel far from their current location. Both of these tuning constants are input as real numbers. We compared the distribution of people on real streets in downtown Edmonton to the distribution of agents among the streets in our simulated environment. Using our new approach with both tuning parameters set to 1.0 (the default values) produced a correlation with real Edmonton of $R^2 = 0.96096$. On the other hand, the classic approach of simulating agents by minimizing Euclidean distance travelled and not considering centrality bias and distance decay in destination selection produced a correlation with the real Edmonton of only $R^2 = 0.75324$. The only significant error made in DBS3's distribution of agents around downtown Edmonton is that DBS3 predicted there would be something that would attract a large number of agents in one specific area of the map, namely where Grant MacEwan University is located. However, our correlational study was performed in August, when classes were not in session. While this result demonstrates one limitation of our non-tagging approach to mobility simulation, the high correlation on the rest of the map mitigates this limitation.

DBS3 is a freely available. It can be used to produce mobility data en masse, or it can be used with the included GUI to visualize the results of a single experiment on a map. It includes libraries for integrating existing programs with either of those uses of mobility data.

1.3 Key Contributions to Network Privacy

Our proposal of issuing k diverse queries, as opposed to just one, is a simple yet effective mechanism for protecting the privacy of a user's location of interest. One of the largest advantages to this approach is that it works well in any sensor network, not requiring any special programming or hardware in the sensor nodes that process the queries.

One key contribution of this work is to formalize three metrics to quantify how well a user's privacy is maintained. These metrics quantify how well the privacy of the user's current location is maintained, how well the privacy of the user's location of interest is maintained, and how well the privacy of the general area of the sensor network in which the user is interested is maintained. As part of studying those metrics, we created an efficient algorithm that could be used by an adversary not only to analyze intercepted messages sent by the user, but also to analyze messages that are sent and not intercepted by the adversary.

We also discuss the security requirements of how the user must choose the destinations of the $k - 1$ fake queries. The user is required to partition the sensor network, and always query the k sensors within the partition of the sensor they actually wish to query. This result, in turn, led us to study two problems that we call PARTITION and PARTITIONING. We demonstrate that these are NP-complete problems on a graph. We demonstrate a 2-approximation for the partition optimization problem. We also designed a divide-and-conquer approach to approximate an answer to the partitioning problem, which runs significantly more quickly than a brute-force approach.

We start our discussion of the security results in a featureless environment in which the popularity of all locations is equal. In this simple environment, we show the direct trade-off between the privacy of a user's current location and their location of interest, controlled directly by the user's choice of k. Other features that may be beyond the user's control, such as the type of cryptography and the type of routing used in the sensor network, also play a role.

We also study the security results in a more feature-rich environment. Using DBS3, we simulated agents moving around a map, and used the distribution of agents to gauge the relative popularity of different locations on the map. Using that information, the adversary was able to significantly improve the guesses about the user's current location and (especially) location of interest. We also study what happens if the adversary is able to assume

that the user is moving towards their location of interest. In that case, the adversary has an additional advantage, unless the user takes additional time before arriving at their location of interest. For example, in downtown Edmonton, an adversary is able to predict more accurately where someone is going to be, knowing only where they currently are, for approximately fifteen minutes. Knowing that the user is travelling to their location of interest translates into an advantage for an adversary attempting to guess that location, e.g., with $k = 3$ in downtown Edmonton, for approximately ten minutes.

Chapter 2

Human Mobility Simulation[2.a]

2.1 Introduction

To study the relationship between human mobility in featured non-homogeneous environments and location privacy, we require a source of mobility data for such users (called "agents" in mobility modelling literature). Trace data obtained from tracking actual movement of real people is an appropriate source of such mobility data, which has been used, e.g., by González et al. [18]. However, it is difficult to collect accurate mobility traces for a large number of people, each individually tracked over a lengthy period of time. Using an electronic device like a cellular phone or laptop as a proxy for each person, for example, yields trace data of limited granularity, with successive locations inferred from associations with access points or cellular towers. Additionally, the collection of trace data is limited to those times when the device is both active and can communicate with the fixed infrastructure.

However, the most pressing issue when resorting to mobility traces is that traces collected in one environment do not generalize to different spatial configurations,[2.b] i.e., to a different building, campus, or city layout. In sharp contrast, generality is a strength of generative mobility models. A generative mobility model is an algorithm that, using an entropy source (e.g., a pseudorandom number generator), produces simulated movement for each agent in an environment. The random waypoint mobility model [33] is one such generative model. Agents moving according to this model proceed in straight lines to randomly chosen destina-

[2.a] This chapter is based on Vogt et al. [62].

[2.b] We use the terms "configured" and "configuration" to mean any environment where not every point is accessible by a human, i.e., an environment with rigid obstacles to free movement.

tions in a featureless environment, pausing at each destination for a random amount of time before choosing a new destination and speed. The random waypoint model is an instance of the more general random trip model [41], in which agents moving in a bounded, connected domain move to random destinations according to predetermined destination-selection and mobility rules. Numerous mobility models similar to random waypoint exist, and they have been surveyed extensively [6].

It is well understood that random waypoint and models directly derived from it are not ideal mobility models to use in a simulation. There are mathematical concerns with these models, such as average agent speed continuously decaying over time in the absence of a positive lower bound on random agent speeds [69]. But the largest problem with random waypoint is a fundamental one: it is not a realistic representation of human movement. People do not move in unobstructed lines to random locations in a featureless environment. This problem is more than aesthetic, since the choice of mobility model can significantly affect the results of a network simulation [6]. Despite these issues, the random waypoint mobility model continues to be used [60, 66].

The key problem we address in this chapter is *how to combine the generalizability of a generative mobility model with the ability to honour both the restrictions and meaningful patterns of human mobility in configured spaces.* Needless to say, we want this combination to be practically useful in simulations, such as privacy studies. As the most important objective of a simulation study is a comparison of a number of systems under a representative range of inputs, we would like to minimize the population of cases constituting a defensible "representation." Therefore, an important — if somewhat informal — constraint is to limit reasonably the degrees of freedom in the parameter space. Our goal is thus to strike a workable balance between the desire to capture the essential patterns of human mobility and the ease of description of a modelled case.

One critical observation is that it does not suffice simply to move agents between random points on an outdoor street map by routes that minimize Euclidean distance travelled. Human movement is not characterized by such simple rules, because these rules do not reflect the mental effort that goes into human navigation [44]. Our problem thus becomes *finding a theory that describes human mobility adequately and that can be used to derive an easily and naturally parameterizable algorithmic, generative mobility model.* We adopt the view on how configured space influences human mobility as it is articulated in the

theory of Space Syntax [26]. Space Syntax is now a relatively mature theory, focused on the design of urban (built) environments. Potential lines of movement unobstructed by rigid obstacles, called *axial lines*, collectively define the *axial map* of an area. Space Syntax seeks to understand how people perceive distance, move, and cluster on any axial map. The key result from Space Syntax is that the most heavily frequented locations in an urban environment are those that are better "integrated" — that is, locations that are a short distance away from many other locations. This observation allows us to approach the construction of the mobility model by relying strictly on the graph-theoretic properties of the map that the agents populate, without the need for additional metadata such as the locations and characteristics of popular shops or other attractions. While those attributes are not irrelevant for the accuracy of description, their inclusion would significantly complicate the model by introducing a number of fuzzy parameters, thereby greatly reducing the model's practical appeal. It is in fact one of the most interesting features of our approach that all those "popularity" parameters of the various spots on the map are implicitly captured by their purely graph-like connectedness with other spots. This is because highly frequented attractions cannot be located in poorly connected areas — the natural feedback cycle of urban development takes care of the requisite correlation. A second key result from Space Syntax is that humans do not perceive the distance between two locations as the Euclidean distance that must be travelled between them. Rather, we perceive distance as some measure of the change in direction of travel that we must make to move between the two destinations. There's no clear answer to how we perceive change in direction of travel, but the total number of changes in direction and the total magnitude of change in direction are two good candidates.

Based on these two key results, we built the Destination-Based Space Syntax Simulator (DBS3). DBS3 is a freely available,[2.c] high-performance mobility simulator for agents in outdoor urban environments. Section 2.2 describes how DBS3 is distinct from related work on realistic mobility models. Section 2.3 continues by describing the design of our new mobility model, including the implementation details of DBS3 — in particular the new multi-expansion A* search (MEA*) used to produce optimal non-Euclidean-distance paths. We demonstrate how the input parameters to DBS3 affect the results of some model simulation studies in Section 2.4. Section 2.5 then verifies the correctness of the mobility model

[2.c] `http://www.cs.ualberta.ca/~vogt/dbs3.html`

by showing high correlation between the mobility patterns produced in DBS3 and observed pedestrian patterns in downtown Edmonton. Finally, we describe how to use DBS3 in Section 2.6, before concluding our discussion on mobility simulation and discussing potential future expansion on this topic in Section 2.7.

2.2 Related Work

There are two main existing methodologies for building realistic mobility models. The first methodology is to extrapolate a generative mobility model from traces of actual human movement. Yoon et al. [71] capture coarse-grained trace data using Wi-Fi connectivity logs on a campus. This coarse-grained data is subsequently broken into trips between ordered pairs (i.e., origins and destinations) of enumerated locations on a campus map. The n-minimal set of paths between origin and destination pairs, measured by Euclidean distance with ties broken arbitrarily, form the set of possible routes used by their generative model. That approach is subsequently extended [38] to consider the time of day, adding realism by considering when students on campus are more likely to travel between different locations. In fact, the mobility patterns of a fixed individual have a high level of predictability given the time of day [54].

The second existing methodology used to build realistic mobility models is to survey individuals to learn how they move in a given environment. Hsu et al. [30] manually defined locations of interest on a campus, then generated a Markovian transition model between these locations by conducting a survey of students. A more involved approach was used by Feeley et al. [12]. They developed a system that not only defined locations of interest, but also sets of activities to be performed by agents at those locations. What activities each agent would perform is determined by the role of that agent, which is drawn from a manually defined set. A similar but time-dependent approach [37] constructed a generative model using surveys of what activities people do at different times during a workday, along with an annotated map describing where offices, shops, and other building types are located.

The benefits of our Space Syntax-inspired approach over these existing methodologies are twofold. First, the pathfinding performed by DBS3 more accurately reflects how humans perceive distance. Using the new MEA* algorithm (discussed in Section 2.3.3), agents can move in a manner that minimizes either the number of changes in direction that they

make or the total magnitude in the change of their direction, as opposed to the traditional approach of minimizing the Euclidean distance travelled. Second, agents in DBS3 choose their sequence of destinations without needing the map on which they move to be annotated. There are no manually defined locations of interest, and there is no need to define where different kinds of buildings are located. Instead, the graph-theoretic properties of the street layout directly define where the most desired destinations are located. Our approach avoids the notions of preset waypoints and flights by deriving them from the untagged map that becomes the sole description of the geometry, i.e., destination points and their bias. As such, our approach generalizes easily across different maps, not requiring new trace data or surveys for new locales. The most important, novel, and unique feature of our approach is that it yields a truly flexible generative method whose only nontrivial input is a map of the studied area. Yet, despite the simplicity of parameterization, DBS3 still realistically reflects how humans perceive distance and how they select destinations and routes.

To the best of our knowledge, there have been two previous attempts at constructing generative mobility models based on Space Syntax. The first, by Dalton and Dalton, is admitted to be a "tentative 'proof of concept' pilot study in order to determine whether the theory of natural movement could be utilized in...a simulation" [9]. Agent destinations are chosen uniformly at random; only the initial locations of agents are influenced by how well-integrated those locations are. Such a simulation will drift into a steady state that has no relation to the initial integration-based distribution of agents (though, as a point of future study, it would be interesting to see if the initial stage of this simulation accurately reflects more specific cases, such as that of people leaving work at the end of the day — mostly starting in well-integrated locations, then drifting out). The second effort, by Kostakos et al. [40] and Kostakos [39], also left a few important loose ends. It does not express the effect of each individual agent's pathfinding: agents simply roam along axial lines, presumably using random selection at intersections to decide where to go next. As a result, the phenomenon of aggregate mobility patterns over a map — strongly connected to how individuals find their way between locations on that map — is lost. The actual result is more like a snapshot in time of agent locations that subsequently drifts, without maintaining the aggregate pattern first seen in that snapshot. In our work, we preserve a per-agent notion of destination selection and pathfinding, and a well-defined steady-state behaviour. In addition, our implementation is made publicly available to be scrutinized,

since a common problem with previous attempts is that the implementations have been tied to particular examples and do not appear to have been widely distributed as general purpose tools.

2.3 Mobility Model Design and Implementation

2.3.1 Agents and Maps

The mobility model used by DBS3 is an instance of a random trip [41] model. Agents choose a destination based on their current location, walk to their destination using a route generated by the pathfinder, then pause for a randomly generated amount of time at that destination before repeating this process. We start with an overview of how agents move in their environment, before discussing the details of the destination selection algorithm in Section 2.3.2, the pathfinding algorithms in Section 2.3.3, and simulation initialization in Section 2.3.4.

In DBS3, agents walk within a connected set of intersecting rectangular areas. For simplicity, we refer to these areas in which agents are permitted to move as "streets," because these areas correspond precisely to actual city streets in the two maps used in this dissertation. In general, however, these rectangular areas need not be streets; DBS3 could simulate agent movement only on sidewalks, or only within a connected set of thin rectangles representing the axial lines of an arbitrary convex space.

Normally, a Space Syntax-inspired simulator like DBS3 would need an axial map of the area to be simulated: a graph representation of the area in which each axial line in the minimal set is represented as a vertex in the graph, and edges connect vertices whose axial lines intersect. Automated generation of axial maps is a hard computational problem, but tools do exist [48, 59, 65]. However, because we are limiting our study to outdoor urban areas, we can assume that there are streets on which our agents will walk, and the streets themselves make a good approximation of the axial map of an urban area. Like Yoon et al. [71], we represented any curved street as a sequence of straight streets intersecting at angles that are neither overly sharp nor overly shallow, where visual approximation and intuition are used to define what is overly sharp or overly narrow.

The first map used in this dissertation is grid-like downtown Edmonton, Canada, illus-

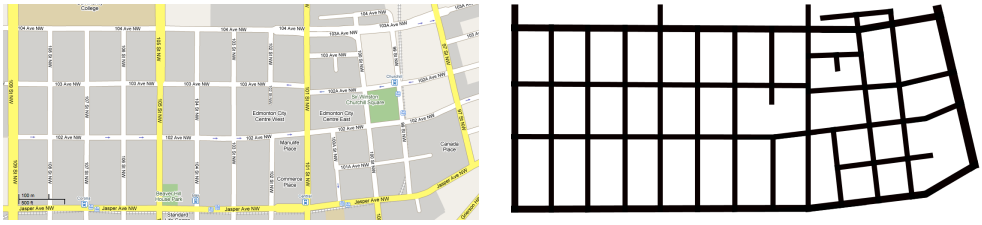


Figure 2.1: A map of downtown Edmonton, Canada from Google Maps [19], and the associated street outlines used as input to DBS3.

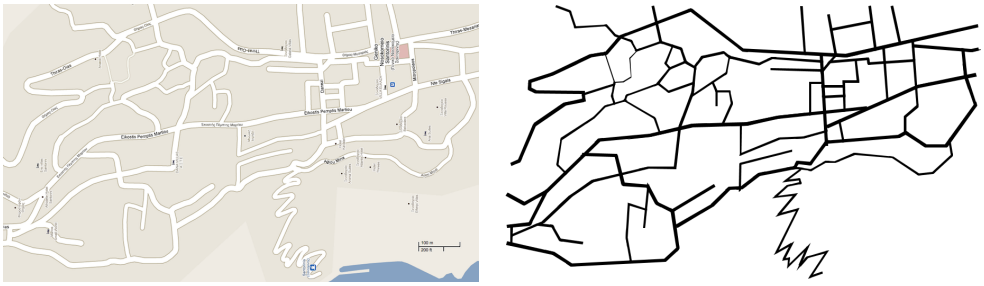


Figure 2.2: A map of Fira, Greece from Google Maps [19], and the associated street outlines used as input to DBS3.

trated in Figure 2.1. The second map we use is the more organic and graph-theoretically interesting Fira, Greece, shown in Figure 2.2.

Like many mobility models, DBS3's adheres to the invariant property that, at all times, there is a fixed population of agents in the simulation environment (a useful invariant in many wireless network simulations, allowing individuals and their mobile devices to be tracked over a long period of time). We denote this set of agents on the map as $\mathbb{A} = \{A_1, A_2, \ldots, A_m\}$. Each agent A_i has a fixed speed S_i when mobile, where S_i is chosen randomly according to the speed distribution $\mathcal{S}$. Upon reaching a destination, agent A_i will pause for a random amount of time t chosen according to the time distribution $\mathcal{T}$. Note that when A_i starts moving to a new destination after the pause of t, the value of S_i does not change. There are two reasons that we chose to keep each agent's speed constant for the duration of the simulation, rather than have agents choose new speeds at each destination. First, it eliminates speed decay — the potentially infinite decrease of average agent speed over the duration of the simulation (though it is also possible to eliminate speed decay by choosing each agent's initial speed from a steady-state speed distribution $\mathcal{S}'$, then choosing subsequent new speeds from the original distribution $\mathcal{S}$ [70]). But more importantly, we feel that our design is more realistic on a per-agent basis, compared to one in which a given

agent may be moving at geriatric speeds to one destination then sprint to the next.

Studies have shown that pedestrian speed is approximately normally distributed, with a mean of 1.52 m/s and a standard deviation of 0.23 m/s [24, 25]. We bound that distribution at three standard deviations from the mean (i.e., 0.83–2.21 m/s), and use the resulting distribution as the default speed distribution $\mathcal{S}$ in DBS3. It has also been demonstrated that pause times, i.e., the amounts of time people spend at destinations, are log-normally distributed [38]. By default, DBS3 uses the arbitrarily chosen pause range of 15–600 seconds with a log-normal distribution. There is some evidence that the speeds at which people move may be log-normally distributed as well, instead of normally distributed [38]. For this reason, DBS3 supports both bounded normal and bounded log-normal distributions (as well as the classic uniform distribution) for both $\mathcal{S}$ and $\mathcal{T}$.

While uniform random values can be generated in an obvious fashion, it is not as clear how to generate bounded normal or bounded log-normal random values. Normal and log-normal distributions are defined over infinite domains. When we generate normal or log-normal values, we must truncate those infinite domains before scaling the generated value to the bounds, which we will call $[min, max]$.

When generating normally distributed values, we truncate the distribution at three standard deviations in either direction from the mean. The domain of this truncated normal distribution contains over 99.7% of the values generated by a non-truncated normal distribution, so the effect on the shape of the distribution is negligible. Because both the distribution and truncation are symmetrical, the expected value of the truncated distribution is identical to the expected value of the infinite-domain distribution.

For consistency with how we generate normally distributed values, we truncate the log-normal distribution in such a way that the expected value is unaffected and the same approximately 99.7% of the original distribution's randomly generated values are contained in the truncated domain. When we generate random log-normal values by computing e^g, where g is a random Gaussian value with a mean of 0 and standard deviation of 1, values that are lower than approximately 0.06 or larger than approximately 44.31 are rejected.[2.d] The expected value of accepted values remains the same as the expected value of all generated values, $e^{1/2}$, and the same proportion of generated values are accepted as in the normal

[2.d] A more precise numerical representation of these bounds can be found in the DBS3 code. We are unaware of a closed-form representation of these bounds.

distribution's truncation.

Pseudocode for how these bounded values are generated is provided as Algorithm 2.1, where GenerateNormal$(0.83, 2.21)$ is the call used to generate values in the default speed range and GenerateLogNormal$(15, 600)$ is for the default pause range. The function $\mathcal{U}(0, 1)$ returns a random uniformly distributed number between 0 and 1, and the function $\mathcal{G}(0, 1)$ returns a random normally distributed unbounded value with a mean of 0 and a standard deviation of 1.

Algorithm 2.1 Generation of bounded, random values in DBS3

```
1:  function GenerateUniform(min, max)
2:      p ← U(0, 1)                                   ▷ Uniform random value in [0, 1]
3:      return min + (max − min) · p

4:  function GenerateNormal(min, max)
5:      repeat
6:          g ← G(0, 1)                               ▷ Normal random with μ = 0, σ = 1
7:      until −3 ≤ g ≤ 3
8:      p ← (g + 3)/6
9:      return min + (max − min) · p

10: function GenerateLogNormal(min, max)
11:     lb ← 0.06134160902282682206561692291493027192911068582928926771205592947
12:     ub ← 44.31383316742639161294271979219893347698550976490274713986352064 04
13:     repeat
14:         g ← G(0, 1)                               ▷ Normal random with μ = 0, σ = 1
15:         l ← e^g
16:     until lb ≤ l ≤ ub
17:     p ← (l − lb)/(ub − lb)
18:     return min + (max − min) · p
```

2.3.2 Destination Selection

One key result in Space Syntax is that better-integrated locations, i.e., those locations that are close to many other locations, see more pedestrian traffic than poorly integrated locations [27]. Some of this effect comes from through-movement — movement through well-integrated streets, as people walk along thoroughfares to get to distant locations. However, this effect is also fuelled by to-movement — that is, well-integrated locations are more likely to be destinations for people than poorly integrated ones.

Within the confines of to-movement, there are two different factors at play. The first is that well-integrated locations are inherently more popular than poorly integrated locations.

We refer to this observation as *centrality bias*. As a concrete example, a restaurant (a popular destination for many people) is typically located along a better-connected roadway, whereas a house (a destination for very few people) is often located in a less-central location. The second factor at play is *distance decay*. Distance decay is the reluctance of people to travel larger distances from their current location to reach a destination. Returning to our example, we expect a restaurant located in a well-connected area to be more popular than a restaurant located in a poorly connected area, just by virtue of being closer to more people.

One of the contributions of this dissertation is to formalize the interplay of centrality bias and distance decay into an algorithm that can be used to select the next destination for each agent in a mobility simulation based on its current location. That is, agents in DBS3 choose their next destination by balancing the tendency to travel to more popular locations (centrality bias) and the tendency to minimize the distance they have to travel from their current location (distance decay). Recall that destination selection is performed without any metadata about the map (e.g., where different types of buildings are located on the map); it is strictly the graph-theoretic properties of the map that matter. Also note that there is no concept of a final destination in DBS3; agents continue to choose new destinations in this random trip model.

We begin by assuming that the number of potential destinations on a street is proportional to the length of the street. Note that we are not yet describing the relative popularity of the destinations on two different streets; rather, we are describing the number of destinations on each street. Consider two streets, s and d. By our assumption, the probability of an agent who is currently on street s choosing a next destination on street d should be proportional to the length of d, $\mathrm{L}(d)$, factoring in how well integrated (i.e., central) street d is and how much distance decay there is from street s to street d. Formally, the potential P for an agent on street s to travel to street d is defined as

$$\mathrm{P}(s,d) = \frac{\mathrm{L}(d)}{\mathrm{I}(d)^{\alpha} \cdot \mathrm{D}(s,d)^{\delta}} \ , \tag{2.1}$$

where $\mathrm{I}(d)$ is a measure of how peripheral d is (with high values for poorly connected streets and low values for well-connected streets), and $\mathrm{D}(s,d)$ represents the human-perceived distance from street s to street d. The two exponents, $\alpha \in \mathbb{R}$ and $\delta \in \mathbb{R}$, are scaling factors given as input to DBS3, used to adjust the relative effects of centrality bias and distance

decay.

Recall from Space Syntax that humans do not perceive distance between two locations as the minimal Euclidean distance that must be travelled between the two. Perception of distance is better correlated with the number of changes in direction that have to be made, or the sum of the magnitudes of change in direction. We use the number of changes in direction as our measure of distance in DBS3's destination selection algorithm, since it is a unitless measure, unlike magnitude of change in direction. Formally, let $\mathrm{T}(s, d)$ be the minimum number of turns an agent would have to make to get from street s to street d. Then,

$$\mathrm{D}(s, d) = \mathrm{T}(s, d) + 1 \; . \tag{2.2}$$

Looking to Equation 2.1, in which $\mathrm{D}(s, d)$ is used in the denominator, note that $\mathrm{D}(s, d) \geq 1$.

We use this same measure of distance to compute how well the locations on each street are integrated. Following the work of Hillier and Iida [27], DBS3 computes how well integrated each potential destination is within its neighbourhood. The neighbourhood of any location is defined by a globally constant radius, $\rho \in \mathbb{Z} \geq 0$, representing the number of turns that a person can make from any given location until they perceive themselves to be in a new "area." The value of ρ expresses what humans find acceptable walking distance, yet the notion of distance is subjective — for example, according to the feature-accumulation hypothesis [44], it depends on the number of "features" encountered and memorized during locomotion along a given path, which is evidently a cognitive limitation. The feature-accumulation hypothesis explains some surprising effects [29], like the overestimation of distances by inner-ring urban residents and the underestimation of distances by outer-ring suburban residents. In short, both cognitive effects and map effects need to be accounted for in suggesting an appropriate value for ρ.

To calculate how well connected the locations on street d are to other locations in the neighbourhood of d, we compute the average distance to each location in the neighbourhood (recalling that the number of locations on each street is proportional to the length of that street) as

$$\mathrm{I}(d) = \frac{\sum_{i \;\mid\; \mathrm{T}(i,d) \leq \rho} (\mathrm{L}(i) \cdot \mathrm{D}(i, d))}{\sum_{i \;\mid\; \mathrm{T}(i,d) \leq \rho} \mathrm{L}(i)} \; . \tag{2.3}$$

Again, for use in Equation 2.1, note that $1 \leq \mathrm{I}(d) \leq \rho + 1$. A large value for $\mathrm{I}(d)$ means that the destinations on street d are poorly connected within their neighbourhood, whereas

a small value indicates that street d is highly connected.

Note that there are two implicit assumptions in these equations. First, it is assumed that $\mathrm{D}(s, d)$ is finite for any pair of streets s and d, i.e., that any street is reachable from any other street. Second, it is assumed that all streets have positive, finite length. DBS3 enforces both of these assumptions.

Equations 2.1–2.3 are used to choose new destinations for agents in DBS3. Specifically, after an agent reaches a destination on street s and completes its pause time, it chooses the street on which its next destination will be located proportionally to the value of $\mathrm{P}(s, d)$ over all possible streets d. This behaviour is formalized in Algorithm 2.2; to select a new destination, an agent calls GetNewDestination(*curLoc*), where *curLoc* is its current location.

Algorithm 2.2 The destination-selection algorithm in DBS3

```
1:  function GetNewDestination(curLoc)
2:      s ← the street on which curLoc is located
3:      n ← the number of streets on the map
4:      for i ← 1 to n do
5:          d ← street number i on the map
6:          A[i] ← P(s, d), computed as per Equation 2.1
7:      index ← ProportionalIndex(A)
8:      str ← street number index on the map
9:      pnt ← a uniform random point on str
10:     return point pnt on street str

11: function ProportionalIndex(array)
12:     n ← the length of array
13:     sum ← Σ_{1≤i≤n} array[i]
14:     cdf[1] ← array[1]/sum
15:     for i ← 2 to n do
16:         cdf[i] ← array[i]/sum + cdf[i − 1]
17:     r ← U(0, 1)                          ▷ Uniform random value in [0, 1]
18:     return min{i | r ≤ cdf[i]}
```

Note that if either α or ρ is zero, centrality bias has no effect on destination selection. If δ is zero, distance decay has no effect on destination selection. If neither centrality bias nor distance decay has an effect, the selection algorithm degenerates into a uniform destination selection algorithm in which all destinations are equally likely to be chosen as the next destination.

By default, DBS3 sets $\alpha = \delta = 1$. Because the maps of downtown Edmonton and Fira both represent small urban areas (or small parts of a larger map), we use $\rho = \infty$. It is

outside of the scope of this dissertation to suggest a specific ρ value for any appreciably larger maps. Hence, we leave ρ up to the discretion and control of the user, with the suggestion that $\rho = \infty$ is acceptable for small maps. Potential future work could even attempt to find a distribution for ρ, with each agent drawing a personalized value for ρ that essentially reflects how familiar they are with a given map (e.g., whether they are a local resident or a visitor).

2.3.3 Pathfinding Algorithm

With the algorithm by which agents can choose a subsequent destination after arriving and pausing at a previous destination complete, we now present an algorithm that agents can use to compute the paths that they will follow between destinations. According to Space Syntax, paths to a destination that minimize either the number of changes in direction or the total magnitude of all the changes in direction are the best-correlated paths to those taken by actual humans. As such, DBS3 supports pathfinding between destinations that minimizes the number of changes in direction or minimizes the sum of the angles of the changes in direction (as well as, for completeness, the classic pathfinding approach of minimizing Euclidean distance travelled). Note that the pathfinder that minimizes the total change in angle should not be confused with search algorithms that attempt to generate smoother paths while still minimizing Euclidean distance, e.g., Theta* [10].

There are two key observations about human movement that influenced how we translated these high-level descriptions of human pathfinding into formal algorithms: (a) human movement lacks pinpoint precision; and, as such, (b) human movement is not perfectly optimized. Consider the street network in Figure 2.3 overlaid with two potential paths from a source to a destination. The angle values in the image represent the change in angle of movement at each change in direction. Given these two potential paths, an agent that minimizes total change in angle would always choose the solid path over the dashed path, as it incurs 2ϵ less in angle change.

However, humans rarely walk directly to the centre of the intersection between two streets or sidewalks, nor do they walk to the exact corners of intersections. As such, a realistic simulated agent seeking to produce paths with low accumulated angle change should not always use the streets corresponding to the solid path in Figure 2.3 instead of the streets

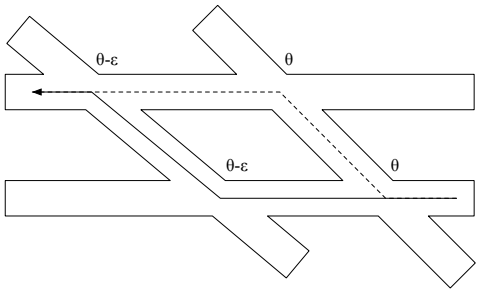


Figure 2.3: Two potential paths from a source to a destination, labelled with the angle changes incurred during each change in direction.

corresponding to the dashed path, in cases where ϵ is small. As an extreme example, consider the case where ϵ is so small as to be imperceptible to a human, but measurable to a computer simulating agents moving in this environment.

To prevent such over-optimizing of chosen paths and to add a small random element to agent behaviour at intersections, our pathfinding algorithm to determine a path from a point *src* to a point *dst* functions according to the following logic:

1. For each intersection i on the map, generate a random point p_i in that intersection (i.e., a new random point is generated in each intersection on every call to the pathfinding algorithm);

2. For each intersection i, mirror p_i into all four quadrants of the intersection, creating four (almost certainly, but not necessarily, distinct) points $p_{i,1}, \ldots, p_{i,4}$.

3. The agent is allowed to move to any point $p_{i,j}$ for any intersection i on its current street, or to *dst* if the destination is on its current street. Let $\mathbb{H}$ be the set, exponential in size to the number of intersections on the map, of all possible paths from *src* to *dst* constrained by these rules.

4. Choose path $h \in \mathbb{H}$ according to one of three rules. Which of the three rules is used is a simulation-global choice of the user:

   (a) $h \in \mathbb{H}$ has the minimal Euclidean distance;

   (b) $\mathbb{H}' \subseteq \mathbb{H}$ is the subset of paths with the minimal number of changes in direction and $h \in \mathbb{H}'$ has the minimal Euclidean distance; or,

   (c) $\mathbb{H}' \subseteq \mathbb{H}$ is the subset of paths with the minimal total magnitude of change in angle and $h \in \mathbb{H}'$ has the minimal Euclidean distance.

To implement this logic in DBS3, we developed an extension to the A* search algorithm [23] called the multi-expansion A* (MEA*) search. We start by presenting a brief overview of the traditional A* algorithm, before presenting our new MEA* algorithm. Finally, we present the StreetCut extension to MEA*, which improves the runtime speed of the algorithm.

Note that the random element in our pathfinding design — the points to which agents travel in intersections are not fixed — necessitates a runtime pathfinding algorithm. Routes between all possible source-destination street segments cannot be precomputed. This apparent complication is in fact an important feature of our scheme. The randomization of choices leading to identical or nearly identical cost values under the assumed optimization criteria is one of the paramount aspects of the art of simulation. From the viewpoint of a realistic model of human behaviour, this kind of nondeterminism clearly appears as something natural, owing to the inherent imperfection of human judgment, and cannot be realistically represented as a simple deterministic optimization problem. The most obvious and intuitively natural kind of fuzziness in the input to the pathfinding algorithm is the uncertainty about an agent's location within an intersection. A randomized representative, p_i, of that continuous space serves to perturb both distances and angles, allowing MEA* to explore a larger, more natural state space. The purpose of mirroring the point into the four quadrants of the intersection is to provide nontrivial yet realistic choices to the pathfinding algorithm.

A*, upon which MEA* is based, works by maintaining metadata for each point on the map to which an agent may move: the lowest acquired cost yet discovered for reaching that point, the previous point along the path that generated that lowest cost, and an estimate of the total cost that will be incurred should the path to the destination be completed from that point. Here, cost traditionally refers to Euclidean distance travelled. The estimate for the total cost of the final path through the point p, $\mathrm{F}(p) = \mathrm{A}(p) + \mathrm{E}(p)$, is computed as the acquired cost of the path up to point p, $\mathrm{A}(p)$, plus a heuristic estimate of the cost that will be acquired by completing the path from point p, $\mathrm{E}(p)$. The heuristic E must be admissible — that is, $\mathrm{E}(p)$ must be at most the cost that will be acquired in completing the path from p (e.g., the length of a straight line from p to the destination, if Euclidean distance is being used as the pathfinding metric). The A* algorithm then iteratively expands the search from the point with the lowest estimated final cost, $\mathrm{F}(p)$, until the destination point is expanded; the path from the source to the destination is then reconstructed by iterating

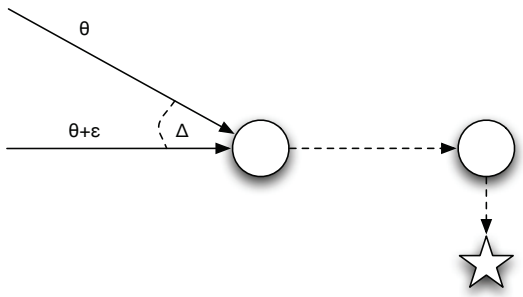


Figure 2.4: Two potential paths to the destination (the star) converging on a shared intermediate point (the left circle). The upper path has both a smaller acquired cost of $\theta < \theta + \epsilon$ and a smaller estimated final cost (assuming E estimates the angle change required to reach the destination by using a straight line to the destination). However, if $\epsilon < \Delta$, the path with acquired cost $\theta + \epsilon$ will produce the minimal-angle path to the destination.

along the chain of previous points stored at each point in the path. The final reconstructed path is optimal in the sense that its cost is minimized, conditioned on the agent being limited to moving to any of the (up to) four generated points within any intersection.

One feature of A* search is that it only maintains one set of metadata for each point on the map. If ever a newer path to point p is discovered that has a lower acquired cost $\mathrm{A}(p)$ than the existing path to point p, the older path is replaced by the new path's acquired cost, estimated total cost, and previous point metadata. When minimizing Euclidean distance travelled, if two paths h_1 and h_2 converge at p, but the acquired cost of h_1 is less than or equal to the acquired cost of h_2, then only h_1 need be expanded; it is guaranteed that h_2 will not yield a better final path to the destination if it is expanded. However, this guarantee fails for minimal-angle pathfinding, where the location of the previous point affects the cost of the outbound edge from a point. Consider the example in Figure 2.4, in which one inbound path to a shared intermediate point has both a smaller acquired cost and a smaller estimated final cost, yet the path generated by the other inbound path will yield the minimal final angle change.

This limitation of A* is addressed by the MEA* search algorithm. In MEA*, multiple inbound paths can co-exist at, and later be expanded from, a shared intermediate point. When possible, one inbound path can still obsolete another when it is guaranteed that one will not produce the optimal path to the destination.

Lemma 2.1. *In the minimal-angle variant of MEA*, if one inbound path h_1 can be reoriented (i.e., rotated) to another inbound path h_2, and the acquired cost of h_1 plus the cost of*

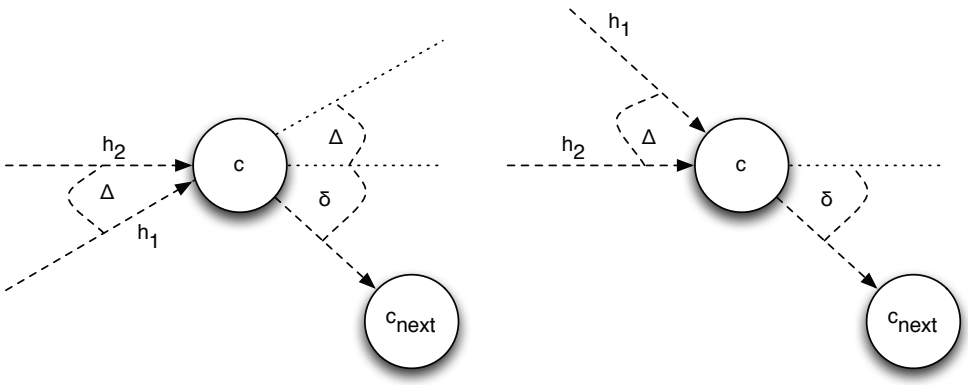


Figure 2.5: Two examples in which a path h_1 that can be reoriented to path h_2 at a cost of Δ, and h_2 can be reoriented to the next intermediate point at a cost of δ.

the reorientation is no more than the acquired cost of h_2, then it is guaranteed that h_2 will not form a better path to the destination than h_1.

Proof. Consider two inbound paths h_1 and h_2 to an intermediate point c, with acquired costs θ_1 and θ_2 respectively. Let Δ be the difference in the inbound angles between h_1 and h_2, and let δ be the cost of reorienting inbound path h_2 to face the next point, c_{next}. Two such situations are depicted in Figure 2.5.

Let θ_1' and θ_2' be the acquired cost at the next point from the expansion of h_1 and h_2 respectively. So, by the definition of δ, $\theta_2' = \theta_2 + \delta$. From the triangle inequality applied to angles, and as depicted in Figure 2.5, the cost of reorienting h_1 to face the next point is bounded within $[0, \Delta + \delta]$. Therefore, $\theta_1 \leq \theta_1' \leq \theta_1 + \Delta + \delta$.

From the lemma description, we assumed that $\theta_1 + \Delta \leq \theta_2$, meaning that $\theta_1' \leq \theta_2 + \delta = \theta_2'$. In other words, the new acquired cost of expanding h_1 to the next point will be no more than the new cost of expanding h_2. And, since both expansions will have an identical inbound angle at the next point (i.e., a line from c to c_{next}), h_2 is guaranteed not to produce a better path than h_1. □

Remark 2.2. The minimal-distance variant of MEA* degenerates into classic A*, saving at most one inbound path to each point. An inbound path h_1 to point c will never yield a better path to the destination than h_2 if its acquired cost is greater than or equal to that of h_2. The minimal-turn variant of MEA* has the same guarantee, provided that expanding an inbound path h at point c to any next point c_{next} necessitates making a turn.

Algorithm 2.3 The MEA* pathfinding algorithm

```
1: function FindPath(src, dst)
2:     queue, pts, cache ← {}, {}, {}
3:     for each street s containing point src do
4:         nc ← Crumb(src, s, NULL, dst)
5:         AddToCache(nc, src, s, cache)
6:         Add nc to queue
7:     while True do
8:         cur ← PollMinimalLB(queue)
9:         if cur.obsolete = TRUE then
10:            continue
11:        else if cur.pLoc = dst then
12:            return ReconstructPath(cur)
13:        else if cur.street contains point dst then
14:            Add Crumb(dst, cur.street, cur, dst) to queue
15:        else
16:            for each i ∈ GetIntersections(cur, dst) do
17:                Expand(cur, i, dst, queue, pts, cache)

18: procedure Expand(cur, i, dst, queue, pts, cache)
19:     nextStr ← i.intersectingStreet
20:     for each p ∈ GetRandomPoints(i, pts) do
21:         nc ← Crumb(p, nextStr, cur, dst)
22:         for each old ∈ GetCache(p, nextStr, cache) do
23:             obs ← CheckObsolete(old, nc)
24:             if obs = ObsoleteNew then
25:                 nc.obsolete ← True
26:                 break
27:             else if obs = ObsoleteOld then
28:                 old.obsolete ← True
29:                 RemFromCache(old, p, nextStr, cache)
30:         if nc.obsolete = False then
31:             Add nc to queue
32:             AddToCache(nc, p, nextStr, cache)
```

Sub-algorithm 2.3.1 Shared helper functions in the MEA* pathfinding algorithm

```
1: function GetRandomPoints(i, pts)
2:     if ∄ {intr, p1, p2, p3, p4} ∈ pts with intr = i then
3:         p ← a random point in i
4:         p1, p2, p3, p4 ← p mirrored into each quadrant of i
5:         Add {i, p1, p2, p3, p4} to pts
6:     return {p1, p2, p3, p4} where {i, p1, p2, p3, p4} ∈ pts

7: procedure AddToCache(crumb, point, street, cache)
8:     Add {point, crumb} to cache

9: procedure GetCache(point, street, cache)
10:    return {c | {point, c} ∈ cache}

11: procedure RemFromCache(crumb, point, street, cache)
12:    Remove {point, crumb} from cache
```

Sub-algorithm 2.3.2 Minimal-angle versions of MEA* helper functions

```
1: function Crumb(physicalLoc, street, prevCrumb, dst)
2:     c ← a new, empty crumb data structure
3:     c.pLoc ← physicalLoc
4:     c.street ← street
5:     c.prev ← prevCrumb
6:     c.obsolete ← False
7:     if c.prev = null then
8:         c.in ← null
9:     else if c.pLoc = c.prev.pLoc then
10:        c.in ← c.prev.in
11:    else
12:        c.in ← vector from c.prev.pLoc to c.pLoc
13:    c.angle ← accumulated angle change thus far
14:    c.dist ← accumulated distance thus far
15:    c.angleLB ← total angle change if going straight to dst
16:    c.distLB ← total distance if going straight to dst
17:    return c

18: function CheckObsolete(old, nc)
19:    if old.in = null then
20:        return ObsoleteNew
21:    else if nc.in = null then
22:        return ObsoleteOld
23:    Δ ← the angle between old.in and nc.in
24:    if (old.angle + Δ < nc.angle) or
25:       (old.angle + Δ = nc.angle and old.dist ≤ nc.dist) then
26:        return ObsoleteNew
27:    else if (nc.angle + Δ < old.angle) or
28:            (nc.angle + Δ = old.angle and nc.dist ≤ old.dist) then
29:        return ObsoleteOld
30:    return ObsoleteNone
```

The core of the MEA* algorithm is formalized in pseudocode in Algorithms 2.3 and 2.3.1. Each time DBS3 requires a path from a point *src* to a point *dst*, it calls FindPath(*src*, *dst*). Because the initial location could be on multiple streets (i.e., in an intersection), a crumb is added to the queue for each possible starting street. The crumb is the basic data structure of the search algorithm, containing not only information like the physical location of the search expansion it represents, but also the acquired and estimated costs of that expansion. For a search using Euclidean distance as the cost metric, the acquired cost would be the Euclidean distance travelled from *src* to the physical location of the crumb, and the estimated cost would be the acquired cost plus the cost estimated by an admissible heuristic to complete the path to *dst*. For a search using total angle change as the metric, the acquired cost would be the change in angle so far as well as the acquired Euclidean distance (for breaking ties, in case two paths yield identical angle change) and the estimated cost would again be the acquired cost plus an admissible heuristic. The data stored by the minimal-angle crumb is illustrated in Algorithm 2.3.2. The heuristic used by DBS3 for each of the three search metrics is as follows:

- For a minimal-distance search, E is the length of a straight line from the crumb to the destination;
- For a minimal-turn search, E is firstly the number of turns required to get from the street on which the crumb is located to the street on which the destination is located (the minimal number of turns between all pairs of streets is precomputed by DBS3). Ties are broken using the Euclidean length of straight lines from the crumbs to the destination (though ties could have been broken instead, e.g., by the angle change incurred to face directly to the destination).
- For a minimal-angle search, E is firstly the angle change that would be incurred in turning from the crumb's inbound path to face directly to the destination (turning clockwise or counterclockwise, whichever produces a smaller change in angle). Ties are broken using the Euclidean length of straight lines from the crumbs to the destination.

Remark 2.3. Each of the heuristics detailed above is admissible for its respective search metric. The minimal-distance heuristic is identical to the one used in the original A* algorithm [23]. For the minimal-turn search, there is clearly no path possible to the destination

that can be constructed with fewer turns than the minimal number of turns between a crumb's street and the destination street (and no tie-breaking Euclidean distance shorter than a straight line from the crumb to the destination). Similarly, for the minimal-angle search — subject to the assumption that agents can only travel in the direction they are facing, which is true in DBS3 — there is no path that could possibly incur less total angle change than turning (clockwise or counterclockwise, whichever is shorter) and proceeding in a straight line to the destination.

Similar to A*, MEA* uses a priority queue structure to store all of the crumbs from which it should expand the search. The PollMinimalLB(*queue*) function removes from the queue and returns the crumb with the lowest estimated total cost. For the minimal-angle search, for example, that would be the crumb with the lowest *angleLB* value (breaking ties with the lowest *distLB* value if necessary). If the crumb with the smallest lower bound on the cost of completing the path is at the destination, the path is reconstructed by tracing back from that crumb to the source (the proof of correctness of this behaviour is identical to that for A*). Otherwise, the crumb is expanded to the four random points chosen for the relevant intersections on that crumb's street, or directly to the destination (if the crumb is located on a street that contains the destination point *dst*). Which intersections are relevant depends on the search metric. While it is not incorrect to expand the search to every intersection on the street, a simple optimization is not to expand the search back to the previous street in the expansion; for minimal-turn pathfinding, DBS3 optimizes further by using the precomputed minimal number of turns between all streets, expanding crumbs only to those intersections whose cross-street is one turn fewer away from the destination than the current street.

Not every crumb placed at a given location need be expanded, though. One crumb placed at a given point may obsolete another crumb located at the same point. In the case of minimal-distance and minimal-turn pathfinding, one crumb will always obsolete another at the same point, yielding a classic A* search in the minimal-distance case, and something very similar to classic A* but with a different distance metric in the minimal-turn case. E.g., with minimal-distance pathfinding: the old crumb already located at that point will obsolete the new crumb if the old one's acquired Euclidean distance is less than or equal to that of the new one; the new crumb will obsolete the old crumb if the new one has a smaller acquired cost. That the new crumb is obsoleted in the case of ties is necessary for

correctness — otherwise, the search algorithm could enter an infinite loop at, e.g., a three-way intersection, where the search algorithm would keep making new, identical crumbs as it repeatedly expands over the same set of intersections.

For minimal-angle pathfinding, however, it is not guaranteed that one of two crumbs at a given location will obsolete the other. Recall from Figure 2.4 that a crumb with a larger acquired cost at a given point may form a better path than a crumb with a smaller acquired cost, if the inbound vectors of the two crumbs are different. However, if the acquired cost of one crumb is much larger, it can still be obsoleted.

Remark 2.4. If Δ is the angle difference between the two inbound vectors of two crumbs c_1 and c_2 at the same point, and if $a_1 + \Delta < a_2$ where a_i is the acquired cost of crumb c_i, then c_1 can obsolete c_2 (essentially, c_1 could be reoriented to c_2 and still have a lower acquired cost).

The minimal-angle obsolescence code is formalized in Algorithm 2.3.2. Regarding one crumb obsoleting another, it is also worth noting the following from a performance perspective.

Lemma 2.5. *If a new crumb obsoletes an old crumb during a vanilla MEA* search (i.e., not using the StreetCut improvement to MEA* that is presented shortly), using any of the minimal-distance, minimal-turn, or minimal-angle distance metrics, it is guaranteed that the old crumb has not yet been polled from the queue.*

Proof. For simplicity, we limit this proof to the minimal-distance pathfinder and its admissible heuristic, though the proof is similar for the other two pathfinding heuristics. Let c_{old} be a crumb at point p_1 with acquired Euclidean distance a_{old} and an estimated total distance of $a_{old} + d_1$, where d_1 is the straight-line distance from p_1 to the destination. Let c_{exp} be a crumb at a different location p_2 that is being expanded to produce a new crumb c_{new} at point p_1. Assume that c_{new} obsoletes c_{old}, i.e., $a_{new} < a_{old}$. To show that the old crumb has not yet been polled off the queue, we show that the estimated cost of c_{exp} must be less than the estimated cost of c_{old} (and therefore c_{exp} would be expanded first). Let λ be the distance between p_1 and p_2. Then,

$$\begin{aligned} & a_{new} = a_{exp} + \lambda < a_{old} \\ \Rightarrow\ & a_{exp} + \lambda + d_1 < a_{old} + d_1 \ . \end{aligned}$$

Pathfinder	*StreetCut Disabled (ms)*	*StreetCut Enabled (ms)*	Runtime Reduction
Minimize Angle	255751	116392	54.49%
Minimize Distance	46195	30569	33.83%
Minimize Turns	3957	3689	6.77%

Table 2.1: The running time of a day-long DBS3 simulation of 1000 agents on the Fira map with $\alpha = \delta = 1$ for each of the three pathfinding metrics, with StreetCut enabled or disabled.

By the triangle inequality, $d_2 \leq d_1 + \lambda$, so

$$a_{exp} + d_2 \leq a_{exp} + \lambda + d_1 < a_{old} + d_1 \ .$$

Therefore, c_{exp} would be expanded prior to c_{old}. □

The ability for one crumb to obsolete another is beneficial, because expanding crumbs unnecessarily degrades performance by filling the priority queue with ever more crumbs to expand. It is beneficial to obsolete as many crumbs as possible, if it can be shown that they are guaranteed not to yield the optimal path to the destination. The basic approach of allowing one crumb to obsolete another at the same point can be extended to allow one crumb to obsolete any of the other crumbs on the same street. We refer to this extension as StreetCut. The important observation underlying this optimization is that if two crumbs are on the same street, they have the same set of possible intersections to which they can expand. We extend the reorientation concept described for the vanilla MEA* minimal-angle obsolescences to allow for translation and reorientation. For example, with the minimal-distance pathfinder, if two crumbs c_1 and c_2 are on the same street and λ is the distance between them, and if $a_1 + \lambda \leq a_2$, then c_1 can obsolete c_2 because c_1 could be translated over to c_2 and still have no more acquired Euclidean distance. The combination of reorientation and translation used for minimal-angle StreetCut is formalized in Algorithm 2.3.3, replacing the listed functions from Algorithm 2.3.1 and Algorithm 2.3.2.

To demonstrate the effectiveness of StreetCut, we simulated 1000 agents on the Fira map moving for one day according to the destination-selection algorithm with $\alpha = \delta = 1$ and each of the three pathfinding metrics. For each pathfinder, we ran the simulation with StreetCut enabled and with StreetCut disabled. The running times, as measured on a 2.93 GHz Core i7 with 8 GB of RAM running OS X 10.6.8, are summarized in Table 2.1.

Sub-algorithm 2.3.3 The StreetCut improvement to minimal-angle MEA*

```
1: procedure AddToCache(crumb, point, street, cache)
2:     Add {street, crumb} to cache

3: procedure GetCache(point, street, cache)
4:     return {c | {street, c} ∈ cache}

5: procedure RemFromCache(crumb, point, street, cache)
6:     Remove {street, crumb} from cache

7: function CheckObsolete(old, nc)
8:     if old.in = null then
9:         return ObsoleteNew
10:    else if nc.in = null then
11:        return ObsoleteOld
12:    Δ ← the angle between old.in and nc.in
13:    λ ← Euclidean distance from old.pLoc to nc.pLoc
14:    if (old.angle + Δ < nc.angle) or
15:       (old.angle + Δ = nc.angle and old.dist + λ ≤ nc.dist) then
16:        return ObsoleteNew
17:    else if (nc.angle + Δ < old.angle) or
18:            (nc.angle + Δ = old.angle and nc.dist + λ ≤ old.dist) then
19:        return ObsoleteOld
20:    return ObsoleteNone
```

The runtime for the minimal-turn pathfinder was the fastest because DBS3 precomputes the number of turns between all street pairs (by running a breadth-first search from each street prior to simulation initialization), allowing it to expand crumbs only to locations that are one turn closer to the final destination. The runtime for the minimal angle pathfinder was the largest because its admissible heuristic was the least accurate of the three, in terms of predicting the final cost of the crumb's path. Determining a more accurate admissible heuristic for minimal-angle searches is one direction of future work. The important result from this experiment, however, is that in all three cases, enabling StreetCut reduced the running time of the simulation. In the case of the minimal-angle pathfinder, the running time was cut by more than half.

Interestingly, the guarantee in Lemma 2.5 generalizes to minimal-distance and minimal-turn StreetCut pathfinding, but it does not hold with minimal-angle StreetCut pathfinding. In minimal-angle StreetCut pathfinding, a crumb can be obsoleted after it has been polled from the priority queue. While not an issue of correctness, it is a negative for performance, since an obsolete crumb is guaranteed not to produce the optimal path. However, in practice, such post-polling obsolescences are rare and therefore not a concern. In the day-long

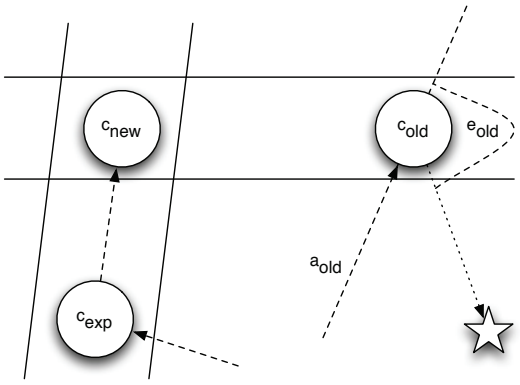


Figure 2.6: The new crumb, c_{new}, is on the same street as the old crumb, c_{old}, but in a different location on the street. The star is the destination, and the angle change incurred by walking in a straight line to the destination is the admissible heuristic, illustrated for c_{old} as e_{old}.

minimal-angle StreetCut simulation on the Fira map, approximately 0.03% of all crumbs that were rendered obsolete had already been polled from the queue. In an identical simulation on the Edmonton map, approximately 0.24% of all obsolescences were to crumbs polled from the queue. In other words, while post-polling obsolescences are possible in minimal-angle StreetCut pathfinding, they are sufficiently rare as not to affect performance meaningfully.

To understand why post-polling obsolescences are possible, yet rare in practice, when minimal-angle StreetCut pathfinding is used, consider the following. Let c_{old} be a crumb at point p_{old} with acquired angle change a_{old} and an estimated final angle change of $a_{old} + e_{old}$, where e_{old} is angle difference between the inbound vector into c_{old} and a vector straight from p_{old} to the destination. Let c_{exp} be a crumb on a different street that is currently being expanded, and let c_{new} be one of the new crumbs that results from the expansion of c_{exp}. Assume that c_{new} is on the same street as c_{old}. This scenario is illustrated in Figure 2.6.

What we wish to determine is under what scenario c_{new} could obsolete c_{old}, given that c_{old} has already been expanded from the priority queue (i.e., this obsolescence is a post-polling obsolescence). Given that c_{old} was expanded before c_{exp}, we know that

$$a_{old} + e_{old} < a_{exp} + e_{exp} \tag{2.4}$$

(technically, the two values could be equal and the tie could have been broken by estimated final Euclidean distance; however, despite being a mathematical possibility, it does not occur

in practice, so for simplicity we ignore that case here). Because c_{new} obsoletes c_{old}, we also know that

$$a_{new} + \Delta_{new,old} < a_{old} \ , \tag{2.5}$$

where $\Delta_{new,old}$ denotes the difference in angle between the inbound vector to c_{new} and the inbound vector to c_{old}.

We can rewrite Equation 2.5 by noting that $a_{new} = a_{exp} + \Delta_{exp,new}$, yielding

$$a_{exp} + \Delta_{exp,new} + \Delta_{new,old} < a_{old} \ . \tag{2.6}$$

Then, we rearrange Equation 2.4 to determine that

$$a_{old} < a_{exp} + e_{exp} - e_{old} \ . \tag{2.7}$$

By combining Equations 2.6 and 2.7, we determine that

$$a_{exp} + \Delta_{exp,new} + \Delta_{new,old} < a_{exp} + e_{exp} - e_{old} \ . \tag{2.8}$$

In other words, post-polling obsolescence can occur only if

$$\Delta_{exp,new} + \Delta_{new,old} + e_{old} < e_{exp} \ . \tag{2.9}$$

While a post-polling obsolescence will occur if and only if Equations 2.4 and 2.5 are true, the necessary (but not sufficient) condition illustrated in Equation 2.9 makes it clearer why post-polling obsolescence is rare in practice: it is necessary for the sum angle change of three reorientations to be less than the angle change of a single reorientation.

2.3.4 Simulation Initialization

One challenge in creating a meaningful simulation is initializing it in steady state. As a simple example, we would not want to start DBS3's mobility simulation every time with all agents standing in the same spot at the centre of the map, because this is not an expected state in which to observe the simulation as $t \to \infty$, where t is the running time of the simulation. At any point in time, we would expect to see some agents moving and some

agents paused, scattered around the map. More formally, let S be any state in which a simulation could be, and let $\mathrm{P}(S)$ be the probability of observing the simulation in that state at time t as $t \to \infty$. Initializing a simulation in steady state is equivalent to choosing a starting state S with probability $\mathrm{P}(S)$.

An unintuitive observation is that those agents that are moving are expected to be travelling between two destinations that are further away than the average distance between destinations on the map [13]. To motivate this point, consider the following example based on Olofsson [45]. Assume an agent travels at a constant speed of 1 m/s with no pause times. Also assume that each time that agent chooses a new destination, it has a 50% chance of choosing a destination 60 m away, and a 50% chance of choosing a destination 120 m away (so the average distance of any trip is 90 m). Note that any given trip has a 50% chance of taking one minute and a 50% chance of taking two minutes. The unintuitive detail is that, if half of the trips are one-minute trips and the other half are two-minute trips, then 1/3 of the agent's *time* is occupied performing one-minute trips and 2/3 of the agent's time is spent doing two-minute trips (because two-minute trips take twice as long to perform). Thus, if a third party observes the agent at some random time t, the observer has a 1/3 chance of seeing the agent performing a 60 m trip, and a 2/3 chance of seeing the agent performing a 120 m trip. So, the expected trip length that the agent is performing in steady state is

$$\frac{1}{3} \cdot 60 \text{ m} + \frac{2}{3} \cdot 120 \text{ m} = 100 \text{ m} > 90 \text{ m}.$$

A common approach to solving the initialization problem is to place agents in some very rough but easily computable approximation of steady state, then run the simulation for some fixed amount of burn-off time to let it converge to steady state before drawing any mobility data from the simulation [41, 6]. This is the initialization approach used by DBS3.

To place agents in a rough semblance of steady state, DBS3 begins by computing the equilibrium distribution of the street-transition Markov chain defined by the destination-selection algorithm presented in Section 2.3.2. Recall that agents on street s choose their next destination street with probability proportional to $\mathrm{P}(s, d)$ over all destination streets d, where P was defined in Equation 2.1. This street-transition algorithm defines a Markov

chain

$$M = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix},$$

where

$$p_{i,j} = \frac{\mathrm{P}(i,j)}{\sum_d \mathrm{P}(i,d)}$$

and n is the number of streets. In other words, $p_{i,j}$ is the probability that an agent will choose a new destination on street j after pausing at a destination on street i.

Because $p_{i,j} > 0$ for all streets i and j (per the assumptions about street length and connectivity made in Section 2.3.2), M is trivially guaranteed to be ergodic. Therefore, M^x converges as $x \to \infty$ to a matrix with identical rows $\pi = [\pi_1\ \pi_2\ \cdots\ \pi_n]$. Each entry π_i in the equilibrium distribution vector π represents the probability that street i contains the x^{th} destination chosen by an agent as $x \to \infty$.

DBS3 places each agent at a destination according to the equilibrium distribution π. That is, for each agent A_i, a street s is chosen according to the distribution π, then a uniform random location is chosen on s at which to place that agent. After being placed in an initial location on street s, agent A_i chooses a new destination street d proportionally to the value of $\mathrm{P}(s,d)$ over all possible streets d, and uniformly chooses a destination on street d. The agent immediately departs the initial location for the destination on street d.

Beginning from this rough but easily computable approximation of the steady state, DBS3 allows each agent to move according to the pathfinding, destination-selection, and pause parameters of the simulation for 86400 seconds (one day) prior to the beginning of the simulation, as observed by a person or program using DBS3 for mobility data. That is, the one-day burn-off period is performed transparently by DBS3. Assuming one day is enough time on the user's map for agent movement to approach steady state, DBS3 simulations begin in steady state from the point of view of the user. Future work will investigate how alternative proposals for initializing random trip mobility models in steady state [41] could be adapted to work in DBS3.

2.4 Mobility Parameters and Wireless Networking Simulations

In the previous section, we discussed the three choices that parameterize a DBS3 simulation: centrality bias, distance decay, and the pathfinding metric. But what effect do those parameters have on wireless network simulations? To study this problem, we built three client programs that abstractly represent three different types of network protocols: information spread, information collection, and token passing. In information spread, one agent begins with information that has to be disseminated to all of the agents in the simulation, either directly or via other agents. In information collection, a single collector agent has to collect information directly from every other agent in the simulation. Finally, in token passing, a single agent at a time can possess the token, and will pass it when possible to any other agent that has not yet had the token, until each agent has possessed the token once. We adjust the centrality bias and distance decay exponents, α and δ, as well as the pathfinding algorithm used by DBS3, to determine what effect these parameters have on the speed at which information is spread, at which information is collected, and at which each agent receives the token.

Because we want to study the effect of mobility parameters on wireless networking results in general, rather than the effect on any particular protocol implementation, we are not concerned with simulating underlying MAC protocols or interference models. Rather, we want to simulate these styles of network operations at a higher, abstract level, to identify more easily the effects of the mobility parameters themselves. So, our client applications function according to simple rules. The information spread simulation works as follows:

1. Of the m agents, initially one is coloured red (representing that it has the information) and the others are coloured black (representing that they want the information);
2. If a red agent comes within r metres of a black agent, the black agent turns red;
3. Continue until all agents are coloured red.

The information collection simulation follows these rules:

1. Of the m agents, initially one is coloured red (the collector agent) and the others are coloured black (meaning that they have information to be collected);

2. If the red agent comes within r metres of a black agent, the black agent turns blue (representing that its information has been collected);

3. Continue until all agents are coloured red or blue.

Finally, the token-passing simulation performs the following steps:

1. Of the m agents, initially one is coloured red (meaning that it possesses the token) and the others are coloured black (representing that they want the token);

2. If the red agent comes within r metres of a black agent, the red agent turns blue (representing that it has previously had the token) and the black agent turns red;

3. Continue until all agents are coloured red or blue.

For each simulation, we arbitrarily fixed the number of agents at $m = 25$ and the transmission range at $r = 30$ metres. Then, we recorded what proportion of agents were coloured red or blue, i.e., not coloured black, over time. The effects that the mobility parameters had on the network simulations varied between the two maps, illustrating the different behaviour of the mobility model in a regular, grid-like environment and its behaviour in a more graph-theoretically interesting, organic environment.

As illustrated in Figure 2.7, changing the centrality bias exponent within the range $\alpha = 0$ to $\alpha = 3$ had no significant impact on information spread or information collection in Edmonton.[2.e] Because the map of downtown Edmonton is predominantly grid structured, most streets are one or two turns away from most other streets. In fact, the diameter of the map (i.e., the largest number of turns required to move between any pair of streets in a minimum number of turns) is only four. As such, most locations are just as well integrated as most other locations, meaning that centrality bias has negligible effect on agent movement.

In Fira, on the other hand, centrality bias had a large effect on the results of the network simulations. Increasing α from zero to three decreased the amount of time the network protocols took to complete, illustrated in Figure 2.8, as agents interacted more frequently in the central areas of the 36-turn diameter map. Whereas it took 12.27 minutes on average for 80% of the agents to become red in the information spread simulations when $\alpha = 0$, it took just over three-quarters of that time, 9.40 minutes, when $\alpha = 3$. In the information

[2.e] For clarity of exposition, the token passing graphs have been omitted, since they were consistently similar to the information collection graphs.

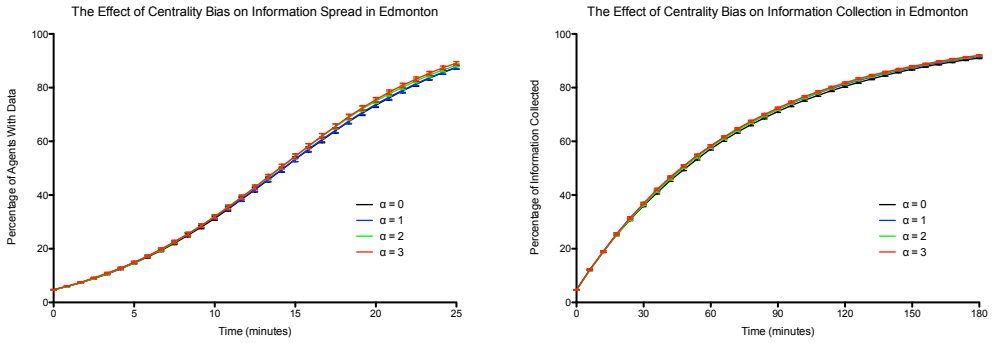


Figure 2.7: The effect of the centrality bias exponent, α, on the rates of information spread and information collection in Edmonton. The distance decay exponent, δ, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

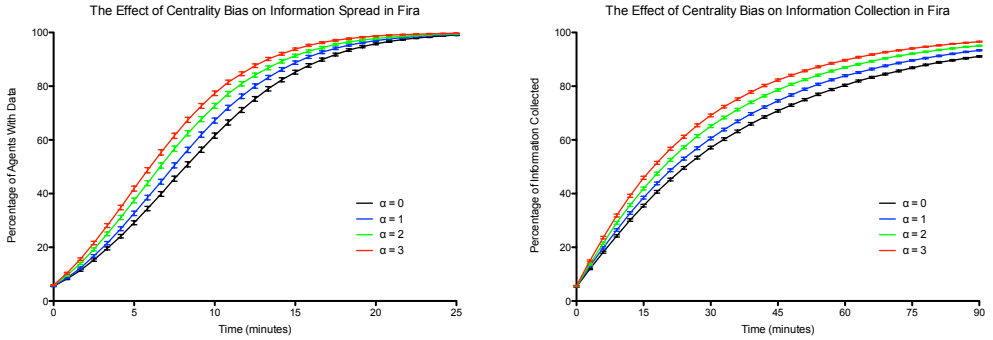


Figure 2.8: The effect of the centrality bias exponent, α, on the rates of information spread and information collection in Fira. The distance decay exponent, δ, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

collection and token passing experiments, 80% of the agents were coloured blue or red in approximately 69.97% of the time and 72.59% of the time respectively, when $\alpha = 3$ as opposed to $\alpha = 0$.

Unlike the effects of increasing centrality bias, the effects of increasing distance decay were not apparent until a large exponent was used. As shown in Figures 2.9 and 2.10, increasing δ from zero to two in the information spreading experiment — thereby increasing the expected amount of time until distant agents would come together — only increased the amount of time until 80% of the agents had the information by 6.09% in Edmonton and by 6.66% in Fira. The effect was even less pronounced in the information collection experiment. However, when δ was increased to three, distance decay began to have a larger effect on the abstracted network protocols. Information took approximately 11.70% longer to reach 80% of the agents in Edmonton; and, on the larger-diameter Fira map where there are more turns

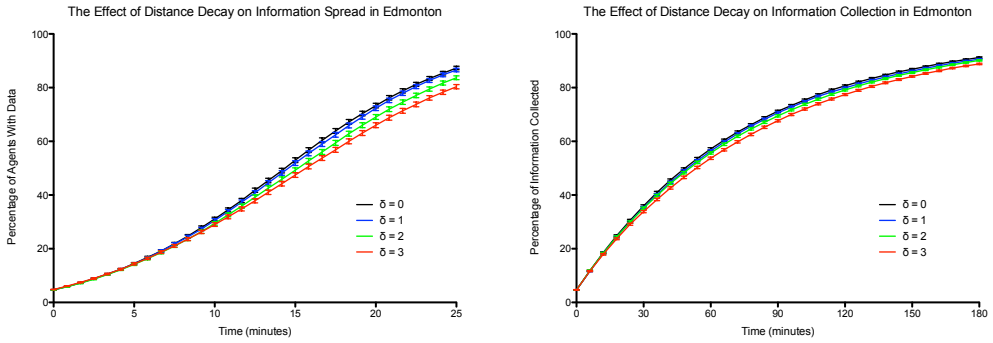


Figure 2.9: The effect of the distance decay exponent, δ, on the rates of information spread and information collection in Edmonton. The centrality bias exponent, α, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

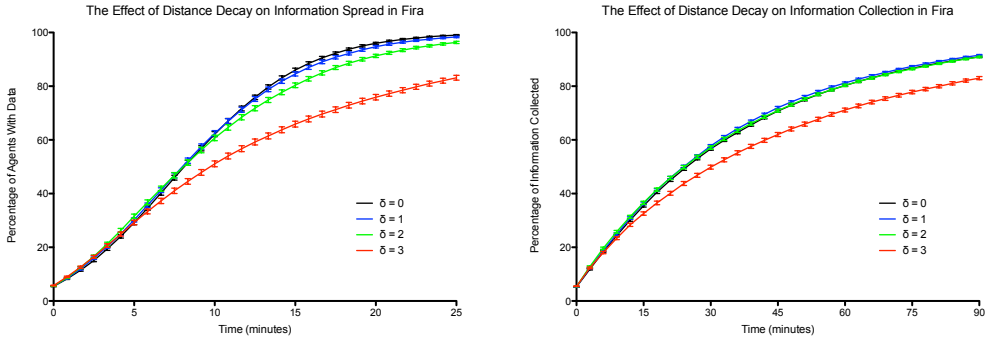


Figure 2.10: The effect of the distance decay exponent, δ, on the rates of information spread and information collection in Fira. The centrality bias exponent, α, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

to be made between distant streets, information spread took approximately 52.56% longer compared to when $\delta = 0$. So while large distance decay exponents do affect the network simulation results, particularly on large-diameter (i.e., less grid-based) maps, reasonable exponent values in the range $0 \leq \delta \leq 2$ have minimal effect on network simulations.

The final mobility model parameter that we tested was the pathfinding algorithm. We ran the information spread, information collection, and token passing experiments with all three pathfinders: the pathfinder that minimizes the number of turns, the pathfinder that minimizes the magnitude of the change in angle, and the classical pathfinder that minimizes Euclidean distance travelled. The effects of the pathfinder choice were most pronounced on the grid-like map of downtown Edmonton. In that environment, a pathfinder that minimizes Euclidean distance travelled will make many additional turns, imparting a diagonal component to the movement of an agent travelling across the map. This convergence of

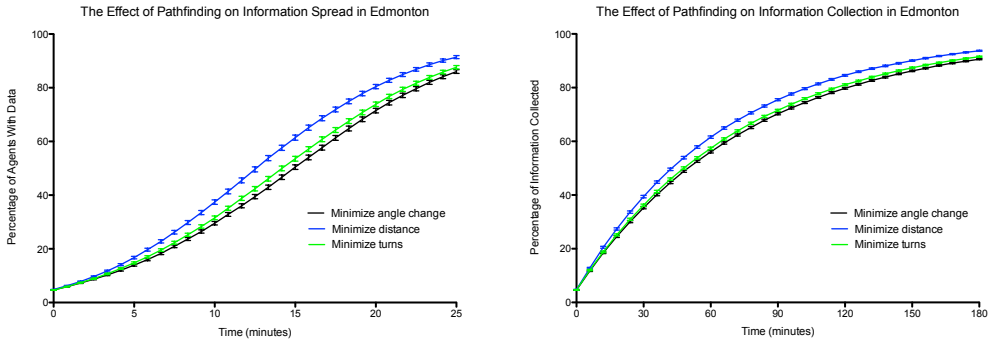


Figure 2.11: The effect of the pathfinding algorithm on the rates of information spread and information collection in Edmonton. The exponents α and δ were fixed at zero. Error bars represents 95% confidence intervals around the mean over 2500 trials.

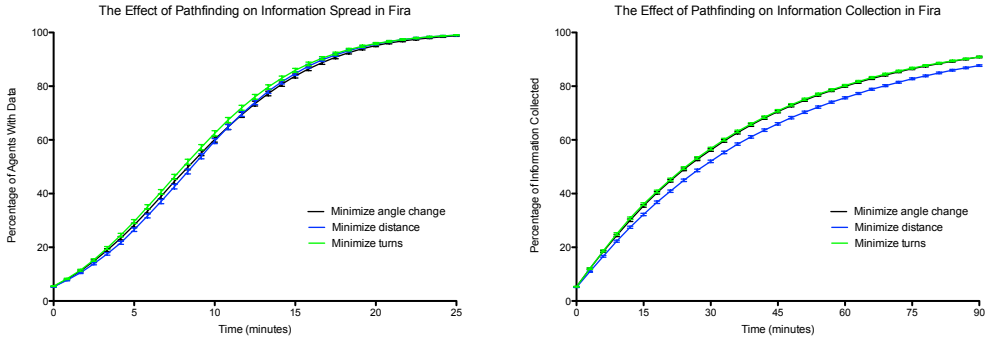


Figure 2.12: The effect of the pathfinding algorithm on the rates of information spread and information collection in Fira. The exponents α and δ were fixed at zero. Error bars represents 95% confidence intervals around the mean over 2500 trials.

agents towards the central part of the grid speeds the rate of information dissemination and collection. As illustrated in Figure 2.11, the minimal-distance pathfinder took approximately 85.99% as long as the angle-minimizing pathfinder to achieve 80% information spread in Edmonton. Interestingly, the opposite effect was observed on the Fira map, where the angle-minimizing and turn-minimizing pathfinders caused agents to prefer the major thoroughfares to the side streets, leading to more agent interaction along those thoroughfares. Information collection, as seen in Figure 2.12, was slower with the pathfinder that minimized Euclidean distance travelled — the turn-minimizing pathfinder achieved 80% information collection in 86.43% of the time taken by the minimal-distance pathfinder. However, the overall effect of the pathfinder choice over all three experiments (information spread, information collection, and token passing) was far greater on the grid-like map of Edmonton, on which there are more potential paths between any two locations.

Note that, regardless of which experiment was performed on which map, the angle-minimizing pathfinder and the turn-minimizing pathfinder produced similar results, whereas the minimal-distance pathfinder was often the odd one out. We postulate that the two more realistic pathfinders were producing movement patterns in which agents frequented many of the same portions of the map, unlike the minimal-distance pathfinder. To test this theory, we began by segmenting the maps on which agents are moving in DBS3. On each street, a segmentation point is placed at either end of the street, as well as at any location where that street is intersected by another street. Recall from Section 2.3.1 that a street in DBS3 does not necessarily correspond to a whole roadway in a city; a named city street could be represented as multiple streets in DBS3 if the roadway curves. Hence, a real city street could be broken into multiple streets in DBS3's representation, and further divided into segments: those portions of the street that fall between two consecutive segmentation points.

We then determined the proportion of pedestrian traffic on each segment of the map in DBS3's steady state. To compute those proportions, we initialized a single agent at a destination chosen according to the equilibrium distribution, π, of the street-transition Markov chain defined in Section 2.3.4. That agent then chose a new destination according to the destination-selection algorithm and travelled to it at a constant speed according to the pathfinding algorithm, where it then chose a new destination. After one million destinations had been reached, the proportion of time the agent spent on each segment of the map was computed. We eliminated pause times from DBS3 during this steady-state distribution computation, since random pauses would skew the distribution of traversed streets towards popular destinations (similar to how Hillier and Iida [27] computed pedestrian density in London by observing pedestrian flow, rather than pedestrian locations).

Using the minimal-turns pathfinder as our baseline, we recorded which ten segments saw the greatest proportion of agent-time in steady state. We then recorded what proportion of agent-time those same ten segments received when the other two pathfinders were used (note that those ten segments were not necessarily the top ten segments used by the other two pathfinders). The results of this experiment for both the Edmonton map and the Fira map are shown in Figure 2.13. Particularly on the grid-based Edmonton map, on which there are many more potential paths between any two locations and the angle formed by most intersections is approximately 90 degrees, the turn-minimizing and angle-minimizing pathfinders produced nearly identical amounts of use for the ten chosen segments, contrasted

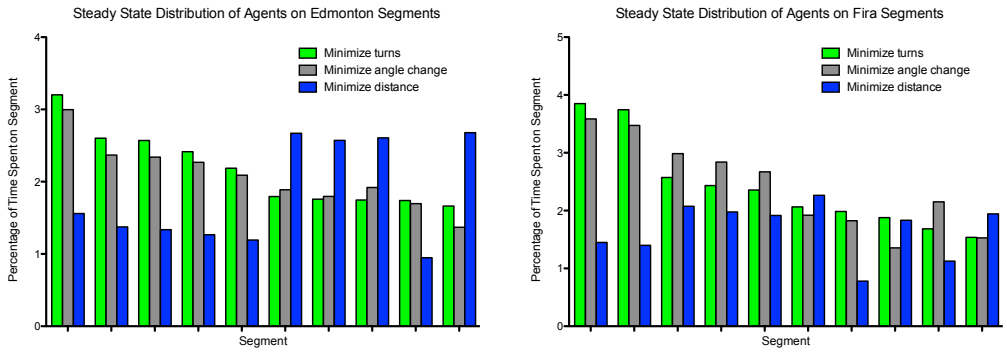


Figure 2.13: The effect of the pathfinding algorithm on the steady state distribution of agents across ten fixed segments in Edmonton and Fira. The ten segments chosen were those segments inhabited by the highest percentage of agents in steady state when the minimizing-turns pathfinder was used. The exponents α and δ were fixed at zero.

with significantly different usage patterns from the minimal-distance pathfinder. The results on the map of Fira were similar, though not as pronounced given the highly varied intersection angles and numerous side streets having only one path in and out. Based on this result, we can conclude that the minimal-angle and minimal-turn pathfinders will yield similar network simulation results in general, compared to the minimal-distance pathfinder.

Based on the results presented in this section, we can conclude that (a) centrality bias will have the greatest effect on simulation results on non-grid maps, in which the larger map diameter produces a more meaningful measure of which locations are centrally located; (b) while distance decay can affect simulation results, particularly in larger radius maps, its effect is minimal when the distance decay exponent is constrained to the reasonable range of $0 \leq \delta \leq 2$; and, (c) while the choice of pathfinding algorithm will have the greatest effect on simulation results on a grid-like map, using either of the more realistic pathfinders (minimal-angle or minimal-turn) will produce similar results, in contrast to the less realistic minimal-distance pathfinder.

2.5 Verification of DBS3

While we investigated, in the previous section, the effect of DBS3's parameters on various model network simulations, the question of what values a DBS3 user should choose remains. In this section, we will verify that when good values are chosen for those parameters, DBS3 will generate mobility data that is highly correlated to the mobility patterns of real people

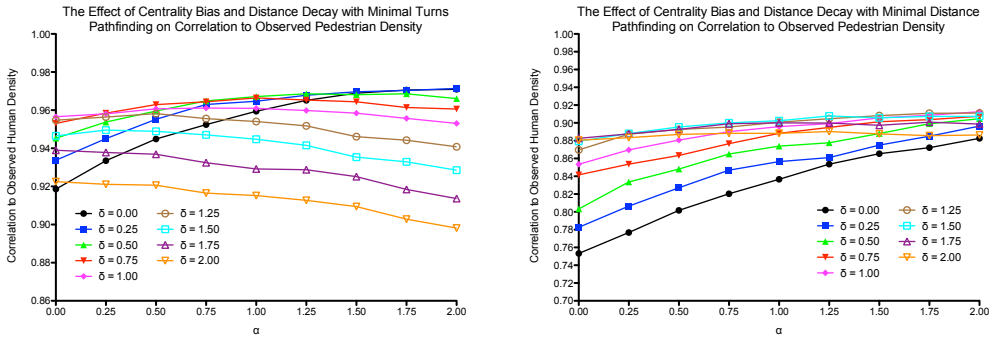


Figure 2.14: The effect of the centrality bias exponent, α, and the distance decay exponent, δ, on the correlation between DBS3's steady state distribution of pedestrians (using two different pathfinders) and the observed distribution of pedestrians in Edmonton.

in an urban environment.

We picked eight segments from the Edmonton map, chosen (a) to represent a wide range of pedestrian densities as predicted by DBS3; (b) to represent a diverse set of locations on the map; and, (c) to allow us to quickly travel between all the segments. We recorded the number of people we observed on those segments as we travelled through them during a Monday lunch hour in August 2011, creating a snapshot of the number of people on each of those segments over a short period of time.

We compared the steady-state distribution of pedestrians generated by DBS3 on those eight segments of the Edmonton map, using the three different pathfinding algorithms as we varied $0 \leq \alpha, \delta \leq 2$, to the observed distribution of pedestrians in Edmonton. This comparison is summarized in Figure 2.14, which shows the correlation between the generated and observed pedestrian densities for both the minimal-turn and minimal-distance pathfinders (the minimal-angle pathfinder produced similar results to the minimal-turn pathfinder and is omitted for clarity of exposition).

There are several key results that emerged from this study. Foremost, the pedestrian distribution generated by DBS3 is highly correlated to the observed distribution: using the minimal-turns pathfinder with $\alpha = \delta = 1$, the coefficient of determination was $R^2 = 0.96096$. In other words, DBS3 generates mobility distributions that are highly reflective of those of actual people.

Additionally, DBS3's mobility model is more accurate than the traditional random waypoint model with pathfinding that minimizes Euclidean distance. Setting $\alpha = \delta = 0$ elimi-

nates both centrality bias and distance decay, yielding a random waypoint model over the configured space of the map. In this case, the minimal-distance pathfinder produced a coefficient of determination of only $R^2 = 0.75324$. In other words, the novel destination-selection and pathfinding algorithms introduced in DBS3 produce far more accurate representations of human movement than traditional approaches.

It should be noted, based on the crisscrossing nature of the minimal-turns correlation graph in Figure 2.14, that optimizing the values of α and δ to best reflect human movement is a nontrivial problem. This result is not surprising, since centrality bias and distance decay are interrelated — one exponent draws agents into better connected areas, while the other dissuades agents from leaving those areas. High values for both α and δ lead to lower correlation with observed pedestrian distributions, as do low values for both exponents. We suggest that $\alpha = \delta = 1$ are good default values, with $0 \leq \alpha, \delta \leq 2$ being a realistic range. Optimizing the values of α and δ to best reflect human movement across a diverse set of configured spaces is an area of future study. Notably, our exercise shows a way to address this problem rigorously for any particular case of a city map: it can be formally defined as maximizing the coefficient of determination for a collection of static population snapshots taken in some representative locations in the city at approximately the same time.

We should also note that the northwest corner of the Edmonton map contains the campus of Grant MacEwan University. Because our correlation study was performed in August when classes were not in session, we did not include segments from the immediate area of the campus in the study. As a casual observation, there were far fewer people on those street segments in August than predicted by DBS3. DBS3 predicted (based on the nature of the configured space) that there would be something that would attract a large number of people in that area of the map. Indeed there is — a university campus. However, DBS3's model does not take into account the seasonal variation in pedestrian density associated with something like a university. This result demonstrates the tradeoff in using a mobility model like DBS3's, which does not annotate the input map: DBS3 does an excellent job of predicting, in aggregate over time, how pedestrian traffic will flow on a given map. However, it cannot predict such details as seasonal or temporal variation in traffic flow for a given location. Overall, because of how easy it is to parameterize a DBS3 simulation compared to a mobility model that requires an annotated map, and how well DBS3 predicted the pedestrian density across the rest of the map, we feel that this tradeoff is well worth it.

2.6 Using DBS3

DBS3 is freely available to researchers requiring mobility data for wireless network simulations or other uses. DBS3 delivers its mobility data to applications using a client-server model: client applications request mobility data as needed, and the DBS3 server replies with data on-demand. This client-server design allows DBS3 to integrate with arbitrary network simulators, e.g., ns-2 [4] and SMURPH/SIDE [15], or other non-network simulators that require mobility data, e.g., an airborne epidemic simulator. There are two different but similar client-server protocols used by DBS3: the University of Alberta Mobility Protocol (UAMP) and the Mobility Visualization Protocol (MVISP).

DBS3's UAMP server should be used when mobility data is needed for a large number of trials of a given experiment. UAMP clients send a simulation request to DBS3, including the number of agents they want simulated, the duration of movement data they need, and a random seed for the server to use (different seeds result in different mobility data). The UAMP server then sends the requested mobility data to the client. UAMP clients can prematurely terminate a simulation if they no longer require any more movement data, so clients that do not know *a priori* how much movement data they will require should request the maximum possible duration from DBS3. As a practical example, a flu simulator could run 100 trials of an experiment by sending 100 different seeds to the UAMP server, thus receiving 100 different sets of movement data.

The MVISP server in DBS3, on the other hand, should be used to visualize the results of a single simulation. MVISP clients receive a simulation specification from DBS3 upon connecting, including the number of agents that DBS3 will be simulating and the duration of the simulation. The MVISP client then receives this movement data and sends notifications back to the MVISP server when agents undergo state changes. For example, an MVISP flu-simulation client might send notifications to DBS3 when agents (i.e., people) change from the uninfected state to the incubating state. DBS3 includes a GUI with a built-in MVISP server, providing a view of state changes as they unfold in a single run of a mobility simulation. Figure 2.15 shows a screenshot of the GUI playing back a mobility simulation. Agent states in that screenshot were computed by an MVISP client that runs a simplistic simulation of a highly virulent airborne disease.

The overall architecture of DBS3 is illustrated in Figure 2.16, which shows how client

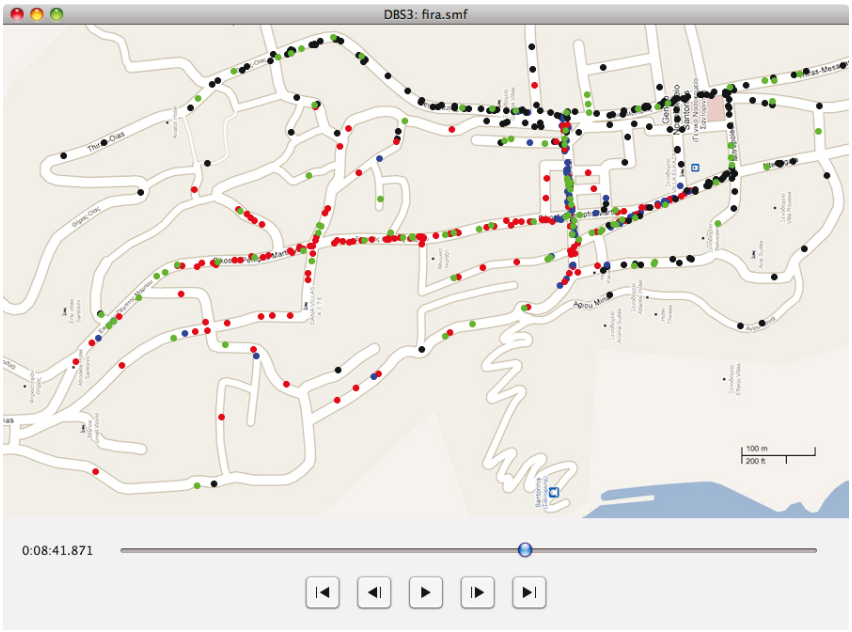


Figure 2.15: A simulation of a highly virulent airborne disease shown in DBS3's GUI, which is displaying agent states computed by an external MVISP client. Agents are shown in the uninfected (black), incubating (blue), contagious (red), or vaccinated (green) states as a virus with a one-minute incubation time and a two-metre infection range spreads among the mobile agents in Fira, Greece.

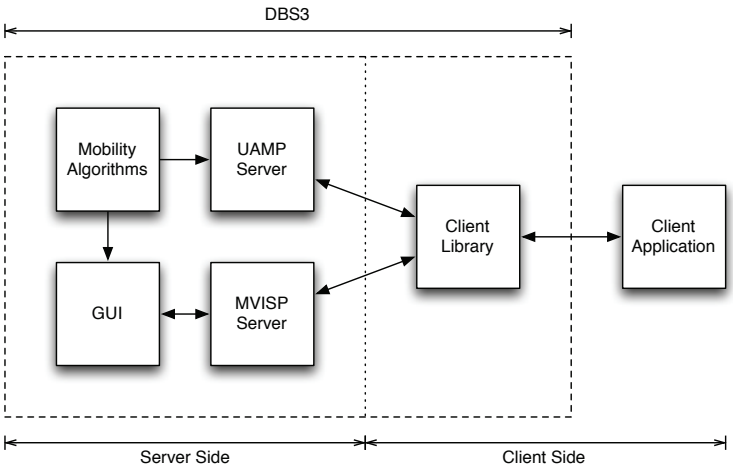


Figure 2.16: The overall architecture of DBS3. The server side is responsible for computing agent mobility and sending those results to clients on-request via a UAMP server. Alternately, individual simulations can be displayed in the GUI, optionally using the MVISP server to receive state changes from an external client. Clients that utilize mobility data sent by DBS3 can be written quickly using the provided UAMP / MVISP client library.

applications (e.g., wireless networking simulators, airborne pathogen simulators, or any other kind of simulator) receive mobility data from DBS3. As illustrated in that diagram, DBS3 includes a client library that allows users of DBS3 to quickly and easily create their own UAMP and/or MVISP client applications. The library is available in both C and Java. The C version is described below, but the Java version is highly similar.

Using the provided C library, clients can connect to a UAMP or MVISP server with the `uampConnect` or `mvispConnect` functions, respectively. The `uampConnect` function takes the number of agents to simulate, the duration of the simulation, and the random seed as parameters. The `mvispConnect` function, on the other hand, takes the number of states in which agents may exist as a parameter, but returns the number of agents and time limit received from the server. Once connected, the library behaves identically regardless of which connect function was used (with the exception that only MVISP clients send state change messages back to the server using the `uampChangeState` function), meaning that a single client application can function as both a UAMP and MVISP client depending on which connect function is called.

When UAMP and MVISP clients request mobility data from the server for a given agent, the server replies with the next location and point in time at which that agent changes speed or direction. Clients are therefore able to interpolate the location of each agent at all times. The client library presents the movement of an agent between two such interpolation points in a `uampCommand` structure that contains the time and location of the agent at the first point and at the second point. This structure is retrieved from the library using the `uampCurrentCommand` function, which takes a single agent index as a parameter. Client programs advance agent mobility by first using the `uampIsMore` function to determine if there is any more mobility data for the agent given as a parameter, or if that agent has reached the end of the simulation. Clients then call the `uampAdvance` function to request the next interpolation point from the server for that agent and update the agent's `uampCommand`. (Technically, the client library requests interpolation points in bulk from the server then buffers them for the client application, improving network performance; but, the from the point of view of the client application, it is as though interpolation points are requested from the server one at a time.)

The `uampAdvance` and `uampCurrentCommand` functions present clients with an asynchronous view of agent movement, in that the client application can advance the agents

independently of each other within the simulation. There are also functions in the client library that present a synchronous view of agent movement. Recall that the `uampCommand` structure for each agent contains an earlier interpolation point and a later interpolation point. For agent A_i, denote the time (i.e., number of seconds into the simulation) of the earlier interpolation point a e_i and the time of the later interpolation point as l_i. That is, the `uampCommand` for agent A_i holds mobility data covering the range of time $[e_i, l_i]$. To advance the synchronous view of the agents, the client first calls the `uampIsAnyMore` function, which determines if the server has any more data for any agent, i.e., if l_i is smaller than the duration of the simulation for any agent A_i. If so, clients can call `uampAdvanceOldest` to advance the agent or agents in the simulation with the smallest l_i value. The `uampIntersectCommand` function, which takes an agent index as a parameter, then returns an interpolated `uampCommand` structure, covering the time period $[e, l]$ where $e = \max\{e_i\}$ and $l = \min\{l_i\}$. By using the `uampAdvanceOldest` function to advance time and the `uampIntersectCommand` function for each agent after each time advance, client programs will receive a synchronized view of agent movement. This synchronized view allows simulators to determine, e.g., when any two clients are within a certain range of each other.

An example of the structure of a UAMP client is shown in Algorithm 2.4. It is slightly simplified compared to an actual UAMP client written, e.g., in C — parameters such as the hostname and port of the UAMP server, as well as error-checking, have been removed for clarity of exposition. The example client is modelled on the information-spread client described back in Section 2.4. One agent begins with the information at time 0. The client then uses the combination of `uampIntersectCommand` and `uampAdvanceOldest` to maintain a synchronous view of all the agents' movements (i.e., the start and end times for the current command is the same across all the agents). The processMovement function (omitted because it does not actually interact with the UAMP server) is then responsible for computing whether an agent with the information comes within a threshold distance of an agent without the information during the current command's time period, then updating the time each agent received the data and the number of agents with the data.

Algorithm 2.4 Example information-spread DBS3 client

```
1:  procedure DBS3Client
2:      agentsWithData ← 1
3:      timeReceived[1] ← 0
4:      for i ← 2 to NUM_AGENTS do
5:          timeReceived[i] ← ∞
6:      client ← uampConnect(NUM_AGENTS, TIME_LIMIT, PRNG_SEED)
7:      while agentsWithData < NUM_AGENTS do
8:          for i ← 1 to NUM_AGENTS do
9:              command[i] ← uampIntersectCommand(client, i)
10:         agentsWithData ← processMovement(command, timeReceived)
11:         if uampIsAnyMore(client) = false then
12:             break
13:         uampAdvanceOldest(client)
14:     uampTerminate(client)
```

2.7 Conclusions

In this chapter, we presented a new generative mobility model inspired by the findings of Space Syntax. This model, implemented as the publicly available DBS3, is generalizable, taking only an unannotated map as input. Agents choose their destinations meaningfully, considering both centrality bias and distance decay in their selection. Additionally, the new MEA* search algorithm allows for fast runtime pathfinding, delivering more realistic minimal-turn or minimal-angle paths (as opposed to the common but unrealistic pathfinding that minimizes Euclidean distance travelled). We showed that setting the centrality bias and distance decay constants to 1 with either of the more realistic pathfinding options yields a high correlation to observed pedestrian densities in downtown Edmonton. Finally, we concluded with discussion about how DBS3 can be used by end-user programs to generate mobility data.

DBS3's design uses an unannotated map as input, which is limiting in a sense: DBS3 was unable to predict the seasonal variation in pedestrian density around a university campus. However, the tradeoff is that DBS3 has a very simple parameter space, with the map being the only nontrivial input. The graph theoretic properties of that map alone are sufficient to generate mobility data that is highly correlated with observed human mobility. Fine tuning the other parameters — namely the centrality bias and distance decay exponents — across a wide range of configured spaces is an important area of future work.

Chapter 3

The Trade-Off Between Location-of-Interest and Current-Location Privacy[3.a]

3.1 Introduction

Having studied the generation of realistic mobility data in urban environments in the previous chapter, we now turn our attention to sensor network privacy. The scenario we investigate in this chapter is one in which a user with a portable device interacts with a nearby sensor node for the purpose of querying another node in the network at a remote location. The user's query is routed through the sensor network to the queried location, and the result is then returned to the user via the reverse path. The privacy of the query and reply themselves — i.e., what the user is asking of the remote node and what the reply is — can be protected using well-studied methods. Symmetric encryption or public-key encryption designed specifically for low-powered sensor nodes [42] can be used to encrypt the query and the reply end-to-end. We emphasize that the problem studied in this chapter is *not* the privacy of the query and the reply themselves. The problem that we study is the privacy of the two *locations* in question for the user: the user's current location and their location of interest.

[3.a] This chapter is a significant extension of Vogt et al. [61].

Consider an adversary who can compromise some of the nodes in the sensor network. If we assume that the adversary is capable of compromising sensor nodes, then the adversary could learn, just as an example, a global symmetric key used to encrypt message headers sent between sensor nodes. The adversary would then learn the ultimate destination of any overheard or intercepted query message, i.e., the adversary would learn the user's location of interest.

The key contribution of this chapter is to present and study a novel method, inspired by k-anonymity [56], for obfuscating the user's location of interest: the user issues not one query to their location of interest, but rather k queries to a diverse set of locations. The trade-off to this approach, aside from the energy cost of processing an additional $k-1$ queries, is that the additional traffic from the user's current location facilitates an adversary determining that location. This chapter will study this trade-off between protecting the privacy of the user's current location and the privacy of their area of interest, and how the user should choose the fake locations to query.

It is important to note that in our scenario, the user trusts their own handheld device. However, the user does not trust the sensor network, as there may be compromised nodes. Additionally, our approach does not require that the sensor nodes use any specialized software or techniques. Our approach will work on any sensor network — the multiple queries are initiated from the handheld device. In that regard, it is fundamentally different from onion routing [17] approaches like Tor [11] to the problem of destination privacy. Not only is onion routing expensive in terms of computational power and energy usage in a sensor network, but it also requires a sensor network specifically designed to utilize onion routing.

We begin in Section 3.2 by discussing related work. We present our adversary model and assumptions in Section 3.3. We then present formal metrics for measuring the privacy of the user's current location, location of interest, and area of interest in Section 3.4. We discuss how a user should choose their $k-1$ fake locations of interest to query in Section 3.5, followed by a discussion of the related "partition" and "partitioning" problems in Section 3.6. We show the results of our k-query scheme in a featureless, uniform environment in Section 3.7, before adding to the environment the richness of mobility data from DBS3 in Section 3.8. In Section 3.9 we conclude and discuss future work.

3.2 Related Work

The relation between privacy and mobility as a general topic has been studied in the past. Hong et al. [28] examined how anonymous communication in mobile ad-hoc networks is impacted as users moved, and Andersson et al. [1] outlined the general requirements for privacy in mobile ad-hoc networks. Gruteser and Grunwald [21] study how users can remain anonymous when using location-based services, and Gedik and Liu [16] study how a k-anonymity approach could work in the context of location-based services. It is possible to identify users from the location information in cell phone records, based on their mobility patterns [64, 72]. The problem of compromised nodes affecting privacy in networks has also been investigated. Pinto et al. [49] examined the unorthodox notion of routing around compromised nodes to preserve user privacy, under the assumption that the physical location of compromised nodes is known.

Existing work on privacy in sensor networks focuses on adding features to the sensor network itself in order to protect user privacy. For example, Misra and Xue [43] show how clusters of nodes can generate and share pseudonyms used as node identities when communicating with a sink. Their work is extended by Ouyang et al. [46] to account for shared keys being compromised. However, these works only look at nodes communicating with a sink, using pseudonyms known only to the endpoints to ensure that eavesdroppers will not know which node is sending information. These schemes are not applicable when nodes need to communicate with each other to route user queries.

Ozturk et al. [47] and Kamat et al. [35] examine a problem similar to our user's queries being tracked to their source, as do Yang et al. [67] in the context of a global eavesdropper. However, these papers have a different focus, namely an adversary that moves over time towards a source node that produces a continuous stream of data. They propose a solution called phantom routing, in which each epoch's data is routed in a random directed walk away from the source, before being flooded to the sink. This solution would not be appropriate for our scenario; e.g., a message intercepted during the directed walk phase carries enough information to immediately yield the number of hops and direction to the source of the user's query. A related problem, in which the adversary moves towards the receiver of sensor network traffic over time, is investigated by Jian et al. [32]. While these approaches are useful for protecting privacy in certain situations, they are not applicable to our problem,

in which the adversary does not have to move towards the user's current location or location of interest. In a military scenario, for example, just learning either of these locations could be sufficient for the adversary.

The largest difference between these existing works and our approach, though, is that we focus on an environment in which the user trusts their own handheld device that communicates with the sensor network, but the network itself is only semi-trusted (i.e., there may be compromised nodes in the network, eavesdroppers, etc.). We don't trust the sensor network to handle our privacy for us; the multiple queries are initiated from the handheld device. As a corollary, our approach will work on any sensor network, without the need for specialized software or techniques in the sensor network.

3.3 Adversary Model and Assumptions

In this dissertation, we consider an adversary who is able to compromise individual sensors in a sensor network. There are no physical or topological restrictions on which nodes the adversary can compromise. The adversary is, for our purposes, incorporeal — that is, there is no requirement for the adversary to move physically in the environment.

Any information possessed by a compromised sensor node is instantly learned by the adversary. For example, any encryption keys known by a compromised node are learned by the adversary. Additionally, the (potentially encrypted) contents of any message routed by or overheard by a compromised node are learned by the adversary.

When a user issues k queries into a sensor network, the adversary knows the value of k. We maintain this assumption even if compromised sensor nodes do not overhear all k of the queries. We make this assumption because the entropy for even a random choice of k by the user is constrained by the size of the sensor network, i.e., it is small.

As a simplifying assumption, we treat the user as though he or she is interacting directly the sensor node closest to his or her physical location. In practice, the user would have, e.g., a handheld device that sends and receives messages to and from some node in the sensor network. However, asymptotically, the additional $2k$ message transmissions (k queries sent by the user's handheld device and k replies from the closest sensor node) make no difference to our analysis of the scheme and would add unnecessary complexity to our simulations. As such, we assume both that the user knows the closest sensor node, and communicates with

it using, e.g., low-power communication that will not be overheard by other nodes.

3.4 Privacy Metrics

We begin by defining our notation. The sensor network consists of a set $\mathcal{N}$ of nodes, where $|\mathcal{N}| = n$. The user will issue k queries, Q_i, each directed to a location L_i, where $\mathcal{L} = \{L_1, L_2, \ldots, L_k\} \subseteq \mathcal{N}$. One node, $L \in \mathcal{L}$, is the user's real location of interest, and the remaining $k-1$ nodes are fake locations of interest. However, some of the nodes in the sensor network may be compromised by the adversary, and those nodes are able to overhear communication on the shared wireless medium. Formally, the adversary attempts to decide on the current location (CL) and location of interest (LOI) of the user using the information overheard.

To study how well this k-anonymity scheme preserves the privacy of the user's LOI and CL, we require formal methods to measure the privacy levels that result from any given set of k queries. In the following section, we define the metrics used to determine how well the user's CL-privacy and LOI-privacy are protected. We follow that up with a discussion of the closely related concept of the privacy of the user's area of interest (AOI) in Section 3.4.2.

3.4.1 Privacy of the Current Location and Location of Interest

How much information a user's queries leak about the user's current location (CL) and location of interest (LOI) depends on how much information the adversary is able to overhear during the routing of the queries. To determine the CL-privacy and LOI-privacy that result from a set of queries, we will simulate the user issuing queries while malicious nodes attempt to ascertain the origin and real destination of those queries. This section demonstrates how an adversary could use the information captured by malicious nodes to narrow down the possible locations where the user could be.

Central to the technique of narrowing down possible locations is the concept of a *possible route*. In the sensor network, which we assume to be connected, there will be a routing algorithm capable of routing messages from any source node to any destination node. Consider a route $\mathcal{R} = (N_1, N_2, \ldots, N_l)$, which is a sequence of l nodes. $\mathcal{R}$ is a possible route from N_1 to N_l if it is possible that the routing algorithm used in the sensor network could have routed a message from N_1 to N_l along the path $N_1, N_2, \ldots, N_l$.

C	The current location (CL) of the user, $C \in \mathcal{N}$
C^*	The adversary's final guess as to the current location of the user
$\mathcal{C}$	The set of possible current locations of the user, as determined by the adversary
k	The total number of queries sent by the user, where $k-1$ are to fake locations of interest and one is to the real LOI
L	The user's real location of interest (LOI)
L^*	The adversary's final guess as to the user's real location of interest
$\mathcal{K}$	The set of query destinations known to the adversary, i.e., $\mathcal{K} \subseteq \mathcal{L}$
$\mathcal{L} = \{L_1, \ldots, L_k\}$	The k locations queried by the user, with $L \in \mathcal{L}$
$\mathcal{L}_N$	The set of possible locations queried by the user, as determined by the adversary, under the assumption that $C = N$ for some $N \in \mathcal{C}$
M_{AOI}, M_{CL}, M_{LOI}	The measures of the user's AOI-, CL-, and LOI-privacy respectively
n	The number of nodes in the sensor network
$\mathcal{N} = \{N_1, \ldots, N_n\}$	The nodes in the sensor network, with $\lvert\mathcal{N}\rvert = n$
$\mathrm{P_G}$ (condition)	A global, aggregate probability over all users that the given condition is true; for example, $\mathrm{P_G}(C = N \mid C \in \mathcal{C})$ is the probability that any user's current location is N, given that their current location is known to be in $\mathcal{C}$
$\mathcal{Q} = \{Q_1, \ldots, Q_k\}$	The k queries sent by the user
$\mathcal{R}_i = (N_{i,1}, \ldots, N_{i,l_i})$	The route taken by query Q_i, transmitted through l_i nodes
$\sigma(\mathcal{L}, \mathcal{N})$	A measure of how physically dispersed the elements of $\mathcal{L}$ are over the network $\mathcal{N}$
$\mathcal{U}$	The set of queries with destinations unknown to the adversary, i.e., $\mathcal{U} \subseteq \mathcal{Q}$

Table 3.1: Major symbols used in Section 3.4

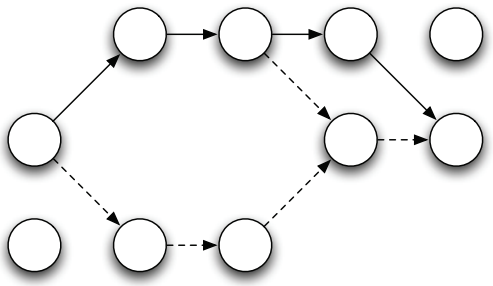


Figure 3.1: The possible routes from the westernmost central node to the easternmost central node if fixed shortest-path routing is used (solid lines) or if random shortest-path routing is used (solid and dashed lines).

In this chapter, we consider two routing algorithms: fixed shortest-path routing and random shortest-path routing. Both guarantee that any message from N_1 to N_l will arrive in the fewest possible hops. In random shortest-path routing, each node maintains a table indexed by the destination of a message, containing all possible next hops that the message could take to arrive in the fewest hops. When a message arrives at a node N_i destined for node N_l, N_i will look into its table at index N_l, and randomly choose one of the entries as the next hop. In fixed shortest-path routing, each node stores only a single next-hop choice for each possible destination. There is exactly one possible route from N_1 to N_l when fixed shortest-path routing is used, but there can be many possible routes between N_1 and N_l with random shortest-path routing. An example in Figure 3.1 shows one possible route from the westernmost central node to the easternmost central node if fixed shortest-path routing is used, but three possible paths if random shortest-path routing is used.

Recall that the user employs a mobile device to communicate with a nearby sensor node, in order to route queries through the sensor network to the LOIs. The user sends the k queries, denoted $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$, to the closest sensor node, C. As simplifying assumptions, we assume both that the user knows the closest sensor node, and that the mobile device and C communicate using low-power communication. That is, the adversary will only overhear communication between the mobile device and C if C is compromised. We also assume that the adversary knows k — the implication being that if the adversary overhears fewer than k of the user's queries, the adversary knows how many queries were not overheard. Each Q_i takes a route $\mathcal{R}_i$ through the network, starting at C and ending at L_i. Denote $\mathcal{R}_i$ as a sequence of nodes with length l_i, $\mathcal{R}_i = (N_{i,1}, N_{i,2}, \ldots, N_{i,l_i})$, where

$N_{i,1} = C$ and $N_{i,l_i} = L_i$. The goals of the adversary are to determine C and to determine the real location of interest, i.e., to determine L.

While our k-anonymity scheme can be built over any existing query mechanism, we assume that query and reply messages are designed to maximize privacy. Specifically, a query message that is being routed from C to L_i cannot contain references to C, nor can a reply; otherwise, the adversary could easily determine C (note, though, that even if a reference to C were contained in queries and replies, this scheme could still be used to preserve the privacy of the user's location of interest, just not their current location). Query messages contain four pieces of information in addition to the query itself: a unique query identifier for Q_i; the destination L_i (which may be the real location of interest or a fake one); the identifier for the node currently transmitting the query, $N_{i,j}$; and, the next hop in the route, $N_{i,j+1}$. When node $N_{i,j+1}$ receives the query, it remembers the previous node in the route for query Q_i, $N_{i,j}$. Replies to the query message contain only the query identifier for Q_i. When node $N_{i,j+1}$ receives a reply to the query, to be routed back to C, $N_{i,j+1}$ uses its memory to identify $N_{i,j}$ as the next hop in the reply path, and sends the reply to $N_{i,j}$ (without unnecessary information such as the identity of $N_{i,j}$ or $N_{i,j+1}$). However, we assume the worst case: the adversary is able to determine which sensor node is transmitting a reply message if that reply is overheard.

If a query message for query Q_i is overheard and the header can be decrypted by the adversary (e.g., if a compromised node is routing the query, or if a global encryption key is used for all transmissions in the sensor network), the adversary learns one of the LOIs, L_i. Additionally, the adversary learns one hop that the query took along the route from C to L_i: $N_{i,j}$ and $N_{i,j+1}$, for $1 \leq j < l_i$, where j and l_i are unknown to the adversary. That is, the adversary learns two consecutive elements in the route, but neither their position in the route nor the length of the route. If a reply message for query Q_i is overheard, the adversary learns only a single node that was involved in the route: $N_{i,j}$, for $1 < j \leq l_i$, again with unknown j and l_i.

The adversary can also construct a list of sensor nodes that were certainly not involved in routing Q_i. Because the adversary has complete knowledge of every message that was routed through compromised nodes in the user's sensor network, the adversary knows which compromised nodes were not involved in routing Q_i. An additional consideration for the adversary is that the malicious nodes could monitor all communication by honest sensor

nodes in their communication range. An adversary may conclude that if a malicious node that is monitoring an honest node within its range did not hear the honest node produce any traffic regarding query Q_i, then the honest node must not have been involved in routing Q_i. However, the malicious node may not have heard a message transmitted by the honest node due to interference or a collision. As such, we assume that the adversary will restrict the list of nodes that certainly were not involved with query Q_i to the set of compromised nodes in the sensor network that did not route Q_i.

The key insight for the adversary, having collected information on the routes of the queries, is that if no possible route from a sensor node N to the known destination of Q_i, L_i, is consistent with the known information about Q_i, then N could not have been the origin of the query. Formally, a route $\mathcal{R} = \{N_1, N_2, \ldots N_l\}$ from N to L_i is *consistent* with that information, assuming the destination L_i of Q_i is known, if:

1. $\mathcal{R}$ is a possible route from N to L_i (i.e., $N_1 = N$, $N_l = L_i$, and the routing algorithm could have used this route);

2. For every query message about Q_i decrypted by the adversary, sent by $N_{i,j}$ to $N_{i,j+1}$, there is some $k < l$ such that $N_k = N_{i,j}$ and $N_{k+1} = N_{i,j+1}$;

3. For ever query message about Q_i overheard but not decrypted by the adversary, sent by $N_{i,j}$ to an unknown destination, there is some $k < l$ such that $N_k = N_{i,j}$;

4. For every reply message about Q_i overheard by the adversary, sent by $N_{i,j}$, there is some $k > 1$ such that $N_k = N_{i,j}$; and,

5. No node that is known not to have routed Q_i appears in $\mathcal{R}$.

A simple example of the insight provided to the adversary from determining whether or not possible routes are consistent with known information is illustrated in Figure 3.2. In the illustrated example, there are two malicious nodes, one of which routed a query to the LOI and another that did not. Using this information, the adversary can narrow down the potential set of current locations of the user.

There is a second insight, about queries for which the adversary does not know the destination. The adversary may still know some information about such a Q_i (e.g., non-decrypted query messages, overheard reply messages, or knowledge about compromised nodes that did not route Q_i). Denote $\mathcal{D}$ as a set of possible destinations for the queries

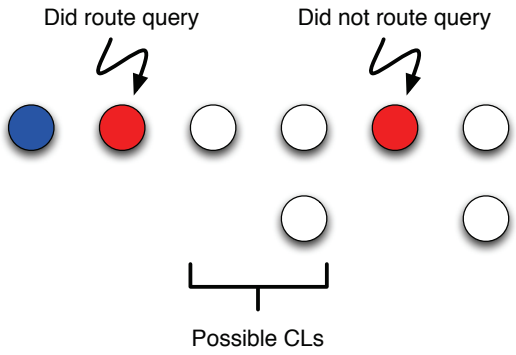


Figure 3.2: The westernmost (blue) node is one of the queried locations, and knowledge of which malicious (red) nodes did and did not route the query to the LOI allows the adversary to narrow down the user's potential current locations.

with unknown destinations. Specifically, let $\mathcal{D}$ be the set of all nodes in $\mathcal{N}$ that are not compromised and are not the destination of a query with a known destination. Let $\mathcal{U}$ be the set of queries for which the destination is unknown to the adversary, where $0 \leq |\mathcal{U}| \leq k$. If it is not possible to assign a unique destination from $\mathcal{D}$ to each query in $\mathcal{U}$, in such a way as to ensure that there is a possible route from sensor node N to the destination of each query in $\mathcal{U}$ that is consistent with all of the known information about that query, then N cannot be the origin of the user's queries.

Using the pseudo-code algorithm presented in Algorithm 3.1 and Algorithm 3.1.1, an adversary can narrow down the possible current locations of the user. Additionally, each assignment of potential destinations to the queries that is consistent with the information known by the adversary represents all the possible query destinations for each possible current location. The pseudocode returns a set $\mathcal{C}$, representing all possible current locations for the user, and sets $\mathcal{L}_{\mathcal{N}}$ for each $N \in \mathcal{C}$, representing all the possible locations of interest (i.e., query destinations) for the user if they are indeed at current location N.

Algorithm 3.1 begins, assuming that the user's current location is not compromised, by considering all honest nodes as potential current locations on line 4. However, any honest nodes from which queries consistent with the adversary's known information could not have been sent are removed as potential current locations on line 10. Potential query destinations from each potential current location obviously includes all known destinations, as computed on line 18. However, the adversary must compute potential destinations for queries with unknown destinations as well, as done on lines 13–17.

The pseudocode presented in Algorithm 3.1.1 illustrates the most natural way of thinking

Algorithm 3.1 Pseudocode for finding all possible current locations and locations of interest for the user

```
1: function NARROW-POSSIBLE-CLS-AND-LOIS
2:     if the node C with which the user is communicating is compromised then
3:         return C = {C}, L_C = {L_1, ..., L_k}
4:     N_honest ← {N ∈ N | N is not compromised}
5:     C ← N_honest
6:     L_N ← {} ∀ N ∈ C
7:     for all queries Q_i for which the destination is known do
8:         for all N ∈ C do
9:             if POSSIBLE-CONSISTENT(N, Q_i) = FALSE then
10:                Remove N from C
11:    K ← {L_i | L_i is known}
12:    U ← {Q_i | L_i is unknown}
13:    if |U| > 0 then
14:        D ← N_honest \ K
15:        for all N ∈ C do
16:            if ANALYSE-UNKNOWN(N, L_N, D, U) = FALSE then
17:                Remove N from C
18:    L_N ← L_N ∪ K ∀ N ∈ C
19:    return C, {L_N | N ∈ C}

20: function POSSIBLE-CONSISTENT(src, Q)
21:    L ← the destination of query Q
22:    if ∃ a possible route from src to L, consistent with everything known about Q then
23:        return TRUE
24:    else
25:        return FALSE
```

Sub-algorithm 3.1.1 A naïve implementation of the analysis of queries with unknown destinations

```
1: function ANALYSE-UNKNOWN(N, L_N, D, U)
2:     foundAssignment ← FALSE
3:     for D' ← each of the (|D| choose |U|) choices of |U| destinations from D do
4:         for all |D'|! assignments of nodes in D' as destinations for U do
5:             if POSSIBLE-CONSISTENT(N, Q) = TRUE ∀ Q ∈ U then
6:                 foundAssignment ← TRUE
7:                 L_N ← L_N ∪ D'
8:     return foundAssignment
```

about how the adversary can use information gained about queries with unknown destinations. The adversary considers all possible assignments of destinations to all queries with unknown locations on line 4, adding any possible destinations to the pool of potential destinations for the current locations on 7. If there are no possible assignments of destinations to queries with unknown destinations, the potential current location is removed from the pool of current location possibilities by a return of FALSE to Algorithm 3.1. However, the naïve implementation of this pseudocode would be highly inefficient. The following lemma allows the adversary to implement the algorithm far more efficiently.

Lemma 3.1. *Given a family of sets of integers, $\{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ where $\mathcal{S}_i \subseteq \mathbb{Z}$, we call a set $\mathcal{G}_m = \{g_1, \ldots, g_m\}$ a* sampling *if $g_i \in \mathcal{S}_i$ for all $1 \leq i \leq m$, and $g_i \neq g_j$ for all $i \neq j$. Let $\mathcal{G}'_m = \{\mathcal{G}_m \mid \mathcal{G}_m$ is a sampling$\}$, i.e., $\mathcal{G}'_m$ is a set of sets, where each $\mathcal{G}_m \in \mathcal{G}'_m$ is a set of m unique integers, with each integer coming from a different one of the $\mathcal{S}_i$, $1 \leq i \leq m$. Then either $\mathcal{G}'_m = \emptyset$ or*

$$\bigcup_{\mathcal{G}_m \in \mathcal{G}'_m} \mathcal{G}_m = \bigcup_{1 \leq i \leq m} \mathcal{S}_i \, .$$

Proof. If there is no way to choose a unique integer from each of $\mathcal{S}_1, \ldots, \mathcal{S}_m$ (including if $\mathcal{S}_i = \emptyset$ for any i), then $\mathcal{G}'_m = \emptyset$, and we are done. Assume there is a way to choose m unique integers, one from each of the $\mathcal{S}_i$ (which implies that $\mathcal{S}_i \neq \emptyset$ for all i). The proof will be by induction on $m \in \mathbb{Z} \geq 1$.

If $m = 1$, then $\mathcal{G}'_m = \{\{s\} \mid s \in \mathcal{S}_1\}$, because $\{s\}$ is a sampling if and only if $s \in \mathcal{S}_1$. As such,

$$\bigcup_{\mathcal{G}_m \in \mathcal{G}'_m} \mathcal{G}_m = \bigcup_{s \in \mathcal{S}_1} \{s\} = \mathcal{S}_1 = \bigcup_{1 \leq i \leq m} \mathcal{S}_i \, .$$

Assume for $j \in \mathbb{Z} \geq 1$,

$$\bigcup_{\mathcal{G}_j \in \mathcal{G}'_j} \mathcal{G}_j = \bigcup_{1 \leq i \leq j} \mathcal{S}_i \, ,$$

and let $\mathcal{S}_{j+1} \subseteq \mathbb{Z}$ be another set of integers. If it is not possible to choose $j+1$ unique integers, one from each of $\mathcal{S}_1, \ldots, \mathcal{S}_{j+1}$, then $\mathcal{G}'_{j+1} = \emptyset$ and we are done. Assume it is possible to choose $j+1$ unique integers as described above. If there are any elements $e \in \mathcal{S}_{j+1}$ that are not contained in $\cup_{1 \leq i \leq j} \mathcal{S}_i$, then a sampling $\mathcal{G}_{j+1}$ could be constructed as

$\mathcal{G}_{j+1} = \mathcal{G}_j \cup \{e\}$, for any sampling $\mathcal{G}_j$ on $\mathcal{S}_1, \ldots, \mathcal{S}_j$. Hence,

$$\bigcup_{\mathcal{G}_{j+1} \in \mathcal{G}'_{j+1}} \mathcal{G}_{j+1} = \bigcup_{1 \leq i \leq j+1} \mathcal{S}_i \ ,$$

because there is guaranteed to be a sampling $\mathcal{G}_{j+1}$ that contains any element of $\mathcal{S}_{j+1}$ not contained in $\cup_{1 \leq i \leq j} \mathcal{S}_i$. □

Given a potential current location $N \in \mathcal{C}$ and queries with unknown destinations $\mathcal{U}$, we can compute potential destinations $\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{U}|}$ for each of the queries. We can then test if it is possible to choose a unique destination for each query, i.e., choose $s_i \in \mathcal{S}_i$ with $s_i \neq s_j$ for all $i \neq j$, where $1 \leq i, j \leq |\mathcal{U}|$. If it is possible to choose *even a single set* of unique destinations, then Lemma 3.1 tells us that the set of all potential destinations for all the queries with unknown destinations is $\cup_{1 \leq i \leq |\mathcal{U}|} \mathcal{S}_i$, i.e., no potential destinations can be excluded. Testing whether it is possible to choose a unique destination from each $\mathcal{S}_i$ can be accomplished by creating a graph $G = \langle (X, Y), E \rangle$, where X contains a vertex for each of the $|\mathcal{U}|$ sets $\mathcal{S}_i$, Y contains a vertex for each potential destination in $\cup_{1 \leq i \leq |\mathcal{U}|} \mathcal{S}_i$, and E contains an edge between the vertex in X representing $\mathcal{S}_i$ and the vertex in Y representing s if and only if $s \in \mathcal{S}_i$. It is possible to choose a unique destination from each $\mathcal{S}_i$ if and only if there is a bipartite matching on G of size $|\mathcal{U}|$, and testing for a bipartite matching is a well-studied problem [8, pp. 664–669]. A revised, efficient algorithm for analyzing queries with unknown destinations is illustrated as Algorithm 3.1.2.

Given the adversary's ability to narrow down the user's possible current locations to a set $\mathcal{C}$ and possible locations of interest to a set $\mathcal{L}_N$ for each $N \in \mathcal{C}$, a simple yet expressive way to measure the security of a user's location of interest and current location is the probability that the adversary will guess the LOI and CL incorrectly, respectively. As a simple example, if the adversary knows the value of k (a necessary assumption to make, as it is unsafe to assume otherwise), and if the adversary overhears all k queries in a uniform, featureless environment (i.e., all n potential query destinations in the network are equally probable destinations), then the user's resulting LOI-privacy will be $\frac{k-1}{k}$, because the adversary can merely guess at random that one of the k destinations is the correct one.

More formally, we begin by assuming the adversary knows — from a global, aggregate perspective — the probability that a given node in the network is a user's current location,

Sub-algorithm 3.1.2 An efficient implementation of the analysis of queries with unknown destinations

```
1: function ANALYSE-UNKNOWN(N, L_N, D, U)
2:     S_i ← {}  ∀ 1 ≤ i ≤ |U|
3:     for all U_i ∈ U do
4:         for all D ∈ D do
5:             if POSSIBLE-CONSISTENT-ASSUME(N, D, U_i) then
6:                 Add D to S_i
7:     X ← a vertex for each S_i, 1 ≤ i ≤ |U|
8:     Y ← a vertex for each destination in ∪_{1≤i≤|U|} S_i
9:     E ← edges between vertex representing S_i and vertex representing D iff D ∈ S_i
10:    if there exists a bipartite matching of size |U| on G = ⟨(X, Y), E⟩ then
11:        L_N ← L_N ∪ S_1 ∪ ... ∪ S_{|U|}
12:        return TRUE
13:    else
14:        return FALSE

15: function POSSIBLE-CONSISTENT-ASSUME(src, L, Q)
16:    Assume, just for now, that the destination of Q is L
17:    if ∃ a possible route from src to L, consistent with all known/assumed about Q then
18:        return TRUE
19:    else
20:        return FALSE
```

and the probability that each node is a user's location of interest given a known current location. We denote these probabilities as $\mathrm{P_G}\left(C = N\right)$ and $\mathrm{P_G}\left(L = M \mid C = N\right)$ respectively. For now, we assume that $\mathrm{P_G}\left(C = N\right) = \mathrm{P_G}\left(L = M \mid C = N\right) = \frac{1}{n}$ for all nodes N and M in the network. We will revisit this assumption in Section 3.8.

First, the adversary corrects the probability of each node being the current location, based on the set of possible current locations. That is, the adversary computes

$$\mathrm{P_G}\left(C = N \mid C \in \mathcal{C}\right) = \begin{cases} \dfrac{\mathrm{P_G}\left(C = N\right)}{\sum_{P \in \mathcal{C}} \mathrm{P_G}\left(C = P\right)} & \text{if } N \in \mathcal{C} \\ 0 & \text{if } N \notin \mathcal{C} \end{cases}$$

for each $N \in \mathcal{C}$. From the computation of this corrected probability, we naturally get the formal definition of CL-privacy, which is the probability of the adversary incorrectly guessing the user's current location. Define

$$C^* = \arg\max_{N \in \mathcal{C}} \left\{\mathrm{P_G}\left(C = N \mid C \in \mathcal{C}\right)\right\} \ ,$$

i.e., C^* is the adversary's best guess as to the user's current location. The formal definition

of CL-privacy can then be stated as

$$\mathrm{M}_{CL} = 1 - \mathrm{P}_{\mathrm{G}}\left(C = C^* \mid C \in \mathcal{C}\right).$$

The user's LOI-privacy is computed in a similar fashion: the probability that the adversary incorrectly guesses the user's actual location of interest. First, note that the user's actual location of interest may be a destination of one of the k queries where the destination is known to the adversary (e.g., if a compromised node routed the query), or it may be the destination of a query where the destination is unknown to the adversary. Based on the value of $\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = N\right)$, an adversary may actually guess that the user's real location of interest is a location that is not known to be a query destination, if the adversary does not know the destination of all k queries. To account for this possibility, we first need to apply a "corrective weight" to the probability that a given location $M \in \mathcal{N}$ is the user's real LOI, assuming that another location $N \in \mathcal{N}$ is their current location. Let $\mathcal{K}$ be the set of all query destinations that are known to the adversary (i.e., $\mathcal{K} \subseteq \mathcal{L}$), then denote

$$\chi(N, M) = \begin{cases} 1 & \text{if } M \in \mathcal{K} \\ 0 & \text{if } M \notin \mathcal{K} \text{ and } |\mathcal{L}_N| = |\mathcal{K}| \\ \dfrac{k - |\mathcal{K}|}{|\mathcal{L}_N| - |\mathcal{K}|} & \text{otherwise.} \end{cases}$$

Using χ as a corrective weight, we can compute the weighted probability that M is the user's location of interest, given that N is their current location and their location of interest is in $\mathcal{L}_N$, as

$$\phi(N, M) = \begin{cases} \dfrac{\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = N\right) \cdot \chi(N, M)}{\sum_{P \in \mathcal{L}_N} \mathrm{P}_{\mathrm{G}}\left(L = P \mid C = N\right) \cdot \chi(N, P)} & \text{if } M \in \mathcal{L}_N \\ 0 & if M \notin \mathcal{L}_N \,, \end{cases}$$

for each $N \in \mathcal{C}$ and $M \in \mathcal{L}_N$. We subsequently define $\psi(N, M)$ as the probability that both N is the current location and M is the real location of interest. Namely,

$$\psi(N, M) = \mathrm{P}_{\mathrm{G}}\left(C = N \mid C \in \mathcal{C}\right) \cdot \phi(N, M).$$

We assume that the goal for the adversary is to choose the most likely real LOI, indepen-

dent of the adversary's guess about the user's current location (we return to this assumption shortly). In this case, the adversary would compute the probability that a node M is the user's real location of interest over all possible current locations as

$$\omega(M) = \sum_{N \in \mathcal{C}} \psi\left(N, M\right) ,$$

for all possible LOIs M (i.e., for any $M \in \cup_{N \in \mathcal{C}} \mathcal{L}_N$). The adversary then guesses the real LOI as $L^* = \arg\max_M \omega(M)$. As such, the user's resulting LOI-privacy is

$$\mathrm{M}_{LOI} = 1 - \omega\left(L^*\right) .$$

Lemma 3.2. *If the adversary knows that any user's real location of interest is in fact their current location, i.e.,*

$$\mathrm{P_G}\left(L = N \mid C = M\right) = \begin{cases} 0 & \text{if } N \neq M \\ 1 & \text{if } N = M \ , \end{cases}$$

then $\mathrm{M}_{CL} = \mathrm{M}_{LOI}$.

Proof. First, note that if $\mathrm{P_G}\left(L = N \mid C = M\right)$ is as described above, then

$$\begin{aligned} \phi(N, M) &= \begin{cases} \dfrac{\mathrm{P_G}\left(L = M \mid C = N\right) \cdot \chi(N, M)}{\sum_{P \in \mathcal{L}_N} \mathrm{P_G}\left(L = P \mid C = N\right) \cdot \chi(N, P)} & \text{if } M \in \mathcal{L}_N \\ 0 & if M \notin \mathcal{L}_N \end{cases} \\ &= \begin{cases} \dfrac{1 \cdot \chi(N, M)}{1 \cdot \chi(N, M) + \sum_{P \in \mathcal{L}_N \setminus \{M\}} 0 \cdot \chi(N, P)} & \text{if } M \in \mathcal{L}_N \text{ and } N = M \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} 1 & \text{if } M \in \mathcal{L}_N \text{ and } N = M \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

We can further simplify ϕ by noting that if $N = M$, then under our assumption about $\mathrm{P_G}\left(L = N \mid C = M\right)$ (which implies one of the k queried locations will always be the user's current location), it is guaranteed that $M \in \mathcal{L}_N$. If the adversary has compromised the user's current location C, the adversary would process the query directed to the real location of

interest (i.e., C) and add C to $\mathcal{L}_C$. If the adversary has not compromised the user's current location, then the adversary would not overhear all of the user's k queries, and would, for any possible current location N, add N to $\mathcal{L}_N$, since a query to a non-compromised current location always represents a possible assignment of a destination to a query with an unknown destination. As such,

$$\phi(N, M) = \begin{cases} 1 & \text{if } N = M \\ 0 & \text{otherwise.} \end{cases}$$

Using the simplified form of ϕ, we can simplify ω as well:

$$\begin{aligned}
\omega(M) &= \sum_{N \in \mathcal{C}} \psi(N, M) \\
&= \sum_{N \in \mathcal{C}} \mathrm{P_G}(C = N \mid C \in \mathcal{C}) \cdot \phi(N, M) \\
&= \begin{cases} \mathrm{P_G}(C = M \mid C \in \mathcal{C}) \cdot 1 + \sum_{N \in \mathcal{C} \setminus \{M\}} \mathrm{P_G}(C = N \mid C \in \mathcal{C}) \cdot 0 & \text{if } M \in \mathcal{C} \\ \sum_{N \in \mathcal{C}} \mathrm{P_G}(C = N \mid C \in \mathcal{C}) \cdot 0 & \text{if } M \notin \mathcal{C} \end{cases} \\
&= \begin{cases} \mathrm{P_G}(C = M \mid C \in \mathcal{C}) & \text{if } M \in \mathcal{C} \\ 0 & \text{if } M \notin \mathcal{C}\,. \end{cases}
\end{aligned}$$

Next, observe that $\mathcal{C} \subseteq \cup_{N \in \mathcal{C}} \mathcal{L}_N$, by our earlier observation that $N \in \mathcal{L}_N$ for all $N \in \mathcal{C}$. Therefore,

$$\max_{M \in \cup_{N \in \mathcal{C}} \mathcal{L}_N} \omega(M) = \max_{M \in \mathcal{C}} \omega(M)\,,$$

since $\omega(M) = 0$ for any $M \in \cup_{N \in \mathcal{C}} \mathcal{L}_N \setminus \mathcal{C}$. Hence,

$$\begin{aligned}
\mathrm{M}_{LOI} &= 1 - \omega(L^*) \\
&= 1 - \max_{M \in \cup_{N \in \mathcal{C}} \mathcal{L}_N} \omega(M) \\
&= 1 - \max_{M \in \mathcal{C}} \omega(M) \\
&= 1 - \max_{M \in \mathcal{C}} \mathrm{P_G}(C = M \mid C \in \mathcal{C}) \\
&= \mathrm{M}_{CL}\,,
\end{aligned}$$

demonstrating that the two measures are equivalent when the adversary knows that the

user's current location is their real location of interest. □

Lemma 3.3. *If the user's current location is compromised, then*

$$\mathrm{M}_{CL} = 0$$

and

$$\mathrm{M}_{LOI} = 1 - \max_{M \in \mathcal{L}} \mathrm{P}_{\mathrm{G}} \left(L = M \mid C = C^* \text{ and } L \in \mathcal{L} \right) .$$

Consider a uniform, featureless environment — that is, an environment in which, from any current location, the probability of each location being any user's real LOI is $\frac{1}{n}$*. If the user's current location is compromised in a uniform, featureless environment, then*

$$\mathrm{M}_{LOI} = \frac{k-1}{k} .$$

Proof. If the node at the user's current location C is compromised, the adversary will "guess" with certainty that the user's current location is $C^* = C$ — i.e., $\mathcal{C} = \{C\}$. The adversary will also know all of the locations queried, i.e., $\mathcal{L}_C = \mathcal{L}$. In this case, the user's CL-privacy drops to

$$\begin{aligned} \mathrm{M}_{CL} &= 1 - \mathrm{P}_{\mathrm{G}} \left(C = C^* \mid C \in \mathcal{C} \right) \\ &= 1 - 1 = 0 . \end{aligned}$$

The user's LOI-privacy also drops to

$$\begin{aligned}
\mathrm{M}_{LOI} &= 1 - \omega\left(L^*\right) \\
&= 1 - \max_{M \in \cup_{N \in \mathcal{C}} \mathcal{L}_N} \omega\left(M\right) \\
&= 1 - \max_{M \in \mathcal{L}} \omega\left(M\right) \\
&= 1 - \max_{M \in \mathcal{L}} \left[\sum_{N \in \mathcal{C}} \psi\left(N, M\right) \right] \\
&= 1 - \max_{M \in \mathcal{L}} \psi\left(C^*, M\right) \\
&= 1 - \max_{M \in \mathcal{L}} \left[\mathrm{P}_{\mathrm{G}}\left(C = C^* \mid C \in \mathcal{C}\right) \cdot \phi\left(C^*, M\right) \right] \\
&= 1 - \max_{M \in \mathcal{L}} \phi\left(C^*, M\right) \\
&= 1 - \max_{M \in \mathcal{L}} \left[\frac{\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = C^*\right) \cdot \chi(C^*, M)}{\sum_{P \in \mathcal{L}} \mathrm{P}_{\mathrm{G}}\left(L = P \mid C = C^*\right) \cdot \chi(C^*, P)} \right] .
\end{aligned}$$

Note that, with the user's current location compromised, $M \in \mathcal{K}$ for all $M \in \mathcal{L}$. Therefore, $\chi(C^*, M) = 1$ for all $M \in \mathcal{L}$. Hence,

$$\begin{aligned}
\mathrm{M}_{LOI} &= 1 - \max_{M \in \mathcal{L}} \left[\frac{\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = C^*\right) \cdot \chi(C^*, M)}{\sum_{P \in \mathcal{L}} \mathrm{P}_{\mathrm{G}}\left(L = P \mid C = C^*\right) \cdot \chi(C^*, P)} \right] \\
&= 1 - \max_{M \in \mathcal{L}} \left[\frac{\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = C^*\right)}{\sum_{P \in \mathcal{L}} \mathrm{P}_{\mathrm{G}}\left(L = P \mid C = C^*\right)} \right] \\
&= 1 - \max_{M \in \mathcal{L}} \mathrm{P}_{\mathrm{G}}\left(L = M \mid C = C^* \text{ and } L \in \mathcal{L}\right) .
\end{aligned}$$

In a uniform, featureless environment, where from any current location the probability of any location being any user's real LOI is $\frac{1}{n}$, that computation simplifies to

$$\begin{aligned}
\mathrm{M}_{LOI} &= 1 - \max_{M \in \mathcal{L}} \left[\frac{\mathrm{P}_{\mathrm{G}}\left(L = M \mid C = C^*\right)}{\sum_{P \in \mathcal{L}} \mathrm{P}_{\mathrm{G}}\left(L = P \mid C = C^*\right)} \right] \\
&= 1 - \max_{M \in \mathcal{L}} \left[\frac{\frac{1}{n}}{\sum_{P \in \mathcal{L}} \frac{1}{n}} \right] \\
&= 1 - \max_{M \in \mathcal{L}} \left[\frac{\frac{1}{n}}{k \cdot \frac{1}{n}} \right] \\
&= 1 - \frac{1}{k} = \frac{k-1}{k} ,
\end{aligned}$$

thus completing the proof. □

Remark 3.4. Previously, we assumed that the adversary would guess the user's current location and real location of interest independently of one another (i.e., a best attempt to get at least one, if not both, of those two locations correct). In some contexts, however (e.g., intercepting a user who is moving from their current location to a location of interest), it might be in the interest of the adversary to choose the most likely $\langle C^*, L^* \rangle$ pair. That is, the adversary wants to maximize their chances of guessing *both* the user's CL and LOI simultaneously correctly. Such an adversary would choose C^* and L^* as

$$\arg\max_{\langle C^*, L^* \rangle} \psi\left(C^*, L^*\right),$$

and the computation of the user's CL- and LOI-privacy would remain the same as

$$\mathrm{M}_{CL} = 1 - \mathrm{P}_{\mathrm{G}}\left(C = C^* \mid C \in \mathcal{C}\right)$$

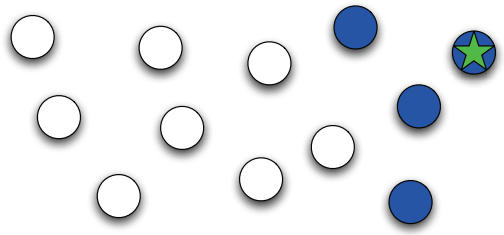
and

$$\mathrm{M}_{LOI} = 1 - \omega\left(L^*\right)$$

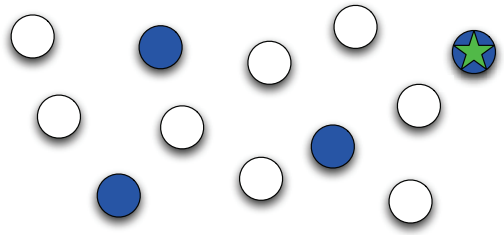
respectively. For the purposes of this dissertation, however, we restrict our investigation to an adversary who guesses the user's current location and location of interest independently, as such an adversary is stronger.

3.4.2 Privacy of the Area of Interest

If it is equally probable that any of the k queried nodes is the real LOI (i.e., in the featureless environment), then the adversary cannot learn which of the k nodes queried is the real LOI. Given this assumption, what is actually meant when we discuss protecting the privacy of the user's LOI? Consider the two scenarios in Figure 3.3. In both cases, the user's real LOI is the starred node in the northeast, and $k - 1 = 3$ fake LOIs are chosen to disguise the real LOI. However, in Figure 3.3(a), all four LOIs are clustered in the east. While an adversary who overhears these queries would not know which node is of interest to the user, it would be obvious that the user is interested in the eastern region of the sensor network. In Figure 3.3(b), the four LOIs are dispersed throughout the sensor network, obfuscating the area of the network in which the user is interested. We need to define a measure of how dispersed the k LOI choices are. That is, we want to measure how well-protected the



(a) A poor choice.



(b) A good choice.

Figure 3.3: A comparison of two choices for $k-1=3$ fake LOIs (dark nodes) given one fixed real LOI (starred node).

privacy of the user's area of interest (AOI) is.

To measure how well the set $\mathcal{L}$ of k LOIs preserves the user's AOI-privacy, we define a function $\sigma(\mathcal{L},\mathcal{N})$ to measure how dispersed the LOIs are over the network $\mathcal{N}$. To allow for comparisons of different methods of choosing the fake LOIs over networks with different topologies, we normalize the score returned by σ. Let $\sigma_{min}(k,\mathcal{N})$ and $\sigma_{max}(k,\mathcal{N})$ be the minimal and maximal values returned by σ, over all $\binom{n}{k}$ possible sets of k LOIs. The normalized measure of AOI-privacy is defined as

$$\mathrm{M}_{AOI}(\mathcal{L},\mathcal{N}) = \begin{cases} 1 & \text{if } \sigma_{min}(k,\mathcal{N}) = \sigma_{max}(k,\mathcal{N}) \\ \dfrac{\sigma(\mathcal{L},\mathcal{N}) - \sigma_{min}(k,\mathcal{N})}{\sigma_{max}(k,\mathcal{N}) - \sigma_{min}(k,\mathcal{N})} & \text{otherwise.} \end{cases}$$

The function σ must have the property that it returns large values for sets $\mathcal{L}$ with minimal clustering of the LOIs, and small values otherwise. Ideally, it is also easy to compute on the user's low-powered, mobile device prior to issuing a query. Next, we discuss some alternatives for σ.

(a) A choice of LOIs that results in two clusters.

(b) A choice of LOIs that disperses the LOIs across the entire network.

Figure 3.4: A comparison of two choices for $k = 3$ LOIs in a five-node sensor network.

Variance-Based σ.

One straightforward approach is to compute a variance-like quantity for the positions of the nodes in $\mathcal{L}$, measuring the squared distance between LOIs:

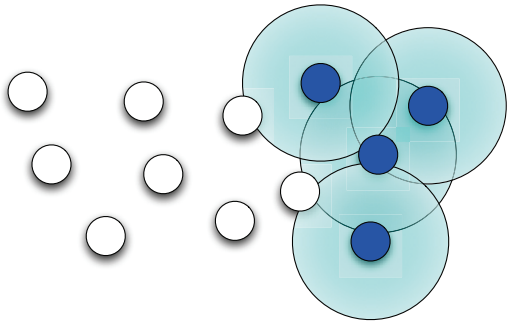
$$\sigma\left(\mathcal{L}, \mathcal{N}\right) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \mathrm{D}\left(L_i, L_j\right)^2 \ ,$$

where D is the Euclidean distance between two nodes. However, this definition of σ does not penalize clustering properly. Consider the examples in Figure 3.4. The choice of fake LOIs in Figure 3.4(a) has two of the LOIs clustered together. In Figure 3.4(b) the three LOIs are dispersed evenly. However, the σ value for the first choice is higher. This result is similar to how the variance for the data set $\{1, 2, 5\}$ is higher than the variance for the data set $\{1, 3, 5\}$.

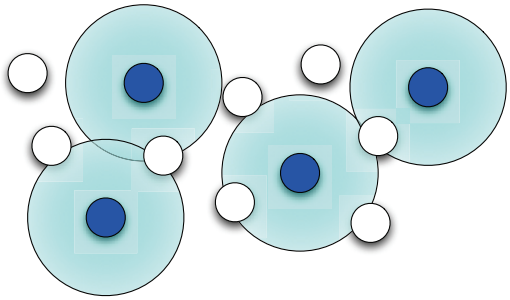
Union of Circles σ.

Another approach is to determine how much overlap exists among the regions around each LOI. Specifically, one can draw a circle of radius r around each LOI and let σ be the area of the union of the k circles. The more clustered the LOIs, the more overlap there would be among the circles, resulting in a smaller area. An example is presented in Figure 3.5. With k clustered LOIs, shown in Figure 3.5(a), there is significant overlap among the circles, unlike with the non-clustered choice shown in Figure 3.5(b).

One way to choose r is to define d as the maximum distance between any two nodes in $\mathcal{N}$, then set $r = \frac{d}{2}$. This definition ensures that there is no overlap between the regions of the two sensor nodes that are farthest apart, but the regions surrounding any two sensor



(a) A choice of LOIs with significant overlap among the circles.



(b) A choice of LOIs with minimal overlap among the circles.

Figure 3.5: A comparison of two choices for $k = 4$ LOIs in a sensor network, each surrounded by a circle with a fixed, arbitrary radius.

nodes that are closer together will overlap. Alternately, r could be chosen as the maximal size such that k nodes could be chosen from $\mathcal{N}$ with no overlap among the k associated circles.

While this σ properly penalizes clustering, it is unclear what an ideal value for r would be. Ultimately, we do not investigate this issue further, because this σ is unnecessarily computationally expensive, compared to the other σ alternatives mentioned subsequently. Note, however, that there are known methods for computing the area of the union of circles, including an $\mathcal{O}(k^2)$ deterministic algorithm and an $\mathcal{O}(k)$ Monte Carlo algorithm [55].

Minimal Pairwise Distance σ.

A simple computation that penalizes two LOIs being close together is to measure the minimal distance between any two LOIs. Formally, let

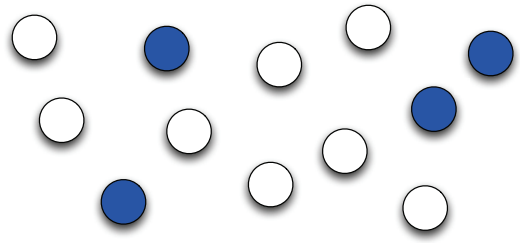
$$\sigma\left(\mathcal{L},\mathcal{N}\right) = \min_{j\neq i}\left\{\mathrm{D}\left(L_i, L_j\right)\right\} \ ,$$

where D is the Euclidean distance between two nodes. The closer the two closest L_i become, the lower the value of σ. Unfortunately, while this approach properly punishes having two queried nodes close together, it does not distinguish between a mediocre situation (e.g., having two of the L_i close together) and a bad situation (e.g., having all of the L_i close together). Consider the situation in Figure 3.6. In Figure 3.6(a), the situation is not ideal: two of the LOIs are close together; that said, it is not a terrible physical distribution of the queried location. Figure 3.6(b), on the other hand, illustrates a far worse situation, in which all of the LOIs are clustered. In both cases, however, the value of $\sigma\left(\mathcal{L},\mathcal{N}\right)$ will be the same: the distance between the two closest nodes (i.e., the two nodes in the upper right).

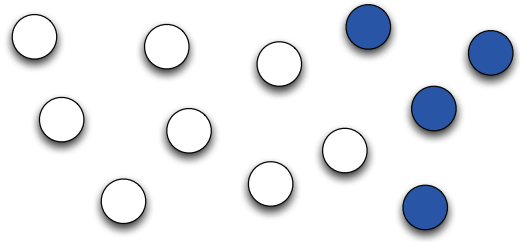
Sum of Minima σ.

Measuring the sum of minimum distances from each LOI to any other LOI penalizes clustering properly and is easy to compute. Formally, let

$$\sigma\left(\mathcal{L},\mathcal{N}\right) = \sum_{i=1}^{k}\min_{j\neq i}\left\{\mathrm{D}\left(L_i, L_j\right)\right\} \ ,$$



(a) A mediocre choice with two LOIs close together.



(b) A bad choice with all of the LOIs close together.

Figure 3.6: A comparison of two choices for $k = 4$ LOIs in a sensor network, both having at least two LOIs close together.

where D is again the Euclidean distance between two nodes. That is, the farther away each of the L_i are from each other, the higher the value of σ. Returning to the five-node examples in Figure 3.4, this σ returns a higher value in the scenario where the LOIs are non-clustered; and, it can be computed quickly, given the location of every node in $\mathcal{L}$. This σ function is essentially a more expressive version of the minimal-pairwise-distance version of σ, in that it not only properly punishes clustering, but also properly distinguishes between mediocre and bad distributions of queried locations.

Considering these four possible σ functions, we use the sum of minima function for most of this dissertation (it will be noted explicitly when one of the other σ functions is used). We chose this σ function since we assume that the user, prior to issuing any set of k queries, will want to precompute the AOI-privacy that will result from the queries using a mobile device with limited processing power. If processing power were not a concern, the union of circles metric could be used instead (though how to best choose the radius of the circles used in the computation remains an open question).

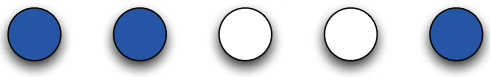
3.5 Choosing the Fake LOIs

Given the metrics necessary to measure how well a set of k queries preserves the privacy of the user's area of interest (AOI) and current location (CL), we now investigate how the user should choose the $k-1$ fake locations of interest (LOIs) to query, given one real LOI. Regardless of how the fake LOIs are chosen, recall our assumption that, from the point of view of the adversary, it is equally probable that any of the k queried nodes is the real LOI. Any implementation of k-anonymity for sensor network queries must take into account real-life limitations. For example, if troops using a military sensor network queried one node at a location of strategic importance and $k-1$ nodes at strategically irrelevant positions, the adversary could guess the real LOI with high probability.

Another situation that could leak the real LOI is if the user issues multiple queries to the real LOI. Any algorithm used to choose the $k-1$ fake LOIs based on the real LOI is required to choose the same $k-1$ fake LOIs each time the user issues a real query. Otherwise, if the user first queries the k nodes in $\mathcal{L}_1$ then later issues queries to the k nodes in $\mathcal{L}_2 \neq \mathcal{L}_1$, and if the adversary can correctly guess that the user was issuing repeat queries to the same node (it is overly optimistic to assume otherwise), the adversary would learn that the real LOI is in $\mathcal{L}_1 \cap \mathcal{L}_2$.

Let F be the choice function that takes the real LOI L and returns a set $\mathcal{L}$ of k LOIs to query, with $L \in \mathcal{L}$. If F is deterministic, repeat queries are of no concern, since $\mathrm{F}(L) = \mathcal{L}$ for each call to $\mathrm{F}(L)$. If F is a random function, taking L and a random seed s_L as parameters and using a cryptographically secure pseudo-random number generator [2] to choose the fake LOIs, the user should encapsulate F in a deterministic function. For example, the user could call $\mathrm{F}'(L) = \mathrm{F}(L, s_L)$, where $s_L = \mathrm{PRF}(s, L)$ is computed using a stored secure seed (i.e., a key), s, and pdeudorandom function PRF. The pseudorandom function could be as simple as a cryptographic hash function, e.g., SHA-2 [5], on the concatenation of s and L, i.e., $\mathrm{PRF}(s, L) = \text{SHA-2-256}(s \circ L)$, with $\circ$ representing concatenation. Alternately, a more robust approach could use a key-derivation function (KDF) as the pseudorandom function [7].

It is not only multiple queries to the real LOI that pose a problem; even a single set of k queries could leak the real LOI. Knowledge of how F works could be sufficient for the adversary to determine the real LOI, given the set $\mathcal{L}$ of all k nodes queried. Consider a



(a) The $k = 3$ sensor nodes that were actually queried by the user.

(b) The nodes that would have been chosen as $\mathcal{L}$ were the westernmost node the real LOI.

Figure 3.7: An example of the σ-maximizing choice function leaking the real LOI.

deterministic F that takes the real LOI L and returns a list $\mathcal{L}$ that contains L and the $k-1$ fake LOIs, such that $\sigma(\mathcal{L}, \mathcal{N})$ is maximized over all possible choices of $k-1$ fake LOIs. This function, which maximizes the user's AOI-privacy, may leak the user's real LOI. Consider the example in Figure 3.7. If the $k = 3$ nodes illustrated in Figure 3.7(a) are queried, the adversary would know that the interior LOI is the real LOI. Were the westernmost node the real LOI, the central node and the easternmost node would have been chosen by F as the fake LOIs, as illustrated in Figure 3.7(b), to maximize σ. Similarly, the easternmost node could not be the real LOI.

For F not to leak information in this fashion, F must generate a set $\mathcal{L} \subseteq \mathcal{N}$ that is closed under F. For a deterministic F, this closure requirement means: if $\mathrm{F}(L) = \mathcal{L}$ for some $L \in \mathcal{N}$, then $\mathrm{F}(L_i) = \mathcal{L}$ for all $L_i \in \mathcal{L}$. The closure requirement for a non-deterministic F is more complex. We assume that the random seed s passed to F is secret; because the seed must remain secret, it must be sufficiently long (e.g., 256 bits) to prevent exhaustive search of the seed space by the adversary. As such, the adversary cannot compute $\mathrm{F}(L_i, s)$ for every $L_i \in \mathcal{L}$. What the adversary can do is estimate probabilities. Define $\mathrm{P}(\mathrm{F}, \mathcal{L}, L_i)$ as the proportion of all seeds in the seed space for which F will generate $\mathcal{L}$, given L_i and a seed as arguments. Given $\mathcal{L}$, the adversary can estimate $\mathrm{P}(\mathrm{F}, \mathcal{L}, L_i)$ for each $L_i \in \mathcal{L}$ by running $\mathrm{F}(L_i, s_{test})$ with a large number of different test seed values. The adversary can then predict that the largest estimate corresponds to the most likely real LOI. That is, if $L_{max} = \arg\max_{L_i \in \mathcal{L}} \{\mathrm{P}(\mathrm{F}, \mathcal{L}, L_i)\}$, then L_{max} has the highest probability of having generated $\mathcal{L}$, and hence is the most likely real LOI. Based on this attack, the closure property necessary for a non-deterministic F is: if $\mathrm{F}(L, s) = \mathcal{L}$ for some $L \in \mathcal{N}$ and seed s, then $\mathrm{P}(\mathrm{F}, \mathcal{L}, L) = \mathrm{P}(\mathrm{F}, \mathcal{L}, L_i)$ for all $L_i \in \mathcal{L}$.

The simplest method of fulfilling the closure requirement is to generate an unbiased random partitioning of the network. When the user wishes to query a real LOI, the fake LOIs also queried are the other nodes in the real LOI's partition. Note that the size of the partitions, while obviously based on the LOI-privacy requirements of the user, may not be able to be uniform. For example, a network of 50 nodes cannot be divided into partitions each of size $k = 3$.

Lemma 3.5. *Any network with $n \geq 3$ nodes can be partitioned such that each partition contains* 3–5 *nodes.*

Proof. By induction. For $n \in [3, 5]$, use a single partition of size n. Assuming you can partition a network of size $n = 3j + k$, where $j \geq 0$, $k \in [3, 5]$, you can partition a network of size $n + 3 = 3(j + 1) + k$ by using an additional partition of size 3. □

Because of Lemma 3.5, we will use partitions of sizes 3-5 whenever the need for partitioning a network arises throughout this dissertation. We further investigate the problem of generating partitions and partitionings on networks in the next section.

3.6 The Partition and Partitioning Problems

The goal of maximizing the minimal distance between nodes in a partition, introduced in Section 3.4.2, as well as the requirement to partition the nodes in a network discussed in Section 3.5, give rise to two optimization problems. We refer to these problems as the partition problem and the partitioning problem.

For this section alone, for simplicity, we will consider the "minimal pairwise distance" definition of σ presented in Section 3.4.2. That is, in both problems, you are given a set of distinct points S in $\mathbb{R}^2$, with $|S| \geq 2$. We define the "goodness" of any subset $T \subseteq S$ as

$$\mathrm{G}(T) = \min \{\mathrm{D}(x, y) \mid a, b \in T, a \neq b\} \ ,$$

where D is Euclidean distance in $\mathbb{R}^2$. In other words, $\mathrm{G}(T)$ is the minimal Euclidean distance between any two points in T. We also extend the notion of goodness to multiple subsets as

$$\mathrm{G}(T_1, \ldots, T_n) = \min \{\mathrm{G}(T_1), \ldots, \mathrm{G}(T_n)\} \ .$$

Or, in other words, the goodness of a set of subsets is only as good as the goodness of the worst subset.

In the partition optimization problem, you are given a set of points S in $\mathbb{R}^2$ and an integer $2 \leq k \leq |S|$. The goal is to find the subset T of S with $|T| = k$ that maximizes $\mathrm{G}(T)$. This problem is equivalent to maximizing the area of interest (AOI) privacy in a single set of queries to k sensor nodes (under the "minimal pairwise distance" definition of σ).

In the partitioning optimization problem, you are given a set of distinct points S in $\mathbb{R}^2$ and two integers $2 \leq k_{min} \leq k_{max} \leq |S|$. The goal is to partition S into subsets $T_1, \ldots, T_m$ with $k_{min} \leq |T_i| \leq k_{max}$ for all i, such that $\mathrm{G}(T_1, \ldots, T_m)$ is maximized. Here, partitioning into subsets has the usual meaning: $T_i \cap T_j = \emptyset$ for all $i \neq j$, and $\cup_i T_i = S$. This problem is equivalent to maximizing the AOI-privacy over all the partitions in the partitioning of a sensor network (again, under the "minimal pairwise distance" definition of σ).

We begin our discussion of these problems in Section 3.6.1 by investigating their hardness. We will follow up that topic with a $\Theta(1)$ approximation for the partition problem in Section 3.6.2, followed by a divide-and-conquer algorithm for effectively solving the partitioning problem in Section 3.6.3.

3.6.1 NP-Completeness

We begin our discussion of the hardness of the partition and partitioning optimization problems by first presenting variants of these two problems on a graph (as opposed to in $\mathbb{R}^2$), phrased as decision problems:

- PARTITION-GRAPH: You are given a graph $G = \langle V, E \rangle$, $k \in \mathbb{Z} \geq 2$, and $d \in \mathbb{Z} \geq 1$. Is there a subset $S \subseteq V$ with $|S| = k$ such that the shortest path between any two elements in S has length at least d?

- PARTITIONING-GRAPH: You are given a graph $G = \langle V, E \rangle$, $k_{min} \in \mathbb{Z}$ and $k_{max} \in \mathbb{Z}$ with $1 \leq k_{min} \leq k_{max} \leq |V|$, and $d \in \mathbb{Z} \geq 1$. Can the vertices of G be partitioned into disjoint sets $T_1, \ldots, T_m$ such that $k_{min} \leq |T_i| \leq k_{max}$ and the shortest path between any two vertices in T_i has length at least d, for all i?

Theorem 3.6. PARTITION-GRAPH *is NP-complete.*

Proof. First, we show that PARTITION-GRAPH is in NP. Consider a candidate solution. The distance between any two vertices in the candidate solution could be established by, e.g., $\mathcal{O}(k)$ breadth-first searches, each running in polynomial time and space. The candidate solution could then be verified by checking the distance between all pairs of vertices in the solution, requiring $\mathcal{O}(k^2)$ distance lookups.

Next, we perform the Turing reduction. Let $\mathrm{A}(G, k, d)$ be an algorithm that solves PARTITION-GRAPH (i.e., returns either TRUE or FALSE). Now, consider the following problem.

- INDEPENDENT-SET: You are given a graph $G = \langle V, E \rangle$, $k \in \mathbb{Z} \leq |V|$. Is there a subset $S \subseteq V$ with $|S| \geq k$ such that no two vertices in S are joined by an edge in E?

INDEPENDENT-SET is known to be NP-complete [36]. Note that an independent set of size k is equivalent to a partition of size k with a distance of at least two hops between all pairs of vertices in the partition. Therefore, an instance $\langle G, k \rangle$ of INDEPENDENT-SET can be solved with a single call to the PARTITION-GRAPH solver, $\mathrm{A}(G, k, 2)$. □

Theorem 3.7. PARTITIONING-GRAPH *is NP-complete.*

Proof. First, we show that PARTITIONING-GRAPH is in NP. Consider a candidate solution. The distance between any two vertices in the graph could be established by, e.g., $\mathcal{O}(|V|)$ breadth-first searches, each running in polynomial time and space. The candidate solution could then be verified by checking the distance between all pairs of vertices in the same partition in the solution, requiring $\mathcal{O}(|V|^2)$ distance lookups.

Next, we perform the Turing reduction. Let $\mathrm{A}(G, k_{min}, k_{max}, d)$ be an algorithm that solves PARTITIONING-GRAPH (i.e., returns either TRUE or FALSE). Now, consider the following problem:

- PARTITION-INTO-TRIANGLES: You are given a graph $G = \langle V, E \rangle$ where $|V| = 3q$ for some $q \in \mathbb{Z}$. Can V be partitioned into disjoint sets $V_1, \ldots, V_q$, where $|V_i| = 3$ and V_i forms a clique, for all i?

PARTITION-INTO-TRIANGLES is known to be NP-complete [52, 14]. Given an instance $\langle G \rangle$ of PARTITION-INTO-TRIANGLES, first construct the complement graph, $G' = \langle V, E' \rangle$, in which two vertices are connected by an edge if and only if they were not connected by an edge in G. Note that V on G can be partitioned into disjoint sets V_i with $|V_i| = 3$ that all form

cliques, if and only if V on G' can be partitioned into disjoint sets V_i with $|V_i| = 3$ that all form independent sets. Having no edges connecting the three nodes is equivalent to them being a partition of size three with a distance of at least two hops between all pairs of vertices in the partition. Therefore, calling the PARTITIONING-GRAPH solver a single time, $\mathrm{A}(G', 3, 3, 2)$, yields the solution to a PARTITION-INTO-TRIANGLES instance $\langle G \rangle$. □

Remark 3.8. The partition optimization problem is known to be NP-hard on $\mathbb{R}^2$ as well as on graphs [63].

Conjecture 3.9. *The partitioning optimization problem is NP-hard on* $\mathbb{R}^2$.

While we do not have a proof that this problem is NP-hard on $\mathbb{R}^2$, we also have no reason to doubt as much. The related problem of partitioning points on $\mathbb{Z}^2$ into a given number of partitions (with no size restrictions) such that the points within any given partition are a minimum distance apart is believed to be NP-hard [34], and another related problem of painting colours onto aircraft to maximize the distance between similar colours is also believed to be hard [51]. A formal proof that the partitioning problem is NP-hard on $\mathbb{R}^2$ remains an open problem.

Remark 3.10. The partition optimization problem can be solved in $\mathcal{O}(n \log n)$ time, where $|\mathcal{S}| = n$, if we restrict the problem to the case of $k = 2$ [31]. The solution is based on performing a Graham scan [20] to find the convex hull of the points $\mathcal{S}$ and determining the two furthest points from each other on the hull [58].

Remark 3.11. The partition optimization problem, for any *fixed* value of k, can be solved in polynomial time. That is because the number of candidate solutions is $\mathcal{O}(n^k)$, where $|\mathcal{S}| = n$, which is polynomial for a fixed k.

3.6.2 Greedy Partition Approximation

We present here a $\Theta(1)$ greedy approximation algorithm to the partition problem, based on private communication with Har-Peled and Sidiropoulos [22]:

1. Pick x_1 to be an arbitrary node in the network; then,
2. Inductively, given $X_i = \{x_1, \ldots, x_i\}$, $i < k$, pick node $x_{i+1} \notin X_i$ such that $\min\left(\mathrm{D}(x_1, x_{i+1}), \ldots, \mathrm{D}(x_i, x_{i+1})\right)$ is maximized.

The algorithm is a simple loop that, in each iteration, considers $\Theta(n)$ nodes to be the next possible node added to the partition — doing so would require only a single distance computation between the node under consideration and the most recent node added to the partition, possibly causing the minimal distance to any node in the partition to be updated for the node under consideration. The loop iterates $k-1$ times, so clearly this algorithm can be implemented in $\mathcal{O}(nk)$ time with $\mathcal{O}(n)$ memory.

This approximation algorithm is similar to one proposed by Ravi et al. [50] for the maximal minimum facility dispersion problem. A version for facility dispersion on continuous graphs was proposed by Tamir [57]. For the convenience of the reader, we have included an original proof that the algorithm in indeed a $\Theta(1)$ approximation — in fact, a 2-approximation — for the partition problem.

Lemma 3.12. *Let $X = \{x_1, \ldots, x_k\}$ be the partition returned by the greedy algorithm, with minimal pairwise distance d'. Let $Y = \{y_1, \ldots, y_k\}$ be an optimal solution to the partition problem. Let $1 < s \leq k$ be the smallest integer such that there exists $t < s$ where $\mathrm{D}(x_s, x_t) = d'$. Then, for all $y \in Y$, there exists an $x_y \in X_{s-1} = \{x_1, \ldots, x_{s-1}\}$ such that $\mathrm{D}(y, x_y) \leq d'$.*

Proof. Let y be an arbitrary node of the optimal solution Y with a distance greater than d' from every node in X_{s-1}. However, the node chosen by the greedy algorithm as x_s has the greatest minimal distance to any node in X_{s-1}, namely d', which is a contradiction. □

Theorem 3.13. *The greedy algorithm is a 2-approximation for the partition problem.*

Proof. Expanding on the notation from Lemma 3.12, let Y be the optimal result to an instance of the partition problem with minimal pairwise distance d. Let X be the result from the greedy algorithm with minimal pairwise distance d', and assume $d' < \frac{d}{2}$.

Define, for all $y \in Y$, $\mathrm{C}_s(y)$ as the single closest node in X_s to y (with ties in distance being broken by the index of the node in X_s). Formally,

$$\mathrm{C}_s(y) = \arg\min_i \{x_i \in X_s \mid \mathrm{D}(x_i, y) \text{ is minimized}\} .$$

By definition, $|X_{s-1}| < k = |Y|$. Therefore, by the pigeon-hole principle, there must exist $y_a, y_b \in Y, a \neq b$, such that $\mathrm{C}_{s-1}(y_a) = \mathrm{C}_{s-1}(y_b)$ (which we denote as x_c).

Since x_c is the closest node in X_{s-1} to y_a, and we know from Lemma 3.12 that there must exist some node $x \in X_{s-1}$ such that $\mathrm{D}(y_a, x) \leq d'$, we know that $\mathrm{D}(y_a, x_c) \leq d'$. Similarly, $\mathrm{D}(y_b, x_c) \leq d'$. Therefore, by the triangle inequality,

$$\mathrm{D}(y_a, y_b) \leq 2 \cdot d' < 2 \cdot \frac{d}{2} = d \,,$$

meaning that the minimal pairwise distance in Y is less than d, which is a contradiction. □

Remark 3.14. The greedy algorithm and its proof in Theorem 3.13 generalize to choosing points to maximize the minimal pairwise distance over any continuous convex space in $\mathbb{R}^2$.

3.6.3 Divide-and-Conquer Partitioning

In this section, we present a divide-and-conquer algorithm for effectively solving the partitioning problem. The idea underlying the algorithm is first to compute the number of nodes that will be assigned to each partition within the partitioning. The algorithm then finds a partition of size k, where k is the largest number of nodes that have to be assigned to any partition (up to a maximum of $k = \mu$). This partition could be found as an optimal partition via brute-force, or via a Monte Carlo algorithm (in our implementation, we use a Monte Carlo algorithm that generates π random partitions and chooses the best one). The nodes in this partition serve as *seeds* that form the basis of subnets of the network. Each node in the network is assigned to the subnet of its closest seed, and the number of nodes to be assigned to each partition are divided evenly among the subnets. The algorithm then continues recursively on each subnet, until the maximal number of nodes to be assigned to each partition in a given subnet is only one — at this point, each node in the subnet is assigned at random to one of the partitions requiring a node from that subnet. The algorithm is formalized in Algorithm 3.2, with the entire process repeating α times and the best partitioning out of all of those being returned on line 14. The integer division that begins the DIVIDE-AND-CONQUER() function is just to divide the number of nodes that will be in each partition as equally as possible.

First, we compared the divide-and-conquer algorithm to a randomized partitioning algorithm and to a brute-force search that finds an optimal partitioning. The randomized algorithm is formalized as Algorithm 3.3. It uses the same integer division technique to

Algorithm 3.2 The divide-and-conquer algorithm for solving the partitioning problem

function DIVIDE-AND-CONQUER($\mathcal{N}, k_{min}, k_{max}, \alpha, \pi, \mu$)
  Preconditions: $1 \leq k_{min} \in \mathbb{Z} \leq k_{max} \in \mathbb{Z} \leq |\mathcal{N}|, \alpha \in \mathbb{Z} \geq 1, \pi \in \mathbb{Z} \geq 1, \mu \in \mathbb{Z} \geq 2$
  $numPartitions \leftarrow |\mathcal{N}|/k_{min}$ ▷ Integer division
  if $numPartitions \cdot k_{max} < |\mathcal{N}|$ **then**
    error Impossible to generate partitions
  $minSize \leftarrow |\mathcal{N}|/numParititons$ ▷ Integer division
  $numLarger \leftarrow |\mathcal{N}| - (minSize \cdot numPartition)$
  $numInPart[i] \leftarrow minSize + 1 \quad \forall\ 0 \leq i < numLarger$ ▷ Arrays indexed from 0
  $numInPart[i] \leftarrow minSize \quad \forall\ numLarger \leq i < numPartitions$
  for $i \leftarrow 0$ **to** $\alpha - 1$ **do**
    Assign N to no partition $\forall\ N \in \mathcal{N}$
    DIVIDE-AND-CONQUER-X($\mathcal{N}, numInPart, numPartitions, \pi, \mu$)
    $p[i] \leftarrow$ the current partitioning
  return the best partitioning out of $p[0], \ldots, p[\alpha - 1]$

procedure DIVIDE-AND-CONQUER-X($\mathcal{N}, numInPart, numPartitions, \pi, \mu$)
  $numSubnets \leftarrow \min(\mu, \max(numInPart[1], \ldots, numInPart[numPartitions]))$
  if $numSubnets = 1$ **then**
    for all partitions P with $numInPart[P] = 1$ **do**
      Assign a random unassigned node in $\mathcal{N}$ to partition P
    return
  $subnet[i] \leftarrow \{\}\ \ \forall\ 0 \leq i < numSubnets$
  $capacity[i] \leftarrow 0\ \ \forall\ 0 \leq i < numSubnets$
  $nDist[i][j] \leftarrow 0\ \ \forall\ 0 \leq i < numSubnets, 0 \leq j < numPartitions$
  $seeds \leftarrow$ MONTE-CARLO-PARTITION($\mathcal{N}, numSubnets, \pi$)
  for all $N \in \mathcal{N}$ **do**
    if $\exists\ i$ such that $seeds[i] = N$ **then**
      $index \leftarrow i$
    else
      $d \leftarrow$ the minimal distance from N to any element in $seeds$
      $indices \leftarrow \{i \mid \mathrm{D}(N, seeds[i]) = d\}$
      $index \leftarrow$ a uniform random element of $indices$
    Add N to $subnet[index]$
    $capacity[index] \leftarrow capacity[index] + 1$
  $onSubnet \leftarrow 0$
  for $onPartition \leftarrow 0$ **to** $numPartitions - 1$ **do**
    for $i \leftarrow 1$ **to** $numInPart[onPartition]$ **do**
      while $capacity[onSubnet] = 0$ **do**
        $onSubnet \leftarrow (onSubnet + 1) \mod numSubnets$
      $nDist[onSubnet][onPartition] \leftarrow nDist[onSubnet][onPartition] + 1$
      $capacity[onSubnet] \leftarrow capacity[onSubnet] - 1$
      $onSubnet \leftarrow (onSubnet + 1) \mod numSubnets$
  for $i \leftarrow 0$ **to** $numSubnets - 1$ **do**
    DIVIDE-AND-CONQUER-X($subnet[i], nDist[i], numPartitions, \pi, \mu$)

function MONTE-CARLO-PARTITION($\mathcal{N}, k, \pi$)
  return the best partition of size k on $\mathcal{N}$, from π randomly generated partitions

equally divide the partition sizes as the divide-and-conquer algorithm, starting on line 3; however, once the partition sizes are equally distributed, it assigns random unassigned nodes to each partition to fill that partition, on line 18. The entire process of creating a partitioning randomly is repeated α times, as with the divide-and-conquer algorithm, and the best partitioning is returned. For the randomized algorithm, we ran 1000 trials using $\alpha = 1000$. The same number of trials were used for the divide-and-conquer algorithm, with $\alpha = 1000$, $\pi = 10$, and $\mu = 3$. Fewer trials were needed for the brute-force approach; we ran 25 trials just to get an accurate timing. The experiment was performed on a 9x5 grid of sensor nodes, and the results are summarized in Figure 3.8. The running times, measured on a 2.93 GHz Core i7 with 8 GB of RAM running OS X 10.8.4, illustrate that the divide-and-conquer partitioning algorithm is orders of magnitude faster than a brute-force search. While it is slower than a randomized approach, the divide-and-conquer algorithm also returned significantly better results than the random algorithm.

Algorithm 3.3 The random algorithm for solving the partitioning problem

```
1:  function RANDOM-PARTITIONING(𝒩, k_min, k_max, α)
2:      Preconditions: 1 ≤ k_min ∈ ℤ ≤ k_max ∈ ℤ ≤ |𝒩|, α ∈ ℤ ≥ 1
3:      numPartitions ← |𝒩|/k_min                          ▷ Integer division
4:      if numPartitions · k_max < |𝒩| then
5:          error Impossible to generate partitions
6:      minSize ← |𝒩|/numParititons                        ▷ Integer division
7:      numLarger ← |𝒩| − (minSize · numPartition)
8:      numInPart[i] ← minSize + 1   ∀ 0 ≤ i < numLarger   ▷ Arrays indexed from 0
9:      numInPart[i] ← minSize   ∀ numLarger ≤ i < numPartitions
10:     for i ← 1 to α do
11:         Assign N to no partition ∀ N ∈ 𝒩
12:         RANDOM-PARTITIONING-X(𝒩, numInPart, numPartitions)
13:         p[i] ← the current partitioning
14:     return the best partitioning out of p[0], ..., p[α − 1]

15: procedure RANDOM-PARTITIONING-X(𝒩, numInPart, numPartitions)
16:     for onPartition ← 0 to numPartitions − 1 do
17:         for i ← 1 to numInPart[onPartition] do
18:             Assign a random unassigned node in 𝒩 to partition onPartition
```

The effect of the variables α (the number of Monte Carlo attempts run for the entire partitioning algorithm) and π (the number of partitions generated each time seeds are needed) is highlighted in Figure 3.9. We modified the values of α and π, while running the divide-and-conquer partitioning algorithm on the same 9x5 grid. For each $\langle \alpha, \pi \rangle$ pair, we ran 1000 trials. Unsurprisingly, increasing the values of α and π increases the average quality of

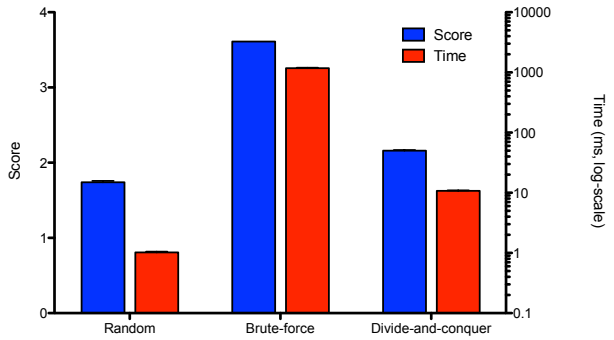


Figure 3.8: A comparison of the scores of the partitioning computed by three different partitioning algorithms, and the running time of each

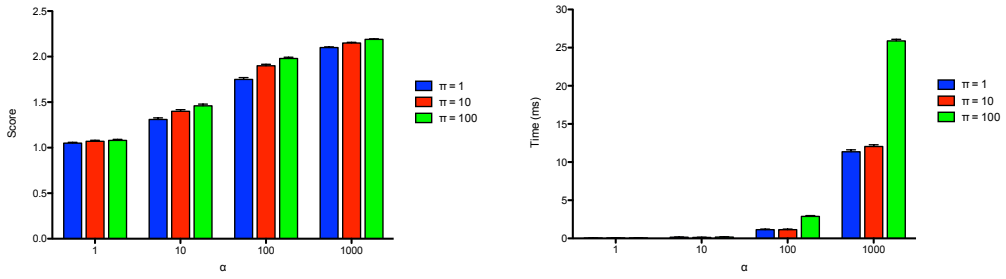


Figure 3.9: The effect of the α and π parameters on the score and running time of the divide-and-conquer partitioning algorithm on a grid topology

the partitioning that is chosen in the end. However, as evidenced by the running time graph, increasing π to large values can increase the running time significantly, for comparably little payoff in the average quality of the partitioning found (relative to increasing α).

We confirmed these results by re-running the experiment, but generating a new random topology for each trial. Each random topology was generated by placing nodes uniformly at random in a 10x10 area, measured in an arbitrary unit of length. To account for the variance in topology, we increased the number of trials for each $\langle \alpha, \pi \rangle$ pair to 5000. The results on the random topology, illustrated in Figure 3.10, are comparable to those seen on the grid topology. Based on these results, we recommend using a value of $\pi = 10$, with the α value based on both the size of the network and the necessity of a higher-scoring result (we chose to use $\alpha = 1000$ for the remainder of this section).

The μ parameter is simply a safeguard for when the individual partitions in a partitioning contain a large number of nodes. If this were the case, the size of the partition used as a seed would be very large. If the seed partition is sufficiently large, relative to the size of

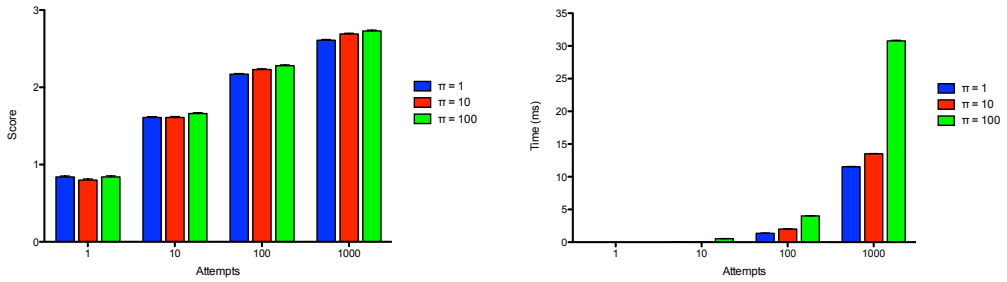


Figure 3.10: The effect of the α and π parameters on the score and running time of the divide-and-conquer partitioning algorithm on a randomized topology

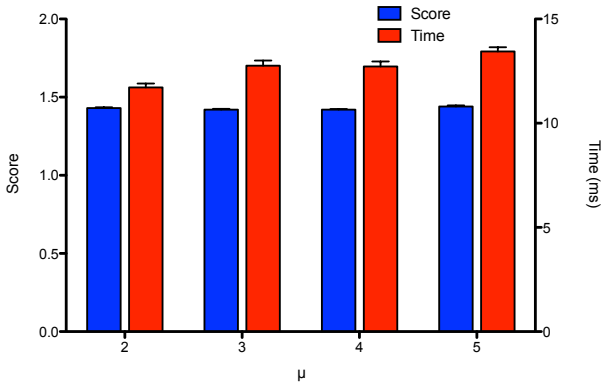


Figure 3.11: The effect of μ, the maximal partition seed size, on the final score and running time of the divide-and-conquer partitioning algorithm

the network (i.e., if there are a small number of large partitions), the divide-and-conquer algorithm would nearly degenerate into a randomized partitioning. In most cases, the value of μ can be safely ignored (i.e., set $\mu = \infty$). As illustrated in Figure 3.11, the μ parameter had no significant effect on the final score, and the individual trial run times are so variable that μ had no overall significant effect on the running time of the algorithm. We recommend using $\mu = 3$ for safety's sake; however, if the usage case precludes large individual partitions, the μ parameter can be entirely ignored.

3.7 Experimental Results in a Uniform Environment

We begin our experiments on the k-anonymity scheme in a featureless environment, devoid of any statistics learned from observing human mobility. The network is a 10x10 grid of nodes placed unit-distance apart, and $\mathrm{P_G}\left(C = N\right) = \mathrm{P_G}\left(L = M \mid C = N\right) = \frac{1}{100}$ for all

nodes N and M in the network. The user's current location is chosen proportional to $\mathrm{P_G}\left(C = N\right)$ (i.e., uniformly at random, for now), the user's real location of interest to be queried is chosen proportional to $\mathrm{P_G}\left(L = M \mid C = N\right)$ (i.e., also uniformly at random for the moment), and the $k-1$ fake queried locations are always chosen uniformly at random.

Remark 3.15. Since each trial of an experiment will observe the effect on the user's CL-privacy and LOI-privacy from a single set of k queries being issued, we do not need to generate a partitioning of the network in the implementation of the experiment. However, it should be noted that this behaviour is equivalent to generating a uniform random partitioning, then querying a partition of size k. Because we wish to observe correlations between, e.g., AOI-privacy and CL-privacy, using a random partitioning scheme (as opposed to the partitioning scheme discussed in Section 3.6.3) is preferred because it gives us the widest range of results.

In our first experiment, message headers between sensor nodes are encrypted using a global key that has been compromised by the adversary. The adversary has compromised 10 of the nodes in the network, and is using them to eavesdrop on communications. The user issues $k = 3$ queries into the network from their current location, and the adversary attempts to determine the user's current location and real location of interest.

Figure 3.12 plots the user's resulting CL-privacy and LOI-privacy from issuing a single set of k queries — each point on the scatterplot represents one of the 1000 trials performed. Unsurprisingly, there is a significant correlation ($P < 0.0001$) between the user's resulting LOI-privacy and CL-privacy, in that high CL-privacy is essential to achieving high LOI-privacy. After all, the more information the adversary gains from overhearing some or all of a set of k queries, the better able the adversary is to determine both the user's current location and location of interest. Interestingly, LOI-privacy in the featureless environment is essentially an "all or nothing" measure, in that the user achieves near-perfect LOI-privacy, i.e., approximately

$$\frac{|\mathcal{N}_{honest}| - 1}{|\mathcal{N}_{honest}|} = \frac{89}{90}\,,$$

or near-minimal LOI-privacy, i.e., approximately

$$\frac{k-1}{k} = \frac{2}{3}\,.$$

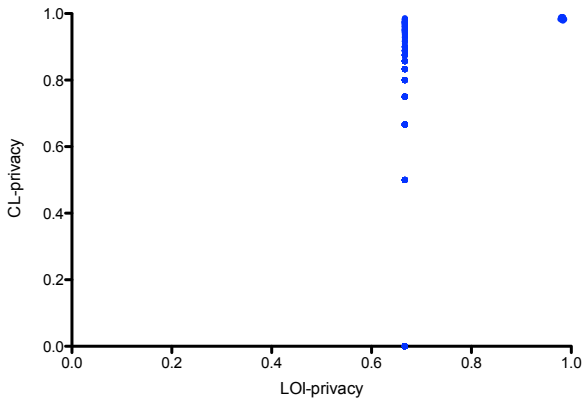


Figure 3.12: The user's resulting CL-privacy and LOI-privacy over 1000 trials with $k = 3$

However, the scatterplot is somewhat misleading: for example, the cluster of points in the upper-right of the graph do not all have LOI-privacy of $\frac{89}{90}$; many have LOI-privacy slightly below that perfect measure. That is, the adversary was always able to deduce, even in tiny measure, locations that could not have been queried. However, in the featureless environment, there is no information for the adversary to work with regarding location popularities, significantly handicapping the adversary who is unable to overhear or deduce all of the queried locations.

Of more interest to a user, however, might be what changes they can make in order to increase the privacy of their current location and location of interest. The most obvious change that the user can make is to adjust k, the number of queries that they send. We repeated the previous experiment, this time adjusting $k \in [2, 5]$. As illustrated in Figure 3.13, lower values of k resulted in significantly higher CL-privacy, as the adversary has fewer query message with which to "triangulate" the user. On the other hand, higher values of k resulted in significantly higher LOI-privacy, as it becomes less likely that the adversary will guess the correct queried location as the user's real location of interest. Each increment of k resulted in a significant change to both privacy metrics ($P < 0.05$), as determined by a one-way ANOVA test with Bonferroni's multiple comparison test [53] on each pair of incremental k values (i.e., from $k = 2$ to $k = 3$, from $k = 3$ to $k = 4$, and from $k = 4$ to $k = 5$), thus indicating a clear trade-off between the user's ability to protect the privacy of their current location and the privacy of their location of interest.

Another parameter within the user's control is their AOI-privacy, which is determined

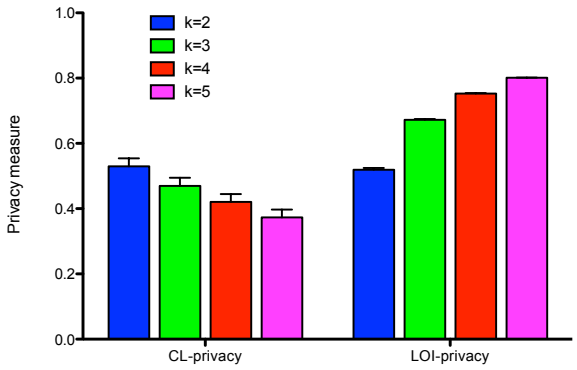


Figure 3.13: The user's resulting CL-privacy and LOI-privacy, as the number of queries k is varied, over 1000 trials

by how well-dispersed their k queries are physically around the network. Figure 3.14 shows the user's CL- and LOI-privacy, relative to their chosen AOI-privacy over 1000 trials. While there is a significant correlation ($P < 0.0001$) between between the user's AOI-privacy and their LOI-privacy, it is not large ($R^2 = 0.0158$). While that may seem surprising, given that the two are conceptually similar measures of how well protected the identity of the user's real query destination is, that lack of large correlation just indicates that how physically spread out the k destinations are is not the most important factor in helping the adversary narrow down the real location of interest. There is also no large correlation ($P < 0.01$, $R^2 < 0.01$) between a user's AOI-privacy and their CL-privacy. Intuitively, one might think that the more dispersed the k queries are, the more opportunity there would be for the adversary to overhear queries and triangulate the user's location. While that is probabilistically true, as indicated by the low P value, it is not a largely meaningful factor to the adversary, as indicated by the low R^2 value.

To confirm these results, we studied a metric related to AOI-privacy, more closely related to the number of opportunities the adversary will have to intercept user queries. That metric is the number of unique hops the user's k queries take (i.e., the number of query message transmissions between unique pairs of sensor nodes), which the user is able to control based on the chosen nodes to query. Figure 3.15 shows the user's CL- and LOI-privacy, relative to the number of distinct hops taken by their query messages over 1000 trials. Again, the number of distinct hops taken by the user's queries is significantly negatively correlated with both the user's resulting CL-privacy ($P = 0.0097$) and their resulting LOI-privacy

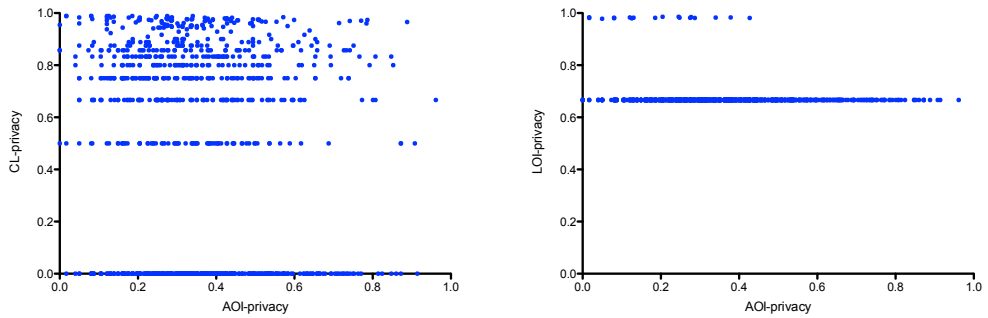


Figure 3.14: The user's resulting CL-privacy and LOI-privacy, as the user's AOI-privacy is varied, over 1000 trials with $k = 3$

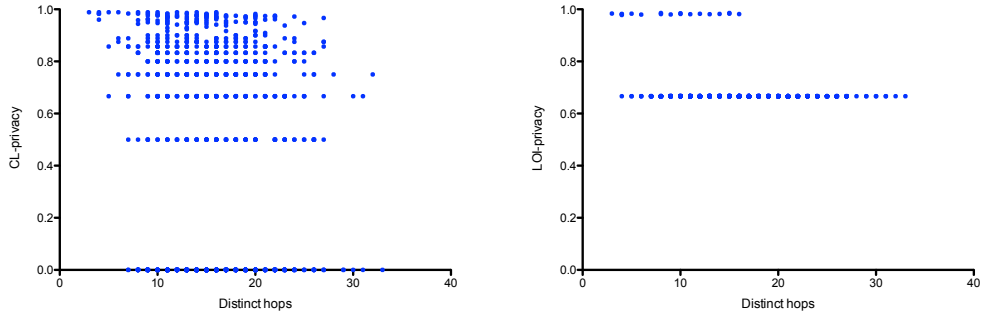


Figure 3.15: The user's resulting CL-privacy and LOI-privacy, as the number of distinct hops in the queries issued by the user is varied, over 1000 trials with $k = 3$

($P < 0.0001$). However, once again, the R^2 values are low: $R^2 = 0.0067$ and $R^2 = 0.0318$ respectively. In other words, the user's control over how dispersed their queries are will, probabilistically, influence their CL- and LOI-privacy, but blind luck based on query routing near compromised nodes is a much more important factor.

There are numerous other factors, potentially beyond the control of users of the network, that affect the adversary's success as well. One critical factor for the adversary is how many nodes are compromised in the network. The more nodes that are compromised, the more likely it is that the adversary will be able to intercept queries, learning the locations queried and facilitating the computation of the user's current location. This result is depicted in Figure 3.16, showing the predictable result that increasing the number of nodes compromised by the adversary decreases both the user's CL- and LOI-privacy. Note, however, that Figure 3.16 illustrates the same trend first illustrated in Figure 3.13: increasing the privacy constant k — a constant that is entirely under the control of the user — decreases the user's CL-privacy but increases the user's LOI-privacy, illustrating the trade-off between a user

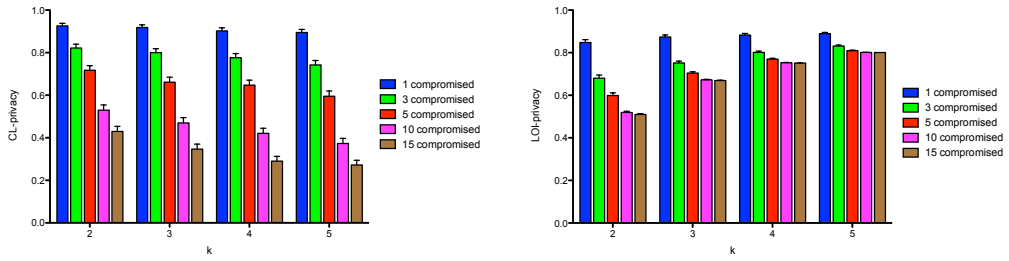


Figure 3.16: The user's resulting CL-privacy and LOI-privacy, as the number of nodes compromised by the adversary varies, over 1000 trials

protecting the privacy of their current location and that of their location of interest.

A property that may or may not be under the control of the user is the type of encryption used in communication between sensors. Recall that the query proper can be encrypted end-to-end between the user and the queried node using any number of available cryptographic techniques. The query itself is not at issue. The issue at hand is what information the adversary can glean from decrypting the information contained in the headers of overheard messages. So far, we have assumed that transmissions between the nodes in the sensor network are encrypted using a global key. Should any node be compromised, the adversary learns the global key; then, if any query transmission is overheard, the adversary learns the destination of that query, as well as two nodes along the path to that destination (the transmitter and the intended recipient of the transmission). However, it is possible that transmissions between sensor nodes are encrypted using link keys — that is, every pair of sensor nodes in transmission range of each other share a unique symmetric key. In this case, an overheard query per se reveals to the adversary only a single node — the transmitter — in the path from the user's current location to the queried location (we must assume that the adversary can determine which node transmitted the query based on metrics such as signal strength, timing, etc. [3]). It is only if a compromised node is the recipient of the transmission that the adversary learns the ultimate destination of the query (and, obviously, the recipient of the transmission). A comparison of the user's CL- and LOI-privacy based on whether global keys or link keys are used is illustrated in Figure 3.17; for the purposes of comparison, an environment in which there is no broadcast transmission (e.g., optical or wired communication [68]), and thus the adversary learns nothing unless a compromised node is the recipient or transmitter of a message, is also included. Each increase in "broadcast security" resulted in a significant increase ($P < 0.05$) to the user's

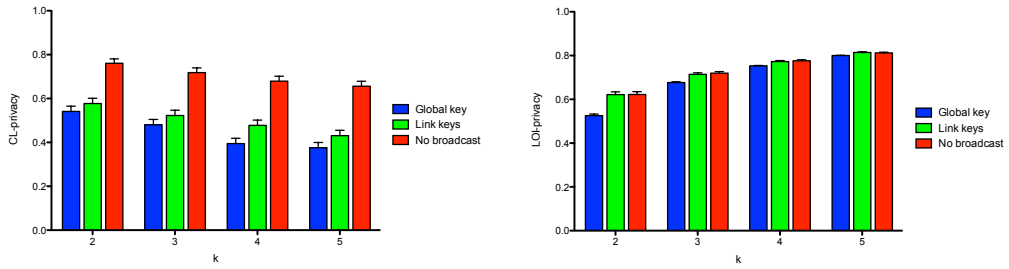


Figure 3.17: The user's resulting CL-privacy and LOI-privacy, as the type of broadcast encryption varies, over 1000 trials

CL-privacy, as determined by a one-way ANOVA test with Bonferroni's multiple comparison test on each incremental pair (i.e., from a global key to link keys, and from link keys to a non-broadcast environment). The first increment illustrates the value to the adversary of being able to learn more nodes along the communication path and the ultimate destination of the communication, for the purposes of inferring from where the initial query could have come. The second increment illustrates the ultimate value to the adversary of overhearing anything at all. On the other hand, the first increment in "broadcast security" resulted in a significant increase to the user's LOI-privacy ($P < 0.05$), but the move from link keys to a non-broadcast environment did not ($P > 0.10$). In other words, being able to decipher the ultimate destination of an overheard query that a compromised node is not routing is helpful to an adversary in determining the user's location of interest, but an overheard query where the adversary cannot decrypt the final destination is practically meaningless towards that goal.

Similar to the type of encryption used in the sensor network, the type of routing used also may or may not be under the control of the user. Here we compare the two types of routing described in Section 3.4.1: fixed shortest-path and random shortest-path routing. In both cases, a query sent from the user's current location to the query destination will arrive in the fewest number of hops possible; however, in fixed shortest-path routing, a query sent between those two locations will always take the same path, whereas in random shortest path routing it could potentially take many different paths. The results are illustrated in Figure 3.18. Switching to random shortest-path routing significantly improved the user's CL-privacy, as the adversary could not ascertain the origin of a query as easily from an overheard message. Obviously, when using a global encryption key, the type of routing used had no effect on

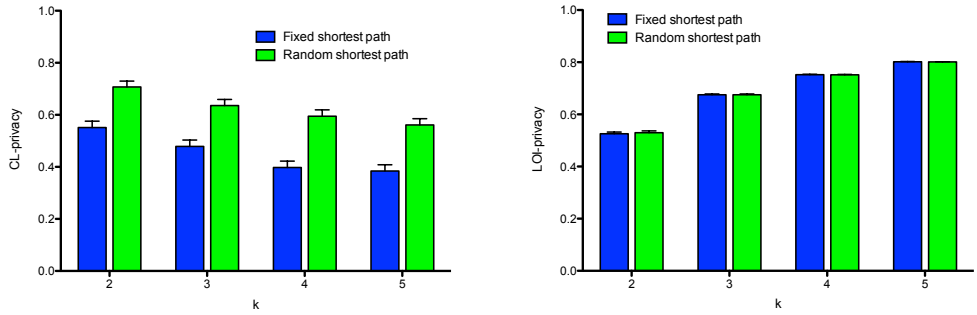


Figure 3.18: The user's resulting CL-privacy and LOI-privacy, as the routing algorithm used in the sensor network varies, over 1000 trials with a global key

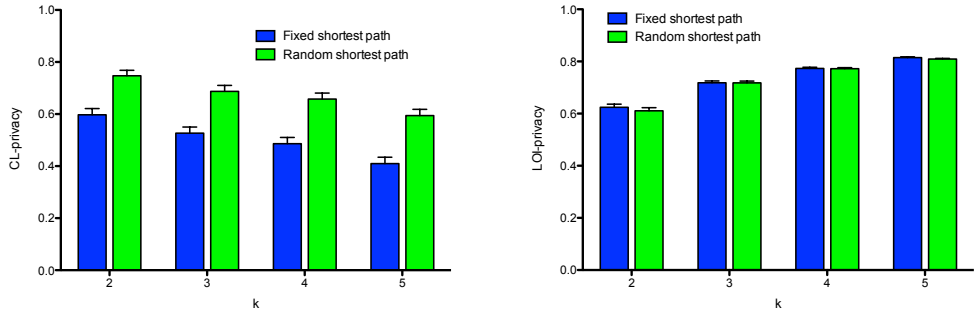


Figure 3.19: The user's resulting CL-privacy and LOI-privacy, as the routing algorithm used in the sensor network varies, over 1000 trials with link keys

the user's LOI-privacy. However, the question remains: if link keys were used instead of a global key, creating a situation in which the adversary could overhear query messages but not learn their destination, would implementing random shortest-path routing increase the user's LOI-privacy by making it more difficult for the adversary to narrow down possible destinations for overheard messages? We repeated the previous experiment, this time using link keys; the results are presented in Figure 3.19. Again, there was no distinction between the LOI-privacy results whether fixed shortest-path or random shortest-path routing were used — in essence, being able to easily guess potential destinations for overheard but non-decrypted packets is not highly important to the adversary.

3.8 Experimental Results with Mobility

Until now, we have run experiments in a uniform, featureless environment, in which $\mathrm{P_G}\left(C = N\right) = \mathrm{P_G}\left(L = M \mid C = N\right) = \frac{1}{|\mathcal{N}|}$ for all nodes N and M in the network. How-

ever, for a real sensor network in an urban environment, some sensor nodes will be located at more population locations than other nodes.

To generate such a rich environment, we returned to DBS3 and the maps of Edmonton and Fira first presented in Chapter 2. We placed sensor nodes centred on both maps in a grid pattern with 100 m spacing, yielding 112 nodes on the Edmonton map and 84 nodes on the Fira map. On both maps, we initialized a 10-million-agent simulation, and computed the closest node to each agent. We then set

$$\mathrm{P_G}\left(C = N\right) = \frac{\nu_0(N)}{10000000}$$

for each $N \in \mathcal{N}$, where $\nu_0(N)$ is the number of agents for whom N is the closest node at time zero. We also set $\mathrm{P_G}\left(L = M \mid C = N\right) = \mathrm{P_G}\left(C = M\right)$ for each $M \in \mathcal{N}$. In other words, the probability that a node is either the current location or the location of interest is proportional to the amount of pedestrian traffic near it. As before, the user's current location is chosen proportional to $\mathrm{P_G}\left(C = N\right)$, the user's real location of interest to be queried is chosen proportional to $\mathrm{P_G}\left(L = M \mid C = N\right)$, and the $k - 1$ fake queried locations are chosen uniformly at random (as previously mentioned in Remark 3.15, this behaviour is equivalent to a random partitioning scheme).

To determine how beneficial this statistical data is to an adversary, we compared the statistically rich environments of Edmonton and Fira to otherwise equivalent environments in which $\mathrm{P_G}\left(C = N\right) = \mathrm{P_G}\left(L = M \mid C = N\right) = \frac{1}{|\mathcal{N}|}$ for all nodes $N, M \in \mathcal{N}$. As in the previous section, message headers between sensor nodes are encrypted using a global key that has been compromised by the adversary. The adversary has compromised 10 of the nodes in the network, and is using them to eavesdrop on communications. The user issues queries into the network from their current location, and the adversary attempts to determine the user's current location and real location of interest. Because of the higher variance LOI-privacy results in the statistically rich environment, we increased the number of trials from 1000 (used in the previous section) to 5000.

The results of this experiment are illustrated for the Edmonton map in Figure 3.20 and for the Fira map in Figure 3.21. As evident from those graphs, both the user's CL- and LOI-privacy fell considerably when this additional statistical data was available to the adversary. Notably, LOI-privacy was more seriously affected than CL-privacy by this

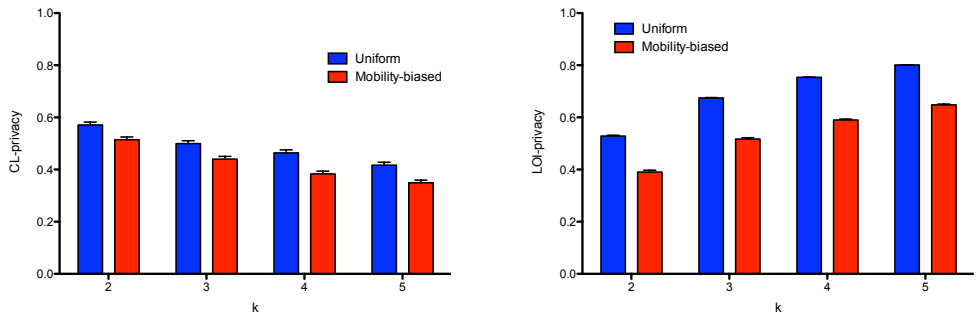


Figure 3.20: The user's resulting CL-privacy and LOI-privacy, in the statistically rich versus uniform environments, over 5000 trials in Edmonton

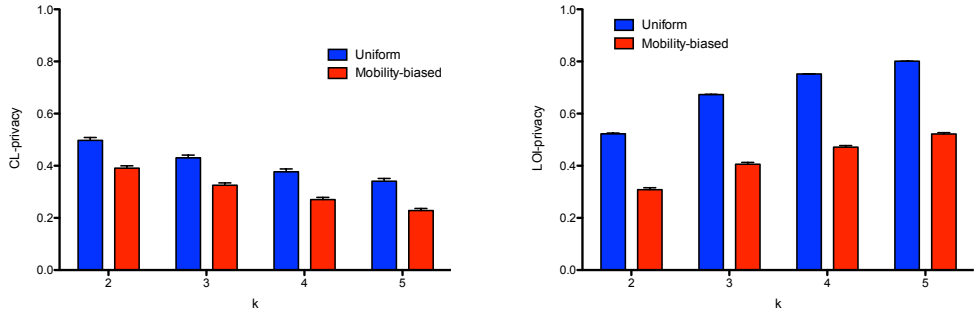


Figure 3.21: The user's resulting CL-privacy and LOI-privacy, in the statistically rich versus uniform environments, over 5000 trials in Fira

additional information, because the random partitioning scheme potentially resulted in low-probability LOIs being grouped with the user's real LOI. Also, note that both privacy measures were affected more seriously in Fira, which is more complex from a Space Syntax standpoint, and has more popular central thoroughfares.

Another situation that could possibly benefit the adversary is if the adversary knows that the user is walking from their current location to their location of interest in the near future. In this way, the user's current location would affect the probabilities that the various nodes in the network are the user's real location of interest, i.e., $\mathrm{P_G}\left(L = M \mid C = N\right)$ for $N, M \in \mathcal{N}$. To generate this even richer world, we advanced the 10-million-agent simulation to some time $t \geq 0$. Keeping

$$\mathrm{P_G}\left(C = N\right) = \frac{\nu_0(N)}{10000000} \ ,$$

we now set

$$\mathrm{P_G}\left(L = M \mid C = N\right) = \frac{\nu_t(N, M)}{\nu_0(N)}$$

for each $N, M \in \mathcal{N}$, where $\nu_t(N, M)$ is the number of agents for whom N is the closest node at time zero and M is the closest node at time t.

To motivate that this additional short-time-horizon information could assist the adversary, consider an adversary who, in the absence of any network communication, wants to predict the user's location at some time $t \geq 0$. Lacking any information about the user to utilize, the adversary would just guess the most popular location on the map is the user's location, guessing successfully with probability $\max_{N \in \mathcal{N}} \mathrm{P_G}\left(C = N\right)$. Compare that adversary to one who wants to predict the user's location at time $t \geq 0$, but also has access to an oracle that reveals the user's location at time $t = 0$. Such an adversary could successfully predict that user's location with probability

$$\sum_{N \in \mathcal{N}} \left[\mathrm{P_G}\left(C = N\right) \cdot \max_{M \in \mathcal{N}} \mathrm{P_G}\left(L = M \mid C = N\right) \right].$$

These two adversaries are compared on the Edmonton map in Figure 3.22 and on the Fira map in Figure 3.23. The more complex Fira map with few thoroughfares leads to a higher baseline success rate for the adversary without an oracle than in Edmonton, and the natural inclination of pedestrians to move towards and stay on those thoroughfares results in the oracle-possessing adversary's success rate declining towards the baseline more quickly than in Edmonton. However, in either case, the adversary with the oracle has a clear advantage over the adversary without the oracle up to a time threshold of $t \approx 10$–15 (where t is measured in minutes).

To further investigate this concept of a time threshold, we reran the previous k-query experiment on the Edmonton and Fira maps. This time, however, we assumed that the user is travelling from their current location to their location of interest — i.e., we used the richer definition of $\mathrm{P_G}\left(L = M \mid C = N\right)$, for some chosen time threshold t between when the user sends their k queries and when they arrive at their real location of interest. The resulting CL- and LOI-privacy for the user in Edmonton, over varying time thresholds, is illustrated in Figure 3.24; the results of the same experiment in Fira are in Figure 3.25.

Counter-intuitively, for small t values, the user's CL-privacy is higher than at larger

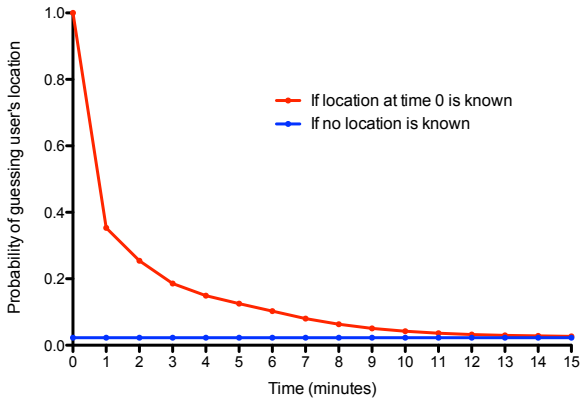


Figure 3.22: The success rate of an adversary predicting a user's location at time $t \geq 0$ with and without an oracle revealing the user's location at time $t = 0$, in Edmonton

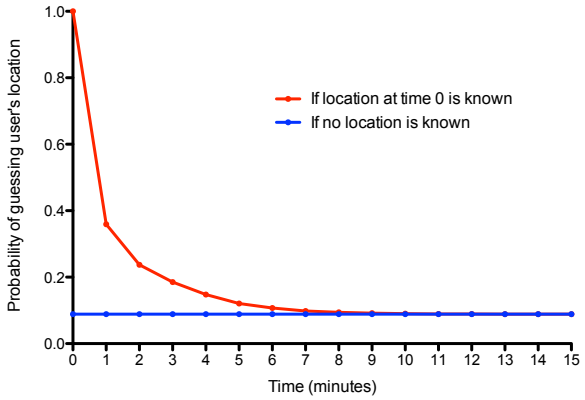


Figure 3.23: The success rate of an adversary predicting a user's location at time $t \geq 0$ with and without an oracle revealing the user's location at time $t = 0$, in Fira

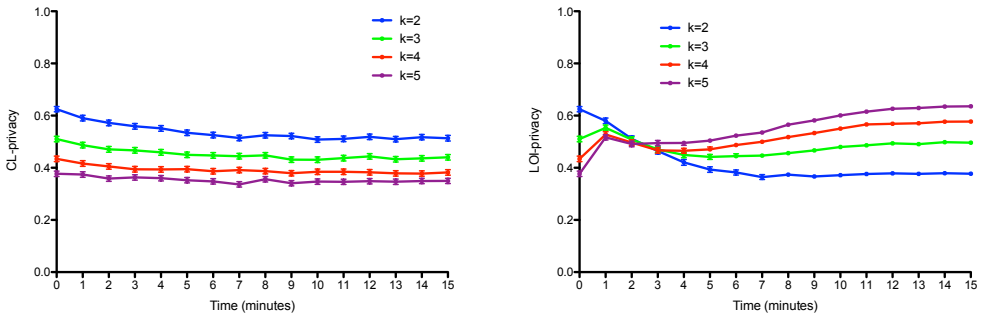


Figure 3.24: The user's resulting CL-privacy and LOI-privacy, when the user queries their location of interest at varying times before arriving at it, over 5000 trials in Edmonton

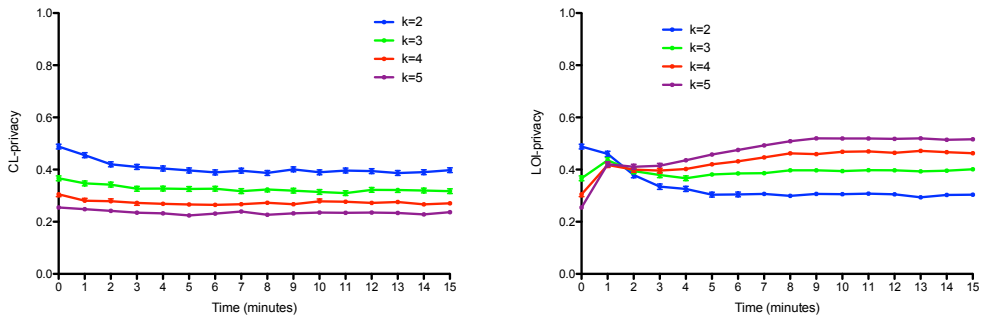


Figure 3.25: The user's resulting CL-privacy and LOI-privacy, when the user queries their location of interest at varying times before arriving at it, over 5000 trials in Fira

t values. Consider the case where, for $t = 0$, one of the queried locations is guaranteed to be the user's current location, effectively decrementing k from the point of view of the adversary's ability to intercept the user's messages. Assuming that the user's CL-privacy has plateaued by $t = 15$ (based on the oracle experiment), we performed a one-way ANOVA test with Bonferroni's multiple comparison test between the user's CL-privacy at $t = 15$ and each previous time (i.e., comparing $t = 0$ to $t = 15$, $t = 1$ to $t = 15$, etc.). In Edmonton using $k = 3$, there was no significant difference ($P > 0.05$) in the CL-privacy at any time $4 \leq t \leq 14$ and the CL-privacy at $t = 15$. Similarly, in Fira using $k = 3$, there was no significant difference ($P > 0.05$) in the CL-privacy at time $3 \leq t \leq 14$ and the CL-privacy at $t = 15$. Based on these results, we conclude that any benefit to the user's CL-privacy from a small t threshold from query to arrival has vanished by 4 and 3 minutes respectively on the Edmonton and Fira maps when $k = 3$.

Regarding the LOI-privacy results, recall from Section 3.7 the fundamental trade-off between the privacy of the user's current location and that of their location of interest: as k increases, the user's LOI-privacy increases at the cost of their CL-privacy. However, recall Lemma 3.2, which states that when the adversary knows the user's current location is their real location of interest (e.g., in this experiment when $t = 0$), the resulting CL- and LOI-privacies are identical. This apparent paradox is visible in the LOI-privacy graphs wherein the lines representing the various k values cross as t increases, before plateauing. These graphs tell us that, when the adversary knows that the user's real LOI is at or near their current location, the LOI-privacy metric is really measuring the privacy of the user's current location. Similar to the ANOVA and Bonferroni results on CL-privacy, there was

no significant difference ($P > 0.05$) between the user's LOI-privacy at any time $10 \leq t \leq 14$ and at time $t = 15$ in Edmonton using $k = 3$; the same holds true in Fira using $k = 3$ for any time $6 \leq t \leq 14$.

From these results, we can conclude that if a user is travelling from their current location to their real location of interest, they should send their k queries long enough in advance of embarking to mitigate any advantage the adversary has in predicting the user's real LOI. How long is "long enough" will depend on the map in question and the number of queries sent. However, "long enough" in our experiments was a surprisingly short period of time, e.g., a mere $t = 10$ minutes when using $k = 3$ on the Edmonton map.

3.9 Conclusions

In this chapter, we presented a user-controlled methodology for preserving the privacy of the user's location of interest in a wireless sensor network: issuing k queries instead of just one. This methodology does not require any special security measures in the sensor network itself (as opposed to, e.g., onion routing). There is a direct trade-off between preserving the privacy of the user's location of interest and the privacy of their current location, controlled by the user's choice of the security constant k, i.e., the number of queries sent by the user.

The presence of user mobility data, reflecting the popularity of different locations, gives the adversary a significant advantage in determining a user's current location and location of interest, relative to an adversary operating in a featureless, uniform environment. It is even more difficult for users moving from their current location to their location of interest to preserve their privacy; however, those users can mitigate this advantage of the adversary by leaving a long enough time between their query and their arrival at the location of interest.

Future work on this scheme should seek network partitioning schemes that yield guarantees of area-of-interest privacy and/or location-of-interest privacy while still satisfying the closure requirement. In particular, satisfying the closure condition in the presence of unequal location popularities may be a substantial challenge.

Chapter 4

Conclusions

We began this dissertation by presenting a novel and highly accurate generative mobility model for outdoor urban environments. This generative model, based on the concept of Space Syntax, is unlike existing realistic generative mobility models in that it takes only an unannotated map as input. No map tagging or labelling is required, because agents choose their destinations based on the physical features of the map itself — specifically, based on centrality bias and distance decay. The new MEA* search algorithm introduced as part of our implementation of this model, the publicly available Destination-Based Space Syntax Simulator (DBS3), allows for fast and realistic runtime pathfinding. Not only do agents not walk along fixed tracks, but they also choose paths based on how people actually perceive distance: in terms of the number of changes in direction or the magnitude in change of direction, as opposed to true Euclidean distance.

The lynchpin experimental result in the chapter on DBS3 was that setting the tuning exponents for centrality bias and distance decay both to 1 yielded agent densities that were highly correlated ($R^2 = 0.96096$) to pedestrian densities in downtown Edmonton. The only significant "mistake" made by DBS3 was that it was unable to predict seasonal variation in pedestrian density around a downtown university campus. DBS3 predicted that the area would be highly populated with pedestrian traffic, but was unable to account for our observations of actual pedestrian traffic in Edmonton being taken at a time when the university was between sessions.

In the second portion of this dissertation, we discussed a method of preserving the privacy of a user's location of interest in a wireless sensor network: querying k locations — one the

user's real location of interest, and $k-1$ fake locations. This method is controlled and executed by the user, and does not require any special security measures implemented in the sensor network. There is a direct trade-off, controlled strictly by the user's choice of k, between protecting the user's location-of-interest privacy and their current-location privacy.

Operating in a non-uniform environment, i.e., one in which the popularity of the different locations that can be queried is non-uniform (e.g., as per DBS3's mobility model), gives a significant advantage to an adversary who is attempting to determine the user's current location and real location of interest. Users moving from their current location to their location of interest in such an environment are at an even greater risk from this adversary. One solution for those users is a behavioural solution rather than a technical one: leaving a long enough window of time between querying their location of interest and arriving at it.

There are two key avenues of future work, one from each half of this dissertation. Fine tuning the centrality bias and distance decay exponents in DBS3 across a wide range of configured spaces (e.g., other cities and other map sizes) is important to better understand if and when the tuning parameters to DBS3 need to be adjusted. For the k-query scheme, devising new network partitioning schemes that provide guaranteed privacy bounds for the user's area-of-interest or location-of-interest privacy (while still satisfying the requisite closure security requirement) could serve to make the scheme even more useful in hostile environments.

Bibliography

[1] Christer Andersson, Leonardo A. Martucci, and Simone Fischer-Hübner. Requirements for privacy-enhancements in mobile ad hoc networks. In *Proceedings of the 3rd German Workshop on Mobile Ad-Hoc Networks*, 2005.

[2] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[3] Mohamed Salah Bouassida, Gilles Guette, Mohamed Shawky, and Bertrand Ducourthial. Sybil nodes detection based on received signal strength variations within vanet. *International Journal of Network Security*, 9(1):22–33, July 2009.

[4] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[5] John Bryson and Patrick Gallagher. Secure hash standard (SHS). FIPS 180-4, March 2012.

[6] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2:483–502, 2002.

[7] Lily Chen. Recommendation for key derivation using pseudorandom functions (revised). NIST Special Publication 800-108, October 2009.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[9] Nick Sheep Dalton and Ruth Conroy Dalton. The theory of natural movement and its application to the simulation of mobile ad hoc networks (MANET). In *Proceedings of the Fifth Annual Conference on Communication Networks and Services Research*, pages 359–363, 2007.

[10] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[12] Michael Feeley, Norman Hutchinson, and Suprio Ray. Realistic mobility for mobile ad hoc network simulation. In *Proceedings of the 3rd International Conference on Ad-Hoc Networks and Wireless*, pages 324–329, July 2004.

[13] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 2, chapter XI. Wiley, New York, 2nd edition, 1971.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[15] Pawel Gburzynski and Ioanis Nikolaidis. Wireless network simulation extensions in SIDE/SMURPH. In *Proceedings of the 2006 Winter Simulation Conference*, pages 2225–2233, 2006.

[16] Buğra Gedik and Ling Liu. A customizable k-anonymity model for protecting location privacy. In *The 24th International Conference on Distributed Computing Systems*, pages 620–629, 2004.

[17] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, February 1999.

[18] Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding individual human mobility patterns. *Nature*, 453:479–482, June 2008.

[19] Google. Google maps. `http://maps.google.com/`, 2010.

[20] R.L̃. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[21] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on mobile systems, applications and services*, 2003.

[22] Sariel Har-Peled and Anastasios Sidiropoulos. Clustering problem. Private communication, February 2013.

[23] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of System Science and Cybernetics*, 4(2):100–107, 1968.

[24] L.F̃. Henderson. The statistics of crowd fluids. *Nature*, 229:381–383, 1971.

[25] L.F̃. Henderson and D.J̃. Lyons. Sexual differences in human crowd motion. *Nature*, 240:353–355, 1972.

[26] Bill Hillier. *Space is the machine: a configurational theory of architecture*. Cambridge University Press, 1996.

[27] Bill Hillier and Shinichi Iida. Network and psychological effects in urban movement. *Lecture Notes in Computing Science*, 3693:475–490, 2005.

[28] Xiaoyan Hong, Jiejun Kong, and Mario Gerla. Mobility changes anonymity: new passive threats in mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 6:281–293, 2006.

[29] Jessica Horning, Ahmed El-Geneidy, and Kevin J. Krizek. Perceptions of walking distance to neighborhood retail and other public services. In *Transportation Research Board 87th Annual Meeting*. National Academy of Sciences, January 2008.

[30] Wei-Jen Hsu, Kashyap Merchant, Haw-Wei Shu, Chih-Hsin Hsu, and Ahmed Helmy. Weighted waypoint mobility model and its impact on ad hoc networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(1):59–63, January 2005.

[31] Jerzy W. Jaromczyk and Pawel Gburzynski. Pozdrowienia z ciechanowa. Private communication, July 2012.

[32] Ying Jian, Shigang Chen, Zhan Zhang, and Liang Zhang. Protecting receiver-location privacy in wireless sensor networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications*, pages 1955–1963, May 2007.

[33] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 353:153–181, 1996.

[34] David S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3:182–195, 1982.

[35] Pandurang Kamat, Yanyong Zhang, Wade Trappe, and Celal Ozturk. Enhancing source-location privacy in sensor network routing. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 599–608, 2005.

[36] Richard M. Karp. Reducibility among combinatorial problems. In Raymond Edward Miller and James W. Thatcher, editors, *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

[37] Jonghyun Kim, Vinay Sridhara, and Stephan Bohacek. Realistic mobility simulation of urban mesh networks. *Ad Hoc Networks*, 7:411–430, 2009.

[38] Minkyong Kim, David Kotz, and Songkuk Kim. Extracting a mobility model from real user traces. In *Proceedings of the 25th Annual IEEE Infocom Conference*, April 2006.

[39] Vassilis Kostakos. Space Syntax and pervasive systems. In Bin Jiang and Xiaobai Yao, editors, *Geospatial Analysis and Modeling of Urban Structure and Dynamics*, pages 31–52. Springer, 2010.

[40] Vassilis Kostakos, Tom Nicolai, Eiko Yoneki, Eamonn O'Neill, Holger Kenn, and Jon Crowcroft. Understanding and measuring the urban pervasive infrastructure. *Journal of Personal and Ubiquitous Computing*, 13:355–364, 2009.

[41] Jean-Yves Le Boudec and Milan Vojnović. Perfect simulation and stationarity of a class of mobility models. In *Proceedings of the 24th Annual IEEE Infocom Conference*, pages 2743–2754, March 2005.

[42] Javier Lopez. Unleashing public-key cryptography in wireless sensor networks. *Journal of Computer Security*, 14(5):469–482, 2006.

[43] Satyajayant Misra and Guoliang Xue. Efficient anonymity schemes for clustered wireless sensor networks. *International Journal of Sensor Networks*, 1(1/2):50–63, 2006.

[44] Daniel R. Montello. The perception and cognition of environmental distance: Direct sources of information. In S.C̃. Hirtle and A˙ U. Frank, editors, *Proceedings of COSIT '97*, pages 297–311, 1997.

[45] Peter Olofsson. *Probabilities: The Little Numbers That Rule Our Lives*. John Wiley & Sons, Inc., 2007.

[46] Yi Ouyang, Zhengyi Le, Yurong Xu, Nikos Triandopoulos, Sheng Zhang, James Ford, and Fillia Makedon. Providing anonymity in wireless sensor networks. In *Proceedings of the IEEE International Conference on Pervasive Services*, pages 145–148, 2007.

[47] Celal Ozturk, Yanyong Zhang, and Wade Trappe. Source-location privacy in energy-constrained sensor network routing. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 88–93, 2004.

[48] John Peponis, Jean Wineman, Mahbub Rashid, S Kim, and Sonit Bafna. On the generation of linear representations of spatial configuration. *Environment and Planning B: Planning and Design*, 25(4):559–576, 1998.

[49] Pedro C. Pinto, João Barros, and Moe Z. Win. Wireless secrecy in large-scale networks. arXiv:1102.3617 [cs.IT], February 2011.

[50] S.S̃. Ravi, D.J̃. Rosenkrantz, and G.K̃. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, March-April 1994.

[51] Capt. Ross E.R̃oley. Spacing points in a three dimensional convex region for maximum separation: a color-space application. Master's thesis, Air Force Aerospace Medical Research Laboratory, April 1985.

[52] Thomas J. Schaefer. Private communication with a third-party, recited, 1974.

[53] GraphPad Software. Graphpad statistics guide: the Bonferroni method. `http://www.graphpad.com/guides/prism/6/statistics/index.htm?stat_the_bonferroni_method.htm`, 2013.

[54] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Nature*, 327:1018–1021, 2010.

[55] Paul G. Spirakis. Very fast algorithms for the area of the union of many circles. Technical Report 98, New York University, December 1983.

[56] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[57] Arie Tamir. Obnoxious facility location on graphs. *Society for Industrial and Applied Mathematics Journal on Discrete Mathematics*, 4(4):550–567, November 1991.

[58] Godfried Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, May 1983.

[59] Alasdair Turner. Depthmap: A program to perform visibility graph analysis. In *Proceedings of the 3rd International Symposium on Space Syntax*, pages 31.1–31.9, May 2001.

[60] Christopher N. Ververidis and George C. Polyzos. A routing layer based approach for energy efficient service discovery in mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 9:655–672, 2009.

[61] Ryan Vogt, Mario Nascimento, and Janelle Harms. On the trade-off between user-location privacy and queried-location privacy in wireless sensor networks. In *Proceedings of the 8th International Conference on Ad-Hoc Networks and Wireless*, pages 241–254, 2009.

[62] Ryan Vogt, Ioanis Nikolaidis, and Pawel Gburzynski. A realistic outdoor urban pedestrian mobility model. *Simulation Modelling Practice and Theory*, 26:113–134, 2012.

[63] D.W. Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information Processing Letters*, 29:281–286, 1988.

[64] Stephen B. Wicker. The loss of location privacy in the cellular age. *Communications of the ACM*, 55(8):60–68, August 2012.

[65] Jean Wineman, James Turner, Sophia Psarra, Sung Kwong Jung, and Nicholas Senske. Syntax2d: An open source software platform for Space Syntax analysis. In *Proceedings of New Developments in Space Syntax Software*, pages 23–26, June 2007.

[66] Xiaojing Xiang, Xin Wang, and Yuanyuan Yang. Stateless multicasting in mobile ad hoc networks. *IEEE Transactions on Computers*, 59(8):1078–1090, 2010.

[67] Yi Yang, Sencun Zhu, Guohong Cao, and Thomas LaPorta. An active global attack model for sensor source location privacy: Analysis and countermeasures. *Security and Privacy in Communication Networks*, 19:373–393, 2009.

[68] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, August 2008.

[69] Jungkeun Yoon, Mingyan Liu, and Brian Noble. Random waypoint considered harmful. In *Proceedings of the 22th Annual IEEE Infocom Conference*, pages 1312–1321, March 2003.

[70] Jungkeun Yoon, Mingyan Liu, and Brian Noble. Sound mobility models. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, pages 205–216, September 2003.

[71] Jungkeun Yoon, Brian D. Noble, Mingyan Liu, and Minkyong Kim. Building realistic mobility models from coarse-grained traces. In *Proceedings of the 4th International Conference on Mobile Systems, Applications, and Services*, pages 177–190, June 2006.

[72] Lisa Zyga. Study shows how easy it is to determine someone's identity with cell phone data. `http://phys.org/news/2013-03-easy-identity-cell.html`, March 2013.