X. Wang, X. Hou, R. Rios, N. Ole Tippenhauer, and M. Ochoa, "Constrained Proximity Attacks on Mobile Targ
ACM Transactions on Privacy and Security (TOPS), vol. 25, pp. 1 - 29, 2022.
http://doi.org/10.1145/3498543
NICS Lab. Publications: https://www.nics.uma.es/publications

Constrained Proximity Attacks on Mobile Targets

XUEOU WANG, Institute of Data Science, National University of Singapore, Singapore
XIAOLU HOU, Faculty of Informatics and Information Technologies, Slovak University of Technology, Slovakia
RUBEN RIOS, NICS Lab & ITIS Software, University of Malaga, Spain
NILS OLE TIPPENHAUER, CISPA Helmholtz Center for Information Security, Germany
MARTÍN OCHOA, Department of Computer Science, ETH Zurich, Switzerland

Proximity attacks allow an adversary to uncover the location of a victim by repeatedly issuing queries with fake location data. These attacks have been mostly studied in scenarios where victims remain static and there are no constraints that limit the actions of the attacker. In such a setting, it is not difficult for the attacker to locate a particular victim and quantifying the effort for doing so is straightforward. However, it is far more realistic to consider scenarios where potential victims present a particular mobility pattern. In this paper, we consider abstract (constrained and unconstrained) attacks on services that provide location information on other users in the proximity. We derive strategies for constrained and unconstrained attackers, and show that when unconstrained they can practically achieve success with theoretically optimal effort. We then propose a simple yet effective constraint that may be employed by a proximity service (for example, running in the cloud or using a suitable two-party protocol) as countermeasure to increase the effort for the attacker several orders of magnitude both in simulated and real-world cases.

CCS Concepts: • **Security and privacy** → **Privacy protections**; **Privacy-preserving protocols**.

Additional Key Words and Phrases: location privacy, proximity attacks, mobility pattern, quantification

ACM Reference Format:
Xueou Wang, Xiaolu Hou, Ruben Rios, Nils Ole Tippenhauer, and Martín Ochoa. 2020. Constrained Proximity Attacks on Mobile Targets. *ACM Trans. Priv. Sec.* 1, 1, Article 1 (January 2020), 29 pages. https://doi.org/10.1145/3498543

1 INTRODUCTION

Proximity services are a special type of location-based service (LBS) where the user is informed about nearby people of interest and their distance rather than their exact location in an attempt to protect location privacy. To that end, queries are sent to the proximity service including the location of the user, the search radius and possibly some information about the target. Unfortunately, when

*The work of Martín Ochoa was done while he was with Appgate Inc.

Authors' addresses: Xueou Wang, xueou.wang@gmail.com, Institute of Data Science, National University of Singapore, innovation 4.0, 04-06, 3 Research Link, Singapore, Singapore, 117602; Xiaolu Hou, houxiaolu.email@gmail.com, Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovicova 2, Karlova Ves, Bratislava, Slovakia, 842 16; Ruben Rios, NICS Lab & ITIS Software, University of Malaga, Ada Byron Research Building, Campus Teatinos, 29071, Malaga, Spain, ruben@lcc.uma.es; Nils Ole Tippenhauer, CISPA Helmholtz Center for Information Security, Stuhlsatzenhaus 5, 66123, SaarbrÃijcken, Germany, tippenhauer@cispa.de; Martín Ochoa, Department of Computer Science, ETH Zurich, UniversitÃďtstrasse 6, 8092, Zurich, Switzerland , martin.ochoa@inf.ethz.ch.

2471-2566/2020/1-ART1 $15.00
https://doi.org/10.1145/3498543

the exact distance to a user is revealed by the proximity service it is possible to retrieve the exact location of the user. Examples include claims that Egyptian authorities leveraged dating apps to track down gay users [6], and others attempted to find locations of Tinder users [19]. Consequently, the need to provide rigorous privacy guarantees in proximity services is evident.

In view of these threats, some effort has been devoted to the development of privacy-preserving proximity testing protocols [8, 12, 15]. These solutions allow two users to learn whether they are within a certain distance of each other but no further information about their location or distance is revealed to the other user. Some of these protocols rely on a trusted third party to handle users' locations while others get rid of this third party and operate in a decentralized way using cryptographic 2-party protocols.

In general, these solutions focus on partially static models, where attackers can change their location freely but victims can not. Capturing the behavior of an optimal attacker in this setting when the victim is static is already hard although some progress has been made in recent years [9, 13]. However, this situation covers only a particular case of a victim's behavior (i.e. user is at home or office) and motivated us to investigate the expected effort for adversaries to localize users that move in a particular mobility pattern, which covers the more general case of a victim that might be travelling, moving around the city etc.

Although there has been extensive study in mobility models [2], attack strategies [11, 13, 19], location privacy protection mechanisms [8, 9, 18] and location privacy quantification metrics [16, 17], there is very limited research on location prediction based on sequential spatio-temporal data using a probabilistic approach. Given that location data acquired from an LBS platform are sequential spatio-temporal data, one can obtain information on the moving patterns/behaviors of the user by analyzing the trajectory dataset.

In this work, we are interested in quantifying the effort of an *arbitrary attacker* issuing proximity queries in finding a user under certain models. In other words: *how quickly can an attacker locate a user based on queries to the LBS*? In addition, we address the following question: *can we design a system that makes the expected location effort of an attacker more difficult while not harming well-intended users*? Inspired by work [9] that imposes constraints on the speed at which an attacker can move in the plane, we have set out to explore how such restrictions would impact the attacker effort in the moving target scenario. The main idea is that normally users will move at a maximum speed, such as walking, biking or car-riding speeds, whereas attackers might make arbitrary queries that would imply unrealistic moving speeds. By limiting the speed at which originating positions by a querying party (attacker) can change, proximity attacks are mitigated. This countermeasure can be applied both in the centralized (i.e. cloud provider) environment, as well as in the 2-party setting with provable guarantees as shown in [9].

Our main contributions are as follows:

(1) We give upper and lower bounds on the minimum number of queries an attacker needs to issue to locate the victim with probability $\frac{1}{2}$ (generalizable to other probabilities). In particular, for a search space of size M and assumptions on victim's initial location and mobility, we show that an optimal attacker needs at most $\frac{M}{2}$ queries to locate a victim with probability $\frac{1}{2}$.
(2) We derive and implement a novel *Linear Jump Strategy* from the proof, and show empirically that its effort falls within the theoretical bounds. We find that for non-uniform initial distributions of victims, the strategy performs worse. To address this, we propose a *Greedy Updating Attacker Strategy*.
(3) We study the impact of constraining the speed of attacker queries, and show that with this countermeasure in place an attacker performs significantly worse in simulated scenarios
(4) We evaluate our approach on a large dataset containing real world mobility patterns [22, 23].

(5) We share the code used in our simulations and bound estimations as an open source project under https://github.com/nicslabdev/proximityattacks.

Compared to [20], we introduce new mobility models for users that follow a given roadmap-based topology and describe algorithms to compute the related probability matrix using a map and/or traffic data. We also introduce constraints for the attacker, which requires a new mobility model and the design and evaluation of novel attacker strategies. Our new results demonstrate that basic speed constraints (for example, enforced in the cloud) greatly increase the time and cost of the attack. We find that, the less random the initial distribution of the victim is, the harder for the attacker to perform efficient attacking. Given the difficulty to extend our analytical results (that apply for arbitrary optimal attackers) from the one-dimensional space to two-dimensions, we show computational evidence that the bound of $\frac{M}{2}$ also applies for several planes of bounded sizes.

The rest of this work is organized as follows. In Section 2, we present our mathematical modeling of search spaces and mobility patterns. Section 3 summarizes the problem statement. We present the mathematical analysis for calculating the probability of an attacker locating a victim using location proximity queries in Section 4. Computational bounds are also derived in this section. In Section 5, we present our practical linear jumping and greedy attacker strategies. Section 6 provides algorithms for making our simulations more realistic by incorporating road topology and mobility information. In Section 7, we evaluate the proposed strategies under synthetic and real-word settings. A countermeasure is presented and evaluated in Section 8, where it is shown that by enforcing such countermeasure the effectiveness of the attacker can be greatly diminished. Related work and conclusions are discussed in Sections 9 and 10, respectively.

2 PRELIMINARIES

2.1 Search Domain

In our model, users are able to move in a finite space that can be divided into discrete *locations*. The granularity of the locations is limited by the maximum precision of the positioning device or privacy considerations [5].

Space. For the two-dimensional case, the search space can be divided into $m \times n$ locations. Each point typically has four adjacent locations, except for the corners. We call this space $\mathcal{S}^M$, with $M = mn$. We focus on the two-dimensional case as it is by far the most common (as used on a regular map), but use the simpler one-dimensional representation to make the analysis more concise.

Most points in a two-dimensional space have four adjacent points, but in a finite search space there are some (edge) points where the number of adjacent points is smaller. These locations can be sequentially numbered. To go back and forth from a single dimension representation of a point to a two-dimensional one we use the following projections:

$$\begin{aligned} proj(i) &: \mathbb{N} \to \mathbb{N} \times \mathbb{Z}_n = (i \text{ div } n, i \bmod n) \\ proj^{-1}(i,j) &: \mathbb{N} \times \mathbb{Z}_n \to \mathbb{N} = in + j, \end{aligned} \tag{1}$$

where $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.

Time. We assume discrete time steps $k \in \{0, 1, 2, \ldots\}$. In each time step, the user $\mathcal{X}$ can move once. The movement of users is represented as a transition in our model.

Location. A user $\mathcal{X}$ located in the space $\mathcal{S}^M$ at time k is denoted $\mathcal{X}_k$, with $k \in \mathbb{Z}_{\geq 0}$ and $\mathcal{X}_k \in \{0, 1, \ldots, M-1\}$. The location of $\mathcal{X}$ at time 0 is called the *initial location of* $\mathcal{X}$. Using the above projection, we can define positioning and movement in both dimensions one and two. Given a space $\mathcal{S}^M$, an entity's possible positions are $\{0, 1, \ldots, M-1\}$.

The movement of entities is represented as a shift in our model. If an entity $\mathcal{X}$ is positioned in the space $\mathcal{S}^M$, at time k, at position $\mathcal{X}_k$, its position at time $k+1$ can be represented as $\mathcal{X}_{k+1} = \mathcal{X}_k + j$ for some $j \in \mathbb{Z}$ such that $\mathcal{X}_{k+1} \in \{0, 1, \ldots, M-1\}$. The set of possible values for $\mathcal{X}_{k+1}$ will be determined by the particular mobility pattern of the entity.

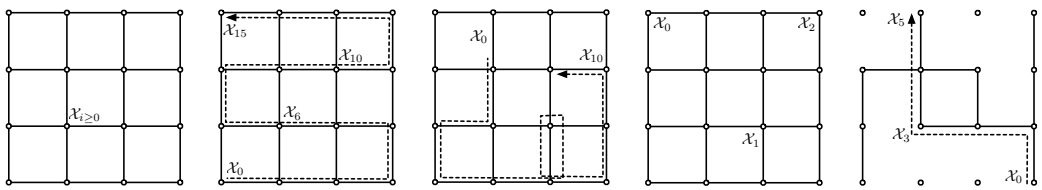
As an example, if an entity in a two-dimensional search space $\mathcal{S}^M$ ($M = mn$), is at $\mathcal{X}_i$ at time i, and at time $i+1$, $\mathcal{X}$ has moved one point upwards and one to the left, we have $\mathcal{X}_{i+1} = \mathcal{X}_i + 1 - n$, and note that using the projection in (1) we get:

$$\begin{aligned} &proj(\mathcal{X}_i + 1 - n) \\ &\quad = (\mathcal{X}_i + 1 - n \text{ div } n, \mathcal{X}_i + 1 - n \bmod n) \\ &\quad = (\mathcal{X}_i \text{ div } n - 1, \mathcal{X}_i \bmod n + 1), \end{aligned}$$

which intuitively means to decrease the x-coordinate of $\mathcal{X}_i$ by one and increase the y-coordinate by one.

2.2 Mobility Models

We now informally describe some common mobility patterns (see Figure 1) to give a better intuition of our modeling of search spaces and descriptions of entities moving in the search space. We will use mobility patterns and mobility models interchangeably throughout this paper. Mobility patterns can be used to describe an attacker's search strategy or a victim's common mobility pattern. In the following sections we will focus first on a Random Walk mobility pattern for a victim to gain an intuition on how different attacker strategies perform in this case, and later we will connect this intuition to the more realistic case of a victim moving on a concrete city street map.



(a) Static (b) Linear (c) Random Walk (d) Strategic Jump (e) Map walk

Fig. 1. Mobility Models

Static mobility model An entity $\mathcal{X}$ that follows a static mobility model starts at a random initial position r in the search space $\mathcal{S}^M$ and remains in the same position. For example, a person who stays at home or in the office. With a fixed $r \in \mathcal{S}^M$:

$$\mathcal{X}_{k+1} = \mathcal{X}_k,\ \mathcal{X}_0 = r,\ k \in \mathbb{Z}_{\geq 0}$$

Linear mobility model An entity $\mathcal{X}$ that follows a linear mobility model starts at an arbitrary initial position $\mathcal{X}_0$ within the search space $\mathcal{S}^M$ ($M = mn$ for dimension two and $M = n$ for dimension one) and keeps moving such that the following conditions are satisfied:

(1) $\{\mathcal{X}_{iM}, \mathcal{X}_{iM+1}, \ldots, \mathcal{X}_{iM+M-1}\} = \{0, 1, \ldots, M-1\}$ for all $i \in \mathbb{Z}_{\geq 0}$;
(2) $\mathcal{X}_{k+1} \in \{\mathcal{X}_k + 1, \mathcal{X}_k - 1\}$ for a one dimensional search space of size n ($k \in \mathbb{Z}_{\geq 0}$).
(3) $\mathcal{X}_{k+1} \in \{\mathcal{X}_k + n, \mathcal{X}_k - 1, \mathcal{X}_k + 1, \mathcal{X}_k - n\}$ for a two dimensional search space of size m by n ($k \in \mathbb{Z}_{\geq 0}$);

This model may apply when a person is driving a car on the road.

Random walk mobility model An entity $\mathcal{X}$ that follows a random walk mobility model starts at a random position and decides its next move uniformly at random from all the positions in its vicinity.

Strategic jump An entity $\mathcal{X}$ that follows this strategy can arbitrarily move to any other position. In other words, the next position is sampled freshly from an arbitrary distribution and can be arbitrarily far away from the current position (thus 'jump'). An attacker, for example, who can fake his/her location as a means to perform attacks, such as trilateration, could be described using the random jump model.

Route in street map In this mobility model there is a baseline map that makes some mobility choices more likely (for instance follow a given street). This adds constraints to the mobility pattern of $\mathcal{X}$, since for instance driving into a lake is impossible or very unlikely. This pattern is not deterministic though, there's a probability distribution that describes the next steps given an initial position, for instance $1/2$ probability of moving north in a street and $1/2$ of moving south.

3 PROBLEM STATEMENT

Based on the modeling of search spaces and possible mobility patterns from Section 2, we proceed to give a formal mathematical description of the problem statement addressed in this work.

3.1 System and Attacker Model

Let Alice ($\mathcal{A}$) be the attacker and Bob ($\mathcal{B}$) be a user whose location is of interest to Alice. Bob uses an LBS that will disclose Bob's presence at location $\mathcal{B}_k$ to other users that claim to be at the same location. Bob (and Alice) can only make one location claim per discrete time step k. Each time k, Bob will move once (and update his location on the LBS), and Alice will thus be able to perform one query to the LBS to verify if Bob is at Alice's claimed location $\mathcal{A}_k$. Alice sends the first query at time $k = 0$, and k is the time of the $(k + 1)$-th query.

The goal of Alice is to minimize the number of queries that she needs to send to LBS to be able to verify Bob's location. Conversely, Bob does not have a particular goal except to use the service privately. He is not even aware of being tracked. Alice on the other hand is assumed to have a priori information about Bob's probability distribution obtained from past observations, external sources, geographic features of a given city or location or a combination thereof. Later in Section 7.3, we will discuss a real data example, where the attacker can obtain information from historical trajectory data of a victim.

Consequently, since Alice can cheat about her location, send as many queries as she wants, and her goal is to find Bob as quickly as possible, her mobility pattern can be arbitrary. Some potential strategies of the adversary are shown in Figure 1. Note that these strategies are not exhaustive since, as we will see in the following sections, an optimal strategy might involve jumping two positions at a time, and this can be arbitrarily complex (2 positions in one directions, then 3 positions in another directions and so on). On the other hand, we assume an honest user like Bob follows a realistic mobility pattern where subsequent locations are contiguous to each other.

Finally note that while we use a third party LBS in this system model for simplicity, similar scenarios could be constructed if Alice and Bob engage in a privacy-preserving proximity protocol that is initiated by Alice and where the inputs of both of them remain private (e.g., the protocol discussed in [8]).

3.2 Problem Statement and Formalization

For a fixed probability p, we would like to calculate the number of steps needed by the attacker in order to locate Bob with probability at least p. In the following we present the underlying formalization.

Probabilistic Locations. Consider a search space $\mathcal{S}^M$. We posit the mobility model of Bob can be described by a transition matrix P where each entry of P is a transition probability p_{ij} representing the probability of Bob moving from location i to location j in one step. Furthermore, we assume the probability of Bob moving from location i to location j is the same at any step. More precisely:

$$\Pr(\mathcal{B}_k = j | \mathcal{B}_{k-1} = i) = p_{ij},\ \forall k \in \mathbb{Z}_{\geq 1} \text{ and } i, j \in \mathcal{S}^M.$$

Thus, it is straightforward to calculate the probability of Bob being at a particular location after k steps by simply taking the kth power of the transition matrix.

Let $B^{(k)}$ ($k \in \mathbb{Z}_{\geq 0}$) be a vector representing the probability of Bob being at each location j ($j \in \{0, 1, \ldots, M-1\}$) after k steps, i.e. $B_j^{(k)} = \Pr(\mathcal{B}_k = j)$. We have $B^{(k+1)} = B^{(k)}P$.

Attacker Strategies. For a fixed attacker strategy $\mathcal{A} = \mathcal{A}_0, \mathcal{A}_1, \ldots$, we are interested in the probability of two events: E_k, and F_j. E_k is the event that Alice locates Bob within k steps:

$$E_k := \{\exists i \leq k \text{ s.t. } \mathcal{A}_i = \mathcal{B}_i\}$$

F_j is the event that Alice locates Bob exactly at step j:

$$F_j := \{\mathcal{A}_j = \mathcal{B}_j\}.$$

Problem Statement. We are interested in finding $k_{O,p}$: the number of queries required by an optimal attacker strategy to locate Bob with probability of at least p. Formally, we define $k_{O,p}$ as follows:

$$k_{O,p} := \min_{\mathcal{A}} k_{\mathcal{A},p}, \tag{2}$$

with $k_{\mathcal{A},p}$ being the number of steps required by a specific attacker strategy $\mathcal{A}$ to locate Bob with probability $0 \leq p \leq 1$. We can find that number as follows:

$$k_{\mathcal{A},p} := \min\{k : \Pr(E_k) \geq p\}.$$

Given the above, we can estimate some upper and lower bounds on these values that will be useful to compare attacker strategies in the following.

Upper Bound for $\Pr(E_k)$. By definition,

$$\Pr(E_k) = \Pr(F_0 \cup F_1 \cdots \cup F_k),$$

which gives the following *upper bound* for $\Pr(E_k)$ since the probability of finding Bob at step i is at most equal to the probability of Bob's most likely location at that step:

$$\Pr(E_k) \leq \Pr(F_0) + \Pr(F_1) + \cdots + \Pr(F_k) \leq \sum_{i=0}^{k} \max_j B_j^{(i)}. \tag{3}$$

Note that the above upper bound on $\Pr(E_k)$ holds for any attacker strategy $\mathcal{A}$.

Lower Bound on k. We define

$$k_{lower,p} := \min\left\{k : \sum_{i=0}^{k} \max_j B_j^{(i)} \geq p\right\}. \tag{4}$$

In view of Equation (3), $k_{lower,p} \leq k_{O,p}$ is a *lower bound* of $k_{O,p}$.

In the following we will mostly focus on the case $p = 0.5$. To simplify the notations, we define

$$k_{\mathcal{A}} := k_{\mathcal{A},0.5} = \min\{k : \Pr(E_k) \geq 0.5\},$$
$$k_O := k_{O,0.5} = \min_{\mathcal{A}} k_{\mathcal{A}},$$
$$k_{lower} := k_{lower,0.5}.$$

Thus if Alice follows strategy $\mathcal{A}$, $k_{\mathcal{A}}$ (resp. $k_{\mathcal{A}} + 1$) is the number of steps (resp. number of queries) needed for Alice to locate Bob with a probability of at least 0.5. k_O (resp. $k_O + 1$) is the minimum number of steps (resp. minimum number of queries) needed for Alice to locate Bob with a probability of at least 0.5 independently of strategy used. In addition, $k_{lower} \leq k_O$ is a lower bound of k_O.

4 ATTACKER EFFORT QUANTIFICATION

Next we present the theoretical analysis on the minimal number of steps required by an optimal attack strategy to locate a particular victim. First, we derive the formula for calculating $\Pr(E_k)$ and then we consider the case of a victim following a random walk mobility model. In Section 4.1.1, with a simple example in a one-dimensional search space of size 5, we illustrate how this formula and the upper bound in (3) can be used to analyze different attacker strategies and get $k_{O,0.5}$. We choose a random walk model to give a rigorous mathematical analysis. In more complex moving patterns, the problem may not have analytical or closed solutions because the transition matrix is indefinable or needs to be recursively updated. Our mathematical derivation will be presented for the one-dimensional case only as we can always project two-dimensions to a one-dimensional space.

4.1 General formula to estimate attacker's effort

In the following we discuss how to derive an explicit formula to compute the probability to find a victim with k steps, and thus, how to compute the minimum number of steps k to find a victim with probability greater than p. This formula assumes that both the mobility pattern and the attack strategy are known.

Given the transition matrix P and the vector $B^{(0)}$ of initial position probabilities, $B^{(k)} = B^{(0)}P^k$ gives the probabilities of Bob being at each different position after k steps. More precisely, $B^{(k)}_j = \Pr(\mathcal{B}_k = j)$ is the probability of Bob at position j after k steps.

Let $P^i_{j_1 j_2}$ denote the $(j_1, j_2)-$entry of the matrix P^i. It gives the probability of Bob going from position j_1 to position j_2 in i steps, i.e. for any $k \in \mathbb{Z}_{\geq 0}$,

$$\Pr(\mathcal{B}_{i+k} = j_2 | \mathcal{B}_k = j_1) = P^i_{j_1 j_2}.$$

Fixing an attacker strategy $\mathcal{A}$, let $\mathcal{A}_i$ denote the position of Alice at step i. For any positive integers $i_1 < i_2$, the probability of Alice locating Bob at both steps i_1 and i_2 is equal to the probability of Bob being at position $\mathcal{A}_{i_1}$ at step i_1 multiplied by the probability of Bob reaching position $\mathcal{A}_{i_2}$ in $i_2 - i_1$ steps, i.e.

$$\begin{aligned}\Pr(F_{i_1} \cap F_{i_2}) &= \Pr(\mathcal{B}_{i_1} = \mathcal{A}_{i_1} \cap \mathcal{B}_{i_2} = \mathcal{A}_{i_2}) &(5)\\ &= \Pr(\mathcal{B}_{i_1} = \mathcal{A}_{i_1}) \Pr(\mathcal{B}_{i_2} = \mathcal{A}_{i_2} | \mathcal{B}_{i_1} = \mathcal{A}_{i_1}) \\ &= B^{(i_1)}_{\mathcal{A}_{i_1}} P^{i_2 - i_1}_{\mathcal{A}_{i_1}\mathcal{A}_{i_2}}. &(6)\end{aligned}$$

To get a general formula for $\Pr(E_k)$, we first note

$$\begin{aligned}\Pr(E_k) &= \Pr(F_0 \cup F_1 \cup \cdots \cup F_k)\\ &= \Pr((F_0 \cup F_1 \cup \cdots \cup F_{k-1})^c \cap F_k)\\ &= \Pr(F_0) + \Pr(F_0^c \cap F_1) + \Pr((F_0 \cup F_1)^c \cap F_2) +\\ &\quad \cdots + \Pr((F_0 \cup F_1 \cup \cdots \cup F_{k-1})^c \cap F_k)\\ &= \sum_{0 \le i \le k} \Pr((F_0 \cup F_1 \cup \cdots \cup F_{i-1})^c \cap F_i), \end{aligned} \tag{7}$$

where F^c is the complement of F, i.e., F_j^c means Alice does not successfully locates Bob at step j. Our GUAS attacker strategy uses this result for aggregate success computation (see Section 5.2). From probability theory we can also write

$$\Pr(E_k) = \Pr(F_0 \cup F_1 \cup \cdots \cup F_k) = \sum_{m=0}^{k} (-1)^{m+1} \left(\sum_{0 \le i_1 < \cdots < i_m \le k} \Pr(F_{i_1} \cap \cdots \cap F_{i_m}) \right) \tag{8}$$

By combining Equation 5 with Equation 8, we have the following general formula for $\Pr(E_k)$:

$$\begin{aligned}&\sum_{m=0}^{k} B^{(m)}_{\mathcal{A}_m} \left(1 + \sum_{\ell=1}^{k-m} (-1)^{\ell} \sum_{m < i_1 < \cdots < i_\ell \le k} P^{i_1-m}_{\mathcal{A}_m \mathcal{A}_{i_1}} \cdots P^{i_\ell - i_{\ell-1}}_{\mathcal{A}_{i_{\ell-1}} \mathcal{A}_{i_\ell}} \right)\\ &\qquad = B^{(0)}_{\mathcal{A}_0} \left(1 - \sum_{i=1}^{k} P^{i}_{\mathcal{A}_0 \mathcal{A}_i} + \sum_{1 \le i_1 < i_2 \le k} P^{i_1}_{\mathcal{A}_0 \mathcal{A}_{i_1}} P^{i_2 - i_1}_{\mathcal{A}_{i_1} \mathcal{A}_{i_2}} - \ldots \right)\\ &\qquad\quad + B^{(1)}_{\mathcal{A}_1} \left(1 - \sum_{i=2}^{k} P^{i-1}_{\mathcal{A}_1 \mathcal{A}_i} + \sum_{2 \le i_1 < i_2 \le k} P^{i_1 - 1}_{\mathcal{A}_1 \mathcal{A}_{i_1}} P^{i_2 - i_1}_{\mathcal{A}_{i_1} \mathcal{A}_{i_2}} - \ldots \right) + \cdots + B^{(k)}_{\mathcal{A}_k}. \end{aligned} \tag{9}$$

If the attacker strategy is not deterministic, then for one fixed sequence of positions $a_0, a_1, a_2, \ldots, a_k$, $\Pr(E_k | \mathcal{A}_0 = a_0 \cap \mathcal{A}_1 = a_1 \cap \cdots \cap \mathcal{A}_k = a_k)$ is given by

$$\sum_{m=0}^{k} B^{(m)}_{a_m} \left(1 + \sum_{\ell=1}^{k-m} (-1)^{\ell} \sum_{m < i_1 < i_2 < \cdots < i_\ell \le k} P^{i_1 - m}_{a_m a_{i_1}} P^{i_2 - i_1}_{a_{i_1} a_{i_2}} \cdots P^{i_\ell - i_{\ell-1}}_{a_{i_{\ell-1}} a_{i_\ell}} \right),$$

and we have the following formula for $\Pr(E_k)$

$$\sum_{1 \le a_0, a_1, \ldots, a_k \le n} \Pr(\mathcal{A}_0 = a_0 \cap \cdots \cap \mathcal{A}_k = a_k) \cdot \Pr(E_k | \mathcal{A}_0 = a_0 \cap \cdots \cap \mathcal{A}_k = a_k) \tag{10}$$

Note that when the attacker strategy is deterministic, for one sequence of positions $a_0, a_1, a_2, \ldots, a_k$ we have $\Pr(\mathcal{A}_0 = a_0, \ldots, \mathcal{A}_k = a_k) = 1$ and we get the formula in (9).

In the following, we will assume Bob follows a random walk strategy that can be represented as a Markovian process with transition probabilities p_{ij}, consistent with the provisions of the random walk mobility model in Section 2.2.

4.1.1 Example for linear and strategic attacks. To illustrate our approach to estimate an attacker's effort, we consider a one-dimensional search space of size 5. The probability of Bob being at each particular position of the search space can be calculated using the initial position matrix $B^{(0)}$ and

transition matrix P as follows:

$$B^{(0)} = \left[\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right], \quad P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Furthermore,

$$\begin{aligned} B^{(1)} &= \left[\frac{1}{10}, \frac{3}{10}, \frac{1}{5}, \frac{3}{10}, \frac{1}{10}\right] \\ B^{(2)} &= \left[\frac{3}{20}, \frac{1}{5}, \frac{3}{10}, \frac{1}{5}, \frac{3}{20}\right] \\ B^{(3)} &= \left[\frac{1}{10}, \frac{3}{10}, \frac{1}{5}, \frac{3}{10}, \frac{1}{10}\right] = B^{(1)} \\ B^{(4)} &= B^{(2)}. \end{aligned}$$

We are interested in calculating $k_O = \min_{\mathcal{A}} k_{\mathcal{A}}$. By (3), for any attacker,

$$\Pr(E_0) \leq \frac{1}{5}, \ \Pr(E_1) \leq \frac{1}{5} + \frac{3}{10} = \frac{1}{2}.$$

Thus $k_O \geq k_{lower} = 1$. Next, we compare two different attacker strategies. We use (9) to calculate $\Pr(E_k)$.

Linear Attacker Consider a linear attacker $\mathcal{A}_{linear}$, i.e. $\mathcal{A}_0 = 0, \mathcal{A}_1 = 1, \mathcal{A}_2 = 2, \ldots$. Then

$$\begin{aligned} \Pr(E_0) &= \Pr(F_0) = \frac{1}{5}, \\ \Pr(E_1) &= \Pr(F_0 \cup F_1) \\ &= \Pr(F_0) + \Pr(F_1) - \Pr(F_0 \cap F_1) \\ &= B_0^{(0)} + B_1^{(1)} - B_0^{(0)} \Pr(\mathcal{B}_1 = 1 | \mathcal{B}_0 = 0) \\ &= B_0^{(0)}(1 - P_{01}) + B_1^{(1)} = \frac{3}{10}, \\ \Pr(E_2) &= B_0^{(0)}(1 - P_{01} - P_{02}^2 + P_{01}P_{12}) \\ &\quad + B_1^{(1)}(1 - P_{12}) + B_2^{(2)} = \frac{9}{20}, \\ \Pr(E_3) &= B_0^{(0)}(1 - P_{01} - P_{02}^2 - P_{03}^3 + P_{01}P_{12} \\ &\quad + P_{02}^2 P_{23} + P_{01}P_{13}^2 - P_{01}P_{12}P_{23}) \\ &\quad + B_1^{(1)}(1 - P_{12} - P_{13}^2 + P_{12}P_{23}) \\ &\quad + B_2^{(2)}(1 - P_{23}) + B_3^{(3)} = \frac{3}{5}. \end{aligned}$$

For an attacker following linear mobility model, we need $k \geq 3$ to have $\Pr(E_k) \geq \frac{1}{2}$, i.e. $k_{\mathcal{A}_{linear}} = 3$.

Strategic jump For an attacker who can jump to any location at any time she wishes, one possible strategy may be to choose $\mathcal{A}_k = m$ s.t.

$$\begin{aligned} B_m^{(k)} &= \max\{B_i^{(k)} : F_j \cap F_k = \emptyset, \ \forall 0 \leq j \leq k-1\} \\ &= \max\{B_i^{(k)} : P_{\mathcal{A}_j i}^{k-j} = 0, \ \forall 0 \leq j \leq k-1\}. \end{aligned}$$

We note that sometimes there are multiple choices of m and sometimes there is no such choice. Nevertheless, for this particular example, if the attacker chooses the strategy $\mathcal{A}_{Strategic}$ with $\mathcal{A}_0 = 0, \mathcal{A}_1 = 3, \mathcal{A}_2 = 3, \mathcal{A}_3 = 0$, then

$$\Pr(E_0) = B_0^{(0)} = \frac{1}{5},$$
$$\Pr(E_1) = B_0^{(0)} + B_3^{(1)} = \frac{1}{2},$$

which gives $k_{\mathcal{A}_{Strategic}} = 1$.

Thus, we have shown for a one-dimensional search space of size 5, $k_O = 1$ and we have given two different attacker strategies to achieve this value: $\mathcal{A}_{linear}$ and $\mathcal{A}_{Strategic}$. We note that Strategic jump is a special case of GUAS presented in Section 5.2 when the attackers knows both $B^{(0)}$ and P.

When n is much bigger than 5, there are many more different attacker mobility models that can be considered. The methods used above for calculating k_O and manually finding the best attacker strategy will not be applicable. In the rest of the paper, we will develop upper and lower bounds on k_O as well as approximate the best attacker strategy through theoretical and experimental approaches.

4.2 Estimating bounds on all attackers

In Section 4.1 we derived a general formula to compute bounds for an arbitrary but fixed combination of victim and attacker mobility patterns. In this subsection we want to derive a general result on a victim that is following a random walk in one dimension against an arbitrary attacker. This will give an intuition on lower and upper bound for general attackers that we will study in two dimensions as well in the following subsection.

In this section, we derive upper and lower bounds on k_O by analyzing the matrices $B^{(k)}$. We show that "for a search space of size n, $\lfloor \frac{n}{3} \rfloor - 1 \leq k_O \leq \lfloor \frac{n}{2} \rfloor$" (see Corollary 1). To achieve this goal, we first estimate the values of $B_j^{(k)}$. For a one-dimensional search space of size n, Bob follows a transition matrix P with initial position vector $B^{(0)}$:

$$B^{(0)} = \left[\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right], \quad P = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & \ldots & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{bmatrix}$$

The probabilities of Bob in each position at step i is given by $B^{(i)} = B^{(0)}P^i$:

$$\begin{aligned} B^{(1)} &= \left[\frac{1}{2n}, \frac{3}{2n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{3}{2n}, \frac{1}{2n}\right], \\ B^{(2)} &= \left[\frac{3}{4n}, \frac{1}{n}, \frac{5}{4n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{5}{4n}, \frac{1}{n}, \frac{3}{4n}\right], \\ B^{(3)} &= \left[\frac{1}{2n}, \frac{11}{8n}, \frac{1}{n}, \frac{9}{8n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{9}{8n}, \frac{1}{n}, \frac{11}{8n}, \frac{1}{2n}\right], \\ B^{(4)} &= \left[\frac{11}{16n}, \frac{1}{n}, \frac{5}{4n}, \frac{1}{n}, \frac{17}{16n}, \frac{1}{n}, \ldots, \frac{1}{n}, \frac{17}{16n}, \frac{1}{n}, \frac{5}{4n}, \frac{1}{n}, \frac{11}{16n}\right], \end{aligned} \tag{11}$$

We have the following observation:

Lemma 1. *For $k \in \mathbb{Z}_{\geq 0}$,*

$$\begin{cases} \frac{1}{n} \leq B_j^{(k)} \leq \frac{3}{2n} & 1 \leq j \leq n-2 \\ \frac{1}{2n} \leq B_j^{(k)} \leq \frac{3}{4n} & j = 0, n-1 \end{cases}.$$

Proof. See Appendix A. □

The above bounds on $B_j^{(k)}$ helps us to get upper and lower bounds on k_O. Next we show a lower bound of k_{lower}, which gives a lower bound for k_O.

Proposition 1.

$$k_O \geq k_{lower} \geq \lfloor \frac{n}{3} \rfloor - 1.$$

Proof. By Lemma 1, for any k,

$$\sum_{i=0}^{k} \max_j B_j^{(i)} \leq \frac{3}{2n} \cdot (k+1) = \frac{3k+3}{2n}.$$

If $k < \lfloor \frac{n}{3} \rfloor - 1$, $\sum_{i=0}^{k} \max_j B_j^{(i)} < \frac{1}{2}$. Thus by definition, we must have $k_{lower} \geq \lfloor \frac{n}{3} \rfloor - 1$. □

By the definition of k_O, for any given attacker strategy $\mathcal{A}$, $k_O \leq k_{\mathcal{A}}$. Next, we construct a specific attacker strategy $\mathcal{A}_{jp}$ and prove an upper bound for $k_{\mathcal{A}_{jp}}$, to obtain an upper bound for k_O.

Lemma 2. *For any $n \geq 4$, there exists a strategy $\mathcal{A}_{jp}$ such that $k_{\mathcal{A}_{jp}} \leq \lfloor \frac{n}{2} \rfloor$.*

Proof. First we notice that $P_{ij}^{\ell} = 0$ if $j - i > \ell$ due to our construction of the random walk P.

(1) Let $n \geq 4$ be even, consider the attacker strategy $\mathcal{A}_{jp}$ with the first $\frac{n}{2}$ positions given by $\mathcal{A}_0 = 0, \mathcal{A}_1 = 2, \mathcal{A}_2 = 4, \ldots, \mathcal{A}_m = 2m, \ldots, \mathcal{A}_{\frac{n}{2}-1} = n-2$. For all $0 \leq i < j \leq \frac{n}{2} - 1$,

$$\mathcal{A}_j - \mathcal{A}_i = 2(j-i) > j - i.$$

Then for $0 \leq i < j \leq \frac{n}{2} - 1$,

$$\begin{aligned} \Pr\left(F_i \cap F_j\right) &= \Pr\left(\mathcal{B}_i = \mathcal{A}_i \text{ and } \mathcal{B}_j = \mathcal{A}_j\right) \\ &= \Pr\left(\mathcal{B}_i = \mathcal{A}_i\right) \Pr\left(\mathcal{B}_j = \mathcal{A}_j | \mathcal{B}_i = \mathcal{A}_i\right) \\ &= B_{\mathcal{A}_i}^{i} P_{\mathcal{A}_i \mathcal{A}_j}^{j-i} = 0. \end{aligned}$$

Together with Lemma 1 we have

$$\begin{aligned} \Pr\left(E_{\frac{n}{2}-1}\right) &= \sum_{i=0}^{\frac{n}{2}-1} \Pr\left(F_i\right) \\ &= B_0^{(0)} + B_2^{(1)} + \cdots + B_{n-2}^{\left(\frac{n}{2}-1\right)} \\ &\geq \sum_{i=0}^{\frac{n}{2}-1} \frac{1}{n} = \frac{1}{n}\frac{n}{2} = \frac{1}{2}. \end{aligned}$$

Hence $k_{\mathcal{A}_{jp}} \leq \frac{n}{2} - 1$.

(2) Let $n \geq 5$ be odd, consider the attacker strategy $\mathcal{A}_{jp}$ with the first $\lfloor \frac{n}{2} \rfloor + 1$ positions given by $\mathcal{A}_0 = 0, \mathcal{A}_1 = 2, \mathcal{A}_2 = 4, \ldots, \mathcal{A}_m = 2m, \ldots, \mathcal{A}_{\lfloor \frac{n}{2} \rfloor} = n-1$. Similarly we can prove $\Pr\left(F_i \cap F_j\right) = 0$ for $0 \leq i < j \leq \lfloor \frac{n}{2} \rfloor$. Together with Lemma 1 we have

$$\begin{aligned} \Pr\left(E_{\lfloor \frac{n}{2} \rfloor}\right) &= \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \Pr\left(F_i\right) = B_0^{(0)} + B_2^{(1)} + \cdots + B_{n-1}^{(\lfloor \frac{n}{2} \rfloor)} \\ &\geq \frac{1}{2n} + \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} \frac{1}{n} = \frac{1}{n}\left(\left\lfloor \frac{n}{2} \right\rfloor + \frac{1}{2}\right) = \frac{1}{2}, \end{aligned}$$

and hence $k_{\mathcal{A}_{jp}} \leq \lfloor \frac{n}{2} \rfloor$.

□

Recall the notation:

$$k_{\mathcal{A}} = \min\{k : \Pr\left(E_k\right) \geq \frac{1}{2}\}, \qquad k_O = \min_{\mathcal{A}} k_{\mathcal{A}}$$

Corollary 1. *For a one-dimensional search space of size n,* $\lfloor \frac{n}{3} \rfloor - 1 \leq k_O \leq \lfloor \frac{n}{2} \rfloor$.

We note that the above Corollary agrees with our simulation results in Figure 3 (when $\alpha = 1e+13$, which is an almost uniform distribution for $B^{(0)}$).

We used the bounds in Lemma 1 to approximate $\sum_{i=0}^{k} \max_{0 \leq j \leq n-1} B_j^{(i)}$ in Proposition 1. We also derived the lower bound of k_{lower}, which then gave lower bounds on k_O. From Equation (11), we can see $\max_{0 \leq j \leq n-1} B_j^{(k)}$ decreases when k increases, which means the upper bounds on $B_j^{(k)}$ derived in Lemma 1 can be tightened for larger values of k. Thus we expect the lower bound for k_O in Corollary 1 to be higher than $\lfloor \frac{n}{3} \rfloor - 1$.

Similarly, the upper bound $\lfloor \frac{n}{2} \rfloor$ for k_O was derived through a lower bound on $B_j^{(k)}$. However, from Equation (11) we see that $B_j^{(k)}$ can achieve higher values than the lower bounds, which suggests that the specific attacker strategy $\mathcal{A}_{jp}$ described in Lemma 2, $k_{\mathcal{A}_{jp}}$ is strictly smaller than $\lfloor \frac{n}{2} \rfloor$. These observations motivate the calculations of exact values of $\sum_{i=0}^{k} \max_{0 \leq j \leq n-1} B_j^{(i)}$ and $k_{\mathcal{A}_{jp}}$, which yield tighter bounds on k_O.

4.3 Computational bounds

In order to have a better understanding of the values k_O for different search spaces, and given the challenges to analytically prove results for arbitrary attackers in two dimensions, we have implemented formula (9) to calculate $\Pr\left(E_k\right)$ for different attacker strategies. We have also implemented the calculation of $\sum_{i=0}^{k} \max_j B_j^{(i)}$ in Equation (3) to obtain k_{lower}. In the following we present the evaluation results for two dimensions. The results were obtained by running our algorithms (see Appendix B) on a virtual machine with 100 GB of RAM and 32 cores running at 2 GHz each.[1]

The crucial factor that decides the value of k_O is the size of the search space. In Figure 2, we give a plot of k_{lower}/n^2 vs. n, where k_{lower} is a lower bound on k_O obtained using Algorithm 7 (in Appendix B) for a two-dimensional search space of size n^2. That is, in this part we consider the search space to be in the shape of a square (i.e. $m = n$).

A two-dimensional analogue to attacker strategy $\mathcal{A}_{jp}$ in dimension one would be attacker strategy, denoted by $\mathcal{A}_{jp2}$, where the attacker follows the route $(1,1) \to (1,3) \to \cdots \to (1,n-2) \to (3,1) \to (3,3) \ldots$, i.e. $\mathcal{A}_0 = n+1$, $\mathcal{A}_1 = n+3$, $\ldots, \mathcal{A}_{\frac{n-1}{2}} = 2n-2, \mathcal{A}_{\frac{n+1}{2}} = 3n+1, \mathcal{A}_{\frac{n+3}{2}} =$

[1] Code is available at https://github.com/nicslabdev/proximityattacks.

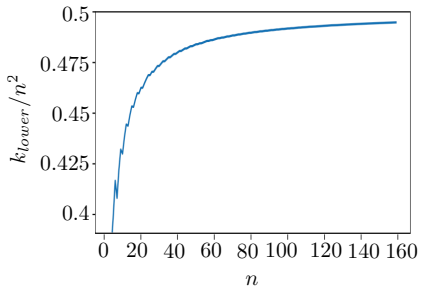


Fig. 2. k_{lower}/n^2, where k_{lower} is a lower bound of k_O for a two-dimensional search space of size n^2

Table 1. $k_{\mathcal{A}_{jp2}}/n$, where $\mathcal{A}_{jp2}$ is the attacker strategy described above

n	$k_{\mathcal{A}_{jp2}}$	$k_{\mathcal{A}_{jp2}}/n^2$
4	11	0.6875
5	15	0.6
6	22	0.6111
7	27	0.5510

$3n+2, \ldots$ for n odd and $(1,1) \rightarrow (1,3) \rightarrow \cdots \rightarrow (1,n-1) \rightarrow (3,1) \rightarrow (3,3) \ldots$, i.e. $\mathcal{A}_0 = n+1$, $\mathcal{A}_1 = n+3$, $\ldots, \mathcal{A}_{\frac{n}{2}} = 2n-1, \mathcal{A}_{\frac{n+2}{2}} = 3n+1, \mathcal{A}_{\frac{n+4}{2}} = 3n+2, \ldots$ for n even. In Table 1 we give the values of $k_{\mathcal{A}_{jp2}}/n^2$ vs. n.

5 PRACTICAL ATTACKER STRATEGIES

In the previous section we derived theoretical bounds for the optimal attacker strategy. The upper bound provides us with a constructive strategy, which we briefly discuss in this section and name it the *Linear Jumping Strategy* (LJS). Unfortunately, the lower bound estimate does not help to find the optimal strategy. While we could implement an exhaustive search to find optimal strategies, that approach is impractical and computationally expensive. In particular, for n locations, we need to evaluate the $k_{\mathcal{A},p}$ for $n^{\lceil \frac{n}{2} \rceil}$ different strategies . This is infeasible even for moderate values of n. Instead, we introduce a (locally) greedy strategy, which is computationally cheaper and allows us to perform simulations on a larger range of settings. We call it the *Greedy Updating Attack Strategy* (GUAS).

In the following we describe these strategies in detail and provide algorithms to compute the expected effort (i.e., number of queries) an attacker needs to perform to locate a victim using such strategies.

5.1 Linear Jumping Strategy

An attacker following a linear jumping strategy sequentially selects every second location in the search space regardless of the initial locations distribution of the victim. In other words, the attacker does not use information on the victim to guide her moves. Given uniform initial distributions and a random walk mobility model of the victim, this strategy is expected to meet the upper bound cost as discussed in the previous section.

The pseudocode that describes how to calculate the effort of the attacker to find the victim with probability of at least 0.5 is described in Algorithm 1. The runtime of this algorithm is linear in n, where n is the size of the search space.

Algorithm 1: Expected effort of Linear Jumping Strategy (LJS)

Result: Number of queries that Alice needs to perform to locate Bob with probability of at least 0.5.

Input: MAX_QUERIES, initial distribution of Bob $B^{(0)}$, transition matrix P

```
// Initialize number of queries and success probability
success = 0;
i = 1;
B̂^(0) = B^(0);                    // Initialize conditional probability vector for Bob
while i < MAX_QUERIES do
    A_i = 2 * (i − 1);             // Attacker strategy: jump to every second position
    // Check if Pr(E_k) ≥ 1/2
    success = success + (1 − success) * B̂^(i−1)_{A_i};
    if success ≥ 0.5 then
        return i;                                  // Minimal number of steps found
    end
    // Update Bob's location probabilities
    B̂^(i−1)_{A_i} = 0;                              // Not found thus set to empty
    B̂^(i−1) = normalize(B̂^(i−1));
    B̂^(i) = B̂^(i−1) · P;                  // Bob's location probability in next step
    i = i + 1;
end
return ERROR;              // Attacker was unable to locate Bob within MAX_QUERIES
```

Algorithm 1 receives as input the initial probability distribution of Bob $B^{(0)}$ together with the transition matrix P. A limit on the number of queries is also input to prevent the algorithm from running indefinitely. The attacker is assumed to start at position 0 and, at each step, the algorithm calculates the success probability $\Pr(E_k)$ based on the probability of Bob being at the same location as the attacker. In case the probability is at least 0.5, it ends returning the number of queries that were necessary to reach that value, otherwise the algorithm recalculates Bob's probability vector taking into account that the attacker could not find him in the last visited location. The vector $\hat{B}^{(i)}$ is normalized to ensure that the sum of its elements is equal to 1.

Note that it is easy to generalize this algorithm so that the attacker starts jumping from an arbitrary position and not always from position 0. It is also possible to use a parameter to indicate a success probability threshold other than 0.5.

5.2 Greedy Updating Attack Strategy

An attacker using the greedy updating attack strategy guides her moves (queries) based on estimates of the victim's location. In particular, Alice selects to jump to the most likely location of Bob according to her estimation.

Algorithm 2 presents the pseudocode used to calculate the effort of a GUAS attacker. The algorithm takes into account that Alice has some assumed initial distribution of Bob's locations $\tilde{B}^{(0)}$, which may be Bob's true initial location distribution $B^{(0)}$ or a different one, to calculate where

Algorithm 2: Expected effort of Greedy Updating Attack Strategy (GUAS)

Result: Number of queries that Alice needs to perform to locate Bob with probability of at least 0.5.

Input: MAX_QUERIES, $B^{(0)}$, P, assumed initial distribution of Bob $\tilde{B}^{(0)}$

```
// Initialize number of queries and success probability
```
$success = 0$;

$i = 1$;

while $i < MAX\_QUERIES$ **do**

  $\mathcal{A}_i = \max_j \tilde{B}_j^{(i-1)}$; `// Attacker strategy: maximum likelihood estimate of B`

  `// Check if Pr(`E_k`)` $\geq 1/2$

  $success = success + (1 - success) * B_{\mathcal{A}_i}^{(i-1)}$;

  if $success \geq 0.5$ **then**

    return i; `// Minimal number of steps found`

  end

  `// Update Bob's location and estimates`

  $\tilde{B}_{\mathcal{A}_i}^{(i-1)} = 0$; `// Attacker estimates this position empty`

  $\tilde{B}^{(i-1)} = \text{normalize}(\tilde{B}^{(i-1)})$;

  $B^i = B^{(i-1)} \cdot P$; `// Bob's actual location probability in next query`

  $\tilde{B}^i = \tilde{B}^{(i-1)} \cdot P$; `// Bob's estimated location probability in next query`

  $i = i + 1$;

end

return ERROR; `// Attacker was unable to locate Bob within MAX_QUERIES`

Alice jumps. In addition, the true initial distribution $B^{(0)}$ is used to check the success probability of Alice based on where she decides to jump.

At each time step i, Alice keeps her current estimate of Bob's location as $\tilde{B}^{(i)}$. Alice uses $\tilde{B}^{(i)}$ to guide her decisions and to keep track of locations she found to be empty previously. That means that $\tilde{B}^{(i)}$ depends on the actual choices of the attacker, while $B^{(i)}$ just depends on $B^{(0)}$ and P.

Alice checks the most likely location of Bob. If Alice succeeds, then we are done; if Alice does not find Bob, she updates $\tilde{B}^{(i)}$ by setting the probability of the location checked in the current query to be 0 and re-normalizing $\tilde{B}^{(i)}$. Thus, the following values of $\tilde{B}^{(i+1)}$ will be computed under the condition that the victim was not at the location tested in query i and earlier.

The runtime of this algorithm is quadratic in n because for each of the up to $\lceil \frac{n}{2} \rceil$ queries, we need to find the minimal value of $\tilde{B}^{(i-1)}$, which is of size n.

In Section 7, we evaluate both LJS and GUAS for a random walk transition pattern and for a real dataset. For the real dataset, we derive a transition matrix that is consistent with the restrictions imposed by the road topologies.

6 ROADMAP-BASED TOPOLOGIES

In practice, human mobility does not follow a random walk pattern and presents a certain degree of determinism. This is to a large extent conditioned by the underlying map (e.g., city roads, lakes, etc.), which results in visiting some particular locations to be highly unlikely to be visited.

Since we formalize the mobility pattern of the target as a transition matrix P where cell (i, j) indicates the probability of moving from location i to location j in one step, we can encode the restrictions imposed by the map into P. Algorithm 3 describes the process of computing the transition matrix P for a given street map.

The algorithm takes the map represented as a graph G, which is in essence how street maps are represented in projects like OpenStreetMap,[2] with V being a set of specific points on the earth surface and E indicating the street connections between every two points. The algorithm also receives the map bounds B, that is the maximum and minimum coordinate points that define the map area, and the cell precision cp, which determines the number of cells that will result from dividing this map area. First, it initializes P as a square matrix with as many rows and columns as the number of cells resulting from diving the map. Then, for each vertex of the graph, we obtain all vertexes connected with it by an edge. Finally, the transition matrix is updated with the same probability for all neighbours. To do so, we need to find the row corresponding to v and the columns of each neighbor n. This can be done by means of the projection function defined in previous sections.

Algorithm 3: Compute transition matrix P from a map

Input: Map $G(V, E)$, map bounds B, cell precision cp

```
// Initialize transition matix P
[nr, nc] = discretize_area(B, cp) ;                    // Divide map into cells
P = zeros(nr * nc, nr * nc) ;                  // One row and one column per cell

foreach v ∈ V do
    x = cell_proj(v, nc) ;                              // Get the row of v in P
    neighs = adjacencies(v) ;                     // Get v's neighbouring points
    foreach n ∈ neighs do
        y = cell_proj(n, nc) ;                        // Get the column of n in P
        P[x, y] = 1/size(neighs) ;            // All neighbours are equally likely
    end
end
return P
```

The result of Algorithm 3 is a transition matrix that encodes only information about the streets and for which all neighbouring points of a particular location are equally likely to be visited from it. Although the utility of this transition matrix is not optimal, it can be improved by incorporating other sources of information. For example, there are some areas of the city which are busier, roads which are more congested, and intersections where turning is more likely.

On the other hand, Algorithm 4 computes a transition matrix P from a trajectory dataset $T = (ID, LON, LAT, TS)$ consisting of a number of timestamped locations for vehicles/persons with different identifiers. A set of time-consecutive locations for a given identifier defines a trajectory. In addition to the trajectory dataset, Algorithm 4 also receives as input a cell precision cp and the map bounds B for which the transition matrix P is to be generated.

After initialization of the transition matrix P, done as in the previous algorithm, for each identifier in the trajectory dataset we obtain all visited locations and for each of these locations, we get all the

[2]https://www.openstreetmap.org/

Algorithm 4: Compute transition matrix P from trajectory data

Input: Trajectory dataset T, map bounds B, cell precision cp

```
// Initialize transition matrix P
[nr, nc] = discretize_area(B, cp);
P = zeros(nr * nc, nr * nc);

foreach id ∈ T.ID do
    L = locations(id, T) ;                    // Get all the locations for this id
    foreach l ∈ L do
        x = cell_proj(l, nc) ;                                  // Get the row of l in P
        [Vp, freq] = next(l) ;     // Get points visited from l and their frequency
        for i = 1, ..., size(Vp) do
            y = cell_proj(Vp[i], nc) ;      // Get column of the visited point in P
            P[x, y] = freq[i]/sum(freq);          // Transition's relative frequency
        end
    end
end
return P
```

points that are visited next and how often. This is possible since points are timestamped. Finally, we update the right positions of P with the relative frequency of the transitions.

Note that the dataset might include trajectories from multiple identifiers and thus the resulting transition matrix would define the busiest roads or turnings in the area. In case the trajectory data belongs to a single identifier, the transition matrix would encode the driving habits of that individual, which can be used to predict future locations of that particular target.

The transition matrix resulting from Algorithm 3 and the one resulting from Algorithm 4 can be combined to include in a single matrix P information on the road topology as well as information on user mobility patterns. In this way, the final transition matrix is the result of different information layers. One obvious benefit of this is that if any of the information layers change, either because new roads become available or the mobility patterns change (e.g., weekdays or weekends), a new transition matrix can be generated according to the new situation. Another benefit is that one may have different individual matrixes and combine them as required for a particular goal.

The algorithm for combining matrices is trivial if the bounds and precision used in both Algorithms 3 and 4 are the same. Such algorithm would consist of adding the matrixes together and then normalizing the resulting matrix.

7 EVALUATION

In this section we evaluate LJS and GUAS for different initial locations of Bob. First, we explain the set up for our experiments. Then, we present the results of our strategies for a transition matrix defining a random walk and for a transition matrix derived from a real trajectory dataset.

7.1 Experiments setup

We consider search spaces in one an two dimensions with variable sizes ranging from 100 to 5000 points. In case we want to simulate a 10×10 search space (two dimensions), we project it onto a one-dimensional search space of 100 points. Unless otherwise stated, in our simulations, we

also assume that the attacker knows the initial distribution of the victim's location $B^{(0)}$. When the attacker does not know the distribution of the victim, she will assume a uniform distribution.

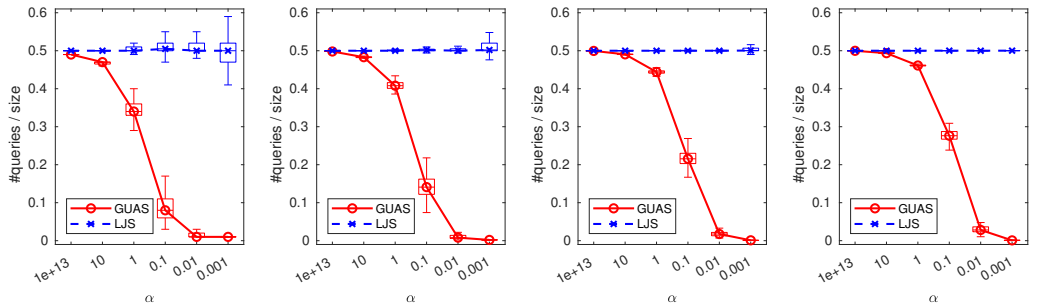
Note that in Sect. 4 we developed an analytical framework that allows one to define an arbitrary distribution for the victim's initial position. However, in order to gain an intuition on the bounds, we often chose a uniform distribution under the assumption that the attacker did not have previous knowledge on the victim and the search space did not favor any particular starting point. This could be the case if, for example, the search space is a densely populated area and the victim could be potentially anywhere. In this Section however, we account for knowledge on the initial distribution by considering a family of Dirichlet distributions with varying parameters.

We run 100 simulations for each of search space size and initial location probability distribution for $B^{(0)}$, which is represented by a Dirichlet distribution with concentration parameter α. For our simulations we consider the following values of α: 0.001, 0.01, 0.1, 1, 10, 1e+13. A positive α value of almost zero indicates an initial distribution with very high likelihood in one location, and almost zero in all others. With increasing α values, $B^{(0)}$ is *closer* to being uniformly distributed.

The code for the simulations was written in Python and simulations were performed on a Core i3-4005U CPU@1.70GHz with 8GB RAM.[3]

7.2 Random Walk Mobility

This section presents the results of using LJS and GUAS against a mobile target that follows a random walk mobility model, which is represented by a transition matrix P like the one in Section 4.2. We show simulations results for search spaces of 100, 500, 2000 and 5000 points. A similar trend was exhibited for other search space sizes within that range.



(a) Space size = 100 (b) Space size = 500 (c) Space size = 2000 (d) Space size = 5000

Fig. 3. GUAS and LJS results for random walk mobility for different search space sizes with success probability of 0.5. Assume attacker knows victim's initial distribution.

The results in Figure 3 represent the effort of the attacker to successfully locate Bob with a probability of at least 0.5 under various initial distributions for Bob. Note that the results are normalized with respect to the search space size to make it easier to compare results. Moreover, we choose a box-plot representation to give a better intuition on the sparsity of the results. The upper and lower borders of the box represent the upper and lower quartiles, and the bar in the box is the median. The upper and lower whiskers represent the maximum and minimum number of queries in all simulations, excluding outliers, which are more than 1.5 times beyond the upper and lower quartiles.

[3]Code is available at https://github.com/nicslabdev/proximityattacks.

Results and discussion. We observe that with a large concentration parameter α, which leads to an *almost* uniform distribution of the initial position, both GUAS and LJS behave similar to the expected upper bound of an optimal attacker against a random walk victim as discussed in Sect. 4.3. Interestingly, LJS still performs similar to the upper bound of an optimal attacker even when the initial distribution is not uniform, which was one of the assumptions in the analysis performed in 4.3. With decreasing concentration parameter value, the initial distribution becomes less uniformly distributed. Assuming that the attacker is aware of the exact initial distribution, GUAS becomes more effective for such non-uniform initial distribution. This is true regardless of the size of the search space, as shown in Figure 3.

7.3 Roadmap-based Mobility

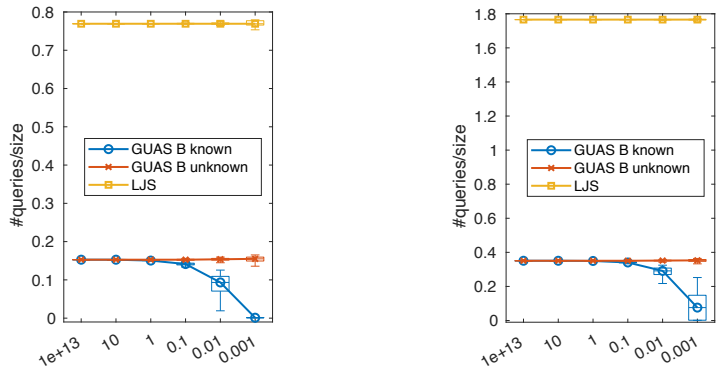
We also performed simulations with trajectory data from the T-Drive dataset released by Microsoft [22, 23]. The dataset comprises GPS trajectories of 10357 taxis between Feb. 2 and Feb. 8 2008 within the city of Beijing, around 15 million points and a total of trajectories that together add up to 9 million kilometers. The average sampling interval is around 177 seconds with a distance of 623 meters.

Although it would be interesting to also perform a roadmap-based mobility analysis for other type of mobility patterns (for instance walking individuals), we believe the T-drive dataset is representative of a real-life scenario where mobility is constrained by a city's topology. In contrast to a dataset where subjects (i.e., cars or persons) have more stable mobility patterns, for instance driving from home to work, the T-drive dataset has subjects that behave more randomly, given that they do not chose the same routes day after day. However this randomness is irrelevant from the point of view of an attacker that has no previous knowledge on a victim's mobility patterns beyond the city's topology.

Using Algorithm 4, defined in Section 6, we build up the transition matrix P_{taxi}. This transition matrix is obtained from trajectory data within the third ring of Beijing. To do so, we establish suitable map bounds for that area and then discretize it in grid cells of 500 meters $\times$ 500 meters, which corresponds to a lat-long precision unit of around 0.005. This results in 884 different locations. For each taxi, we check which grid cell the taxi is moving to by reading the data in chronological order on a daily basis. Whenever there is a transit from, say location (or grid cell) x to y, we record this transit. After processing and recording all the transits for the first taxi, we can normalize the resulting matrix and build up a transition matrix only for this taxi. We do this for all the taxis, aggregate all the transition matrices and then normalize them to arrive at P_{taxi}.

Results. We simulated different initial distributions for the victim, and studied the case where the attacker knows these distributions and when the distributions are unknown and thus need to be guessed (see Sect. 5). If the attacker is unaware of the initial distributions for the victim, she assumes a uniform initial distribution. See Figure 4a and 4b for results with success probabilities of 0.5 and 0.8 respectively. Success is determined by how close to uniform is the victim's distribution. While for larger values of α the attacker's guess is close enough to not decrease performance significantly, if the victim's initial distribution is more uniform ($\alpha \leq 0.01$), the attacker without knowledge on the initial distribution will perform worse, while the attacker with knowledge on initial B can leverage this in the GUAS strategy to decrease the required number of guesses.

Comparing performance of GUAS and LJS in Fig. 4a and Fig. 4b, we observe that for a less structured/dynamic mobility pattern, which for example could be modeled by some highly sparse transition matrix, GUAS performs better than LJS. Our interpretation of this result is that a realistic transition matrix leads to a set of locations that are significantly more likely than others. For LJS, it



(a) Probability p=0.5 (b) Probability p=0.8

Fig. 4. Attack strategies performance on T-Drive dataset for unknown and known initial distributions with success probabilities of 0.5 and 0.8.

does not matter if the attacker knows the initial distribution since LJS does not utilize knowledge on the victim.

8 COUNTERMEASURES

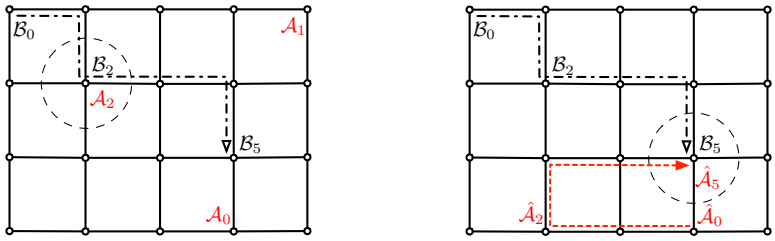
The previous section demonstrates the effort required for an attacker following either the LJS or the GUAS strategy in a real scenario derived from a dataset consisting of spatio-temporal trajectory data. We showed that an attacker using either strategy is able to consistently locate the victim with probability 0.5 in a number of steps that is linear with the size of search space, N. More precisely, an attacker is able to locate a victim in at most $\frac{N}{2}$ steps, being more efficient to use GUAS than LJS. In particular, GUAS requires $\frac{N}{6}$ queries while LJS needs more than $\frac{3}{4}N$ queries. This means that using GUAS in our Beijing dataset the attacker could locate a victim with 50% probability in 134 steps, within an area of 0.25 km^2.

In the following we introduce a simple countermeasure that requires very little effort on the side of the victim and study the effect it has on the effort that the attacker needs to perform in order to locate the victim.

8.1 Speed Constraint

The main advantage of the GUAS strategy is that, at each step, the adversary can jump to any arbitrary location of the search space. To reduce this advantage, we impose a limit on the speed at which any user of the system can move based on some reasonable bounds thereby preventing an adversary from claiming to be at a very distant location from the current one in the next time step. This solution can be easily deployed in centralized location-based services but also in decentralized deployments, as demonstrated by MaxPace [9].

When a speed constraint is implemented by the system, the attacker cannot use the original GUAS strategy described in Algorithm 2. In this case, the attacker cannot move freely in the search space and is forced to choose the next position from a limited number of positions in the neighborhood of her current location thus describing a linear walk – jumping is not allowed. The attacker then chooses the most likely position of the victim from those in her vicinity (see Figure 5). We refer to this strategy as *constrained GUAS* (cGUAS). The procedure for calculating the cost of cGUAS is basically the same as Algorithm 2 but with a slight change in the attacker strategy. Now the

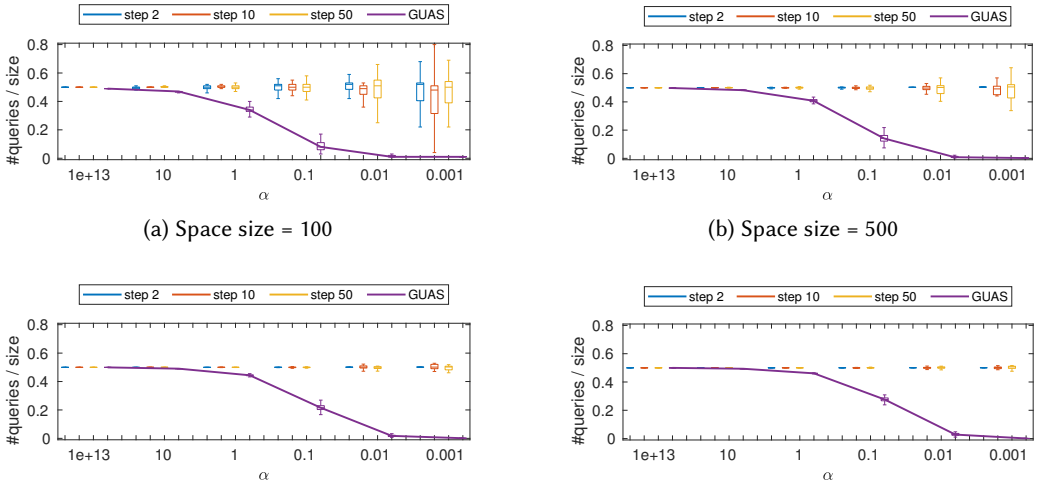


(a) Unconstrained attacker (b) Constrained attacker

Fig. 5. Comparison of the number of queries needed to find the target $\mathcal{B}$ of an unconstrained attacker $\mathcal{A}$ vs. a constrained attacker $\hat{\mathcal{A}}$

attacker cannot choose freely from $\tilde{B}^{(i)}$ but only those $\tilde{B}^{(i)}_j$ such that j is adjacent or close enough (respecting the speed constraints) to her previous location $\mathcal{A}_{i-1}$.

As in the previous section, we start by looking at the behaviour of the cGUAS strategy in a scenario where the victim moves according to a random walk mobility model. We performed 100 simulations for a search space of 100, 500, 2000 and 5000, and a concentration parameter ranging from 1e+13 to 0.001 (as in previous figures). Assuming the attacker does not know Bob's initial distribution, we can let the attacker be at any random location initially. Without loss of generality, we place attacker at position 5 of the search space. The results are presented as boxplots in Figure 6.



(a) Space size = 100 (b) Space size = 500

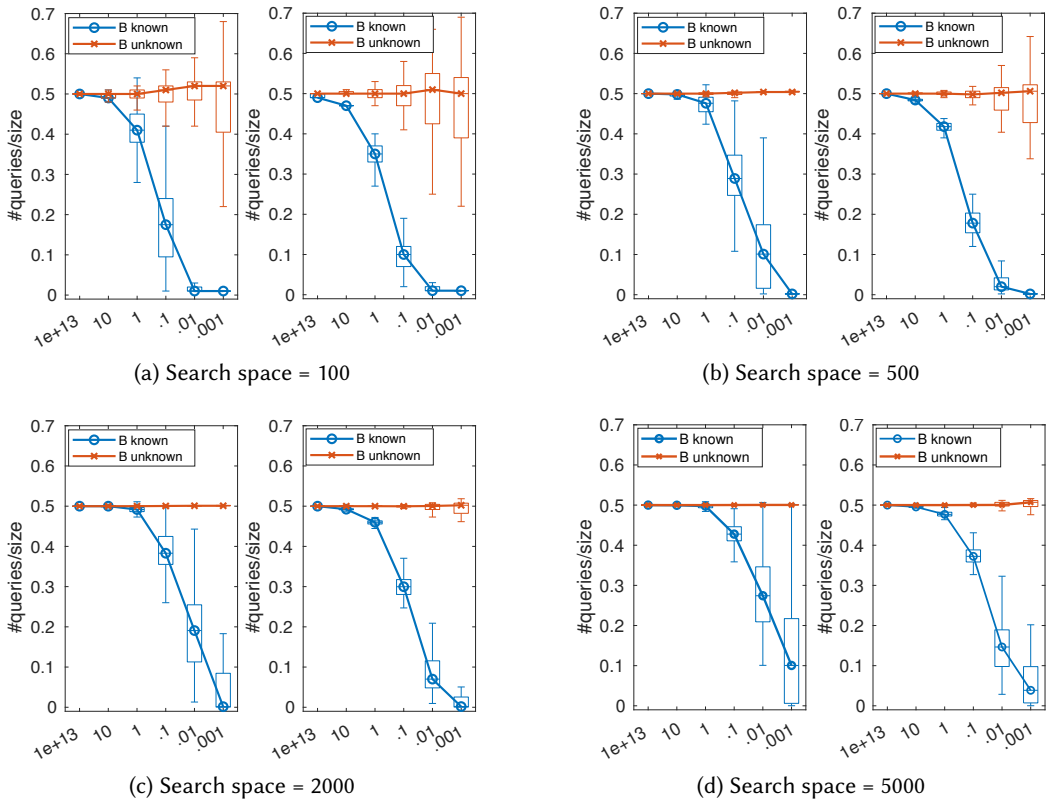
(c) Space size = 2000 (d) Space size = 5000

Fig. 6. GUAS and cGUAS with different step constraints for random walk mobility and success probability of 0.5.

First, it is worth noting that when the concentration parameter α is large, cGUAS performs similar to the original GUAS and LJS. This makes sense since the attacker can only choose to jump to locations which are within a two-step distance of her current location and most of these locations are likely to have a similar probability since a large α represents an initial distribution close to

uniform. Second, as the initial distribution becomes sparsely distributed and more locations have very low probabilities, which happens when α approaches 0, it becomes increasingly difficult for the cGUAS attacker to locate Bob. This is exactly the opposite behaviour when the attacker is capable of using the unconstrained version of GUAS as the attacker can freely jump to the location with the highest probability regardless of the distance.

Also note that the variance of attacker's effort becomes much larger in cGUAS. This is due to the fact that in some cases, if the attacker is lucky enough, he/she may start at a position close to the victim and the victim also does not take long strides from the previous location (random walks tend to stay close to the source). Otherwise, it will purportedly take much longer time and cost to perform the attack. Even when the attacker knows the victim's initial distribution, the speed constraint can impair attacker's performance. In fact, the attacker can not make the most use of his/her knowledge on the victim. In other words, imposing a simple constraint on the speed of the users has a significant impact on security. For example, if we compare data in Figure 3 and Figure 6, the effort of the adversary has on average increased from $0.001N$ up to $0.08N$ in the case of a search space of size 2000 and a concentration parameter $\alpha = 0.001$, which indicates that adversaries need to spend about 80 times more effort to find the target when their choices are constrained.



(a) Search space = 100

(b) Search space = 500

(c) Search space = 2000

(d) Search space = 5000

Fig. 7. cGUAS performance with step size 2 (left) and step size 50 (right) for known and unknown distribution and probability of 0.5.

To better understand how different speed constraints affect attacker's performance, we also vary the constraint step under the assumption that attacker knows about initial distribution (see Figure 7).

Again, when the victim displays a fairly random initial location distribution, the constraint steps do not have much impact. As the initial distribution becomes more informative and the constraint steps get relaxed, the attacker can make a more effective use of information about the victim and is able to conduct a more efficient attack. Even so, compared to Figure 3, the cGUAS protocol can sufficiently limit the attacker's behaviour.

Since a completely random walk may not be realistic, we also look into the performance of the proposed countermeasure on the T-Drive dataset. The results shown in Figure 8 indicate that the attacker's performance becomes substantially unstable and impaired unless attacker has perfect knowledge on victim's initial position, when in this case, the concentration factor does not have as much of an impact on the strategy and there are only small variations on the performance.

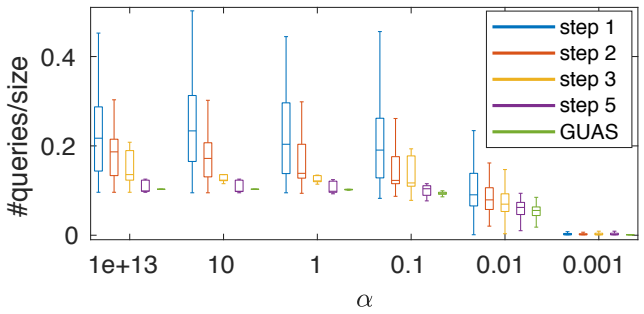


Fig. 8. GUAS and cGUAS with different step constraints for T-Drive dataset with success probability of 0.5.

Example. To gain an intuition of our derived bounds in a concrete application scenario, consider the following example. Assume a grid contains $N = 2000$ squares of size $0.25\ km^2$ and we consider a simulation were $\alpha = 0.001$. In this scenario, users of the location proximity service may query the location of other users every 3 minutes, which allows legitimate users to check whether friends are nearby at reasonable time intervals. In this setting, an attacker jumping freely on the search space by using the GUAS strategy would need to query $0.001N = 2$ times and is thus expected to wait 3 minutes to locate a victim with more than $\frac{1}{2}$ probability. Now, to increase their privacy, users may choose to set a constraint on the expected number of steps a querying party may travel between two consecutive queries. In this setting, if set to s steps per query, this means a speed of at most $s * 2 * \sqrt{0.5} * 20km/h$, given that a querying party will travel at most 2 square diagonals at each step. For $s = 2$, this is ca. $56km/h$, which is a reasonable maximum speed for a moving vehicle in a city. An attacker constrained by cGUAS needs in this case $0.08N = 160$ steps, which will take ca. 8 hours.

9 RELATED WORK

An extensive body of research has been devoted to describing and understanding mobility models for mobile ad hoc and other types of networks, such as vehicular networks [3, 10]. Typically, these papers describe mobility models where mobile nodes are independent of or dependent on each other, namely entity mobility models or group mobility models. The goal is usually to study the behavior of individual entity mobility models and help researchers decide which model is most suitable. Random mobility models are the models of choice of most authors.

Similarly, other authors have studied the *Rendezvous Problem* [21], which consists of finding an optimal strategy for two or more mobile entities who are unaware of each others' location, to meet. This problem and some slight variations of it has attracted much attention from the research

community because of its potential application to many engineering problems like the one we are considering in this paper. However, as far as we are concerned, this problem remains open [1, 4]. The main difference between rendezvous search problems and the one we are tackling in this paper is that rather than having two entities trying to find each other, here one of the entities is a victim that might not even aware of the existence of the attacker trying to find him/her.

Another line of work studies the probability for n independent entities to meet while they all follow random walk trajectories in dimension two [7, 14]. For $n = 2$, this is related to the special case when Alice chooses a random walk strategy. The difference is that we assume the entities take a step after every fixed length of time interval, while in the aforementioned papers they consider an entity takes a step after a time interval whose length follows a Poisson distribution.

In the realm of location privacy, the effort an attacker needs to locate a victim has been studied in different settings [9, 13, 16, 17]. In [9, 13], the focus was on static victim. A particular attacker model was used against moving target in [9]. Although this attacker model was shown to be optimal for a static victim, its efficiency as an attacker model for moving victim was not discussed. This is precisely the focus of our paper.

In [16, 17], the focus is on the quantification of users' location privacy by analyzing location-based applications and location-privacy preserving mechanism (LPPM). They assume an obfuscated trace (obtained by an LPPM) of the victim is available to the attacker and the goal of the attacker is to find out the real trace [17] or the real location [16] of the victim. In contrast, in this paper the attacker is only aware of the mobility model of the victim, without any knowledge of the victim's trajectory.

10 CONCLUSIONS

In this paper, we have presented a framework to reason about the expected effort of attackers attempting to locate moving targets using proximity testing. We first provide mathematical analysis for asymptotic bounds (on the search space) on the best attacker strategies under a random walk mobility model for the moving victim. We then derive two concrete strategies, and evaluate their performance over a range of parameters. The LJS strategy is found to work well for random walk mobility and close to uniform initial distribution of Bob, while the GUAS strategy requires less queries for less uniform initial distributions of Bob (which are known to the attacker). We then derive a realistic mobility model from a real dataset consisting of spatio-temporal trajectory data, and analyze the performance of our strategies. In that setting, we find that the GUAS strategy consistently requires less than $\frac{N}{6}$ queries (for success probability $p = 0.5$), while LJS requires more than $\frac{3}{4}N$, where N is the search space size. We have thus shown theoretically and practically that (using a strategy suitable to the setting), an attacker is able to locate a victim with 50% probability with at most $\frac{N}{2}$ steps. For example, using GUAS in our Beijing dataset the attacker could localize a victim with 50% probability in 134 queries, with a grid area size of $0.25km^2$. If we assume an allowed query frequency of 3 minutes, this would mean an expected waiting time of over 6 hours for the attacker. On the other hand, we have shown that a countermeasure that constrains the speed at which an attacker can move significantly increases the expected location effort. This can mean up to a 80 times delay in the expected location time on the dataset used for our evaluations.

ACKNOWLEDGMENTS

This work has been partially supported by Spanish Ministry of Science and Innovation and the Regional Ministry of Economy, Knowledge, Business and University of the Junta de AndalucÃŋa through projects PID2019-110565RB-I00 and P18-TP-3724, respectively. R. Rios was funded by the "Captación de Talento para la Investigación" fellowship from the University of Malaga.

REFERENCES

[1] Steve Alpern. 2013. *Ten Open Problems in Rendezvous Search.* Springer New York, New York, NY, 223–230. https://doi.org/10.1007/978-1-4614-6825-7_14

[2] Fan Bai and Ahmed Helmy. 2004. A survey of mobility models. Wireless Adhoc Networks, University of Southern California, USA. , 147 pages.

[3] Tracy Camp, Jeff Boleng, and Vanessa Davies. 2002. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing* 2, 5 (2002), 483–502. https://doi.org/10.1002/wcm.72

[4] L. Chen and K. Bian. 2015. The Telephone Coordination Game Revisited: From Random to Deterministic Algorithms. *IEEE Trans. Comput.* 64, 10 (Oct 2015), 2968–2980. https://doi.org/10.1109/TC.2015.2389799

[5] Jorge Cuellar, Martín Ochoa, and Ruben Rios. 2012. Indistinguishable Regions in Geographic Privacy. In *Proceedings of the ACM Symposium on Applied Computing (SAC)* (Trento, Italy) *(SAC '12)*. ACM, New York, NY, USA, 1463–1469. https://doi.org/10.1145/2245276.2232010

[6] Natasha Culzac. 2014. Egypt's police 'using social media and apps like Grindr to trap gay people'. Article on The Independent.

[7] A. Gaudillière. 2009. Collision probability for random trajectories in two dimensions. *Stochastic Processes and their Applications* 119, 3 (2009), 775–810. https://doi.org/10.1016/j.spa.2008.04.007

[8] Per A. Hallgren, Martín Ochoa, and Andrei Sabelfeld. 2015. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *Proceedings of the Annual Conference on Privacy, Security and Trust (PST)*. 1–6. https://doi.org/10.1109/PST.2015.7232947

[9] Per A. Hallgren, Martín Ochoa, and Andrei Sabelfeld. 2016. MaxPace: Speed-constrained location queries. In *Proceedings of IEEE Conference on Communications and Network Security (CNS)*. 136–144. https://doi.org/10.1109/CNS.2016.7860479

[10] J. Harri, F. Filali, and C. Bonnet. 2009. Mobility models for vehicular ad hoc networks: a survey and taxonomy. *IEEE Communications Surveys Tutorials* 11, 4 (Fourth 2009), 19–41. https://doi.org/10.1109/SURV.2009.090403

[11] Ming-Shih Huang and Ram M Narayanan. 2014. Trilateration-based localization algorithm using the lemoine point formulation. *IETE Journal of Research* 60, 1 (2014), 60–73.

[12] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. 2011. Location Privacy via Private Proximity Testing. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[13] Iasonas Polakis, George Argyros, Theofilos Petsios, Suphannee Sivakorn, and Angelos D. Keromytis. 2015. Where's Wally?: Precise User Discovery Attacks in Location Proximity Services. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 817–828. https://doi.org/10.1145/2810103.2813605

[14] Zbigniew Puchala and Tomasz Rolski. 2005. The exact asymptotic of the time to collision. *Electronic Journal of Probability* 10 (2005), 1359–1380.

[15] Jaroslav Sedenka and Paolo Gasti. 2014. Privacy-preserving distance computation and proximity testing on earth, done right. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. 99–110. https://doi.org/10.1145/2590296.2590307

[16] Reza Shokri, George Theodorakopoulos, George Danezis, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2011. Quantifying Location Privacy: The Case of Sporadic Location Exposure. In *Proceedings of the International Conference on Privacy Enhancing Technologies (PETS)* (Waterloo, ON, Canada). Springer-Verlag, Berlin, Heidelberg, 57–76.

[17] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying Location Privacy. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, Washington, DC, USA, 247–262. https://doi.org/10.1109/SP.2011.18

[18] Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2012. Protecting Location Privacy: Optimal Strategy Against Localization Attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (Raleigh, North Carolina, USA). ACM, New York, NY, USA, 617–627. https://doi.org/10.1145/2382196.2382261

[19] Max Veytsman. 2014. How I was able to track the location of any Tinder user. http://blog.includesecurity.com/2014/02/how-i-was-able-to-track-location-of-any.html

[20] Xueou Wang, Xiaolu Hou, Ruben Rios, Per Hallgren, Nils Ole Tippenhauer, and Martín Ochoa. 2018. Location Proximity Attacks Against Mobile Targets: Analytical Bounds and Attacker Strategies. In *Computer Security*, Javier Lopez, Jianying Zhou, and Miguel Soriano (Eds.). Springer International Publishing, 373–392.

[21] Richard Weber. 2012. Optimal Symmetric Rendezvous Search on Three Locations. *Math. Oper. Res.* 37, 1 (Feb. 2012), 111–122. https://doi.org/10.1287/moor.1110.0528

[22] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 316–324.

[23] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *Proceedings of the SIGSPATIAL International conference on advances in geographic information systems*. ACM, 99–108.

A PROOF OF LEMMA 1

Proof. We prove the above claims by mathematical induction. For $B^{(0)}$, the claim is true. We assume it is true for $B^{(k)}$, then for $B^{(k+1)}$

(1) $2 \le j \le n-3$, $B_j^{(k+1)} = \frac{1}{2}\left(B_{j-1}^{(k)} + B_{j+1}^{(k)}\right)$, by induction hypothesis

$$\frac{1}{n} = \frac{1}{2}\left(\frac{1}{n} + \frac{1}{n}\right) \le B_j^{(k+1)} \le \frac{1}{2}\left(\frac{3}{2n} + \frac{3}{2n}\right) = \frac{3}{2n}.$$

(2) $j = 1, n-2$, $B_1^{(k+1)} = B_0^{(k)} + \frac{1}{2}B_2^{(k)}$, by induction hypothesis

$$\begin{aligned} \frac{1}{n} = \frac{1}{2n} + \frac{1}{2}\cdot\frac{1}{n} \quad &\le B_2^{(k+1)} \quad \le \frac{3}{4n} + \frac{1}{2}\cdot\frac{3}{2n} = \frac{3}{2n}; \\ \frac{1}{n} \quad &\le B_{n-2}^{(k+1)} \quad \le \frac{3}{2n}. \end{aligned}$$

(3) $j = 0, n-1$, $B_0^{(k+1)} = B_1^{(k)}\frac{1}{2}$, by induction hypothesis

$$\begin{aligned} \frac{1}{2n} = \frac{1}{2}\cdot\frac{1}{n} \quad &\le B_1^{(k+1)} \quad \le \frac{1}{2}\cdot\frac{3}{2n} = \frac{3}{4n}; \\ \frac{1}{2n} \quad &\le B_{n-1}^{(k+1)} \quad \le \frac{3}{4n}. \end{aligned}$$

□

B ALGORITHMS FOR COMPUTATIONAL BOUNDS IN SECTION 4.3

To calculate the initial position vectors $B^{(0)}$ and the transition matrices P, we use a matrix data structure in the numpy library. For dimension one, $B^{(0)} = \left[\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right]$ is a matrix of size 1 by n and P is a matrix of size n by n that can be obtained from Algorithm 5.

Algorithm 5: Calculation of transition matrix P for random walk in one dimension

```
Input: size n;
for i = 0, 1, ..., n − 1 do
    for j = 0, 1, ..., n − 1 do
        if i + 1 = j or i − 1 = j then
            P[i][j] = 1/2;
        end
        else
            P[i][j] = 0;
        end
    end
end
P[0][1] = 1;
P[n − 1][n − 2] = 1;
return P
```

For dimension two, we assume $m = n$. The general case when $m \neq n$ can be calculated using similar methods. $B^{(0)} = \left[\frac{1}{n^2}, \frac{1}{n^2}, \ldots, \frac{1}{n^2}\right]$ is a matrix of size 1 by n^2 and P is a matrix of size n^2 by n^2 such that $P[a][b] = \Pr(\mathcal{B}_{k+1} = b | \mathcal{B}_k = a)$ following the notations for random walk mobility model in Section 2.2. This can be achieved using Algorithm 6. For each coordinate (i, j), $0 \le$

$i, j, \leq n-1$ in the n by n grid, line 4-15 counts how many points out of the four neighbors of $(i, j) : \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$ are also in the grid. And this number is assigned to the variable `counter`. The probability of Bob going from (i, j) to its neighbors in the grid is equally distributed, which is given by `1/counter`.

Algorithm 6: Calculation of transition matrix P for random walk in 2 dimensions

```
1  Input: size n;
2  for i = 0, 1, ..., n − 1 do
3      for j = 0, 1, ..., n − 1 do
4          counter= 0;
5          if i ≥ 1 then
6              counter = counter + 1;
7          end
8          if j ≥ 1 then
9              counter = counter +1;
10         end
11         if i ≤ n − 2 then
12             counter = counter +1;
13         end
14         if j ≤ n − 2 then
15             counter = counter +1;
16         end
17         if i ≥ 1 then
18             P[n ∗ i + j][n ∗ (i − 1) + j] =1/counter;
19         end
20         if j ≥ 1 then
21             P[n ∗ i + j][n ∗ i + j − 1] =1/counter;
22         end
23         if i ≤ n − 2 then
24             P[n ∗ i + j][n ∗ (i + 1) + j] =1/counter;
25         end
26         if j ≤ n − 2 then
27             P[n ∗ i + j][n ∗ i + j + 1] =1/counter;
28         end
29     end
30 end
31 return P
```

B.0.1 Algorithm for k_{lower}. Recall from Equation (4) that k_{lower} is a lower bound of k_O. Equation (3) is used to find the value of k_{lower} through Algorithm 7.

`max(Btemp)` gives the maximum value of the entries in the vector `Btemp` and `loop` is a user specified input. The output `k`$= \min_t\{t : \sum_{i=0}^{t} \max_j B_j^{(i)} \geq 0.5\} = k_{lower}$. For a one-dimensional search space of size n, following Lemma 2, we can set `loop`$= \lfloor \frac{n}{2} \rfloor$. For a two-dimensional search space of size n^2, by trial and error, we can set `loop`$= n^2$.

Algorithm 7: Algorithm for k_{lower}

```
Input: initial position vector B, transition matrix P, loop;
Btemp = B;
for k in range(loop) do
    Probability = Probability + max(Btemp);
    if Probability>=0.5 then
        return k;
    end
    Btemp = Btemp*P;
end
```

B.0.2 Algorithm for Calculating $k_{\mathcal{A}}$. Given matrices $B^{(0)}$ and P, we calculate two lists `Blist` and `Plist` such that `Bist`$[m] = B^{(m)} = BP^m$ and `Plist`$[m] = P^m$ using Algorithm 8. In the algorithm, `noofloop` is a user specified value which decides the length of the lists `Blist` and `Plist`. I is an identity matrix of the same size as P.

Algorithm 8: Algorithm for calculating `Blist` and `Plist`

```
Input: initial position vector B, transition matrix P, noofloop;
Blist=[];
Plist=[];
Blist.append(B);
Plist.append(I);
for m = 1, 2, . . . ,noofloop do
    Pm = Plist[m − 1] ∗ P;
    Plist.append(Pm);
    Blist.append(B*Pm);
end
return Blist, Plist;
```

Next, for a given attacker strategy $\mathcal{A}$, represented as a list `Alist` (`Alist`$[i] = \mathcal{A}_i$), the calculation of $k_{\mathcal{A}}$ was implemented using Algorithm 9. `Combinations(m,l,k)` is the collection of all $(l+1)-$combinations of integers between m and k in increasing order that begins with m. For example, `Combinations(1,2,4)` $= [[1,2,3],[1,2,4],[1,3,4]]$. `loop` is a user-specified value equal to the variable `loop` for calculating `Blist` and `Plist`. Note that to calculate $\Pr(E_k)$, we need both `Blist` and `Plist` to be of length at least k.

As the value of k increases, the time for calculating `Combinations(m,l,k)` ($\forall 0 \le m \le k,\ 1 \le l \le k-m$) increases exponentially. When the size of the search space increases, we need to calculate for bigger k, thus the calculation of these combinations is a bottle neck of our program. To reduce the time, we tried to pre-calculate the combinations for different values of k. But the number of those combinations grows very fast, so that even for k up to 28, we have a pre-computed file with size more than 1GB. This is why the results in Table 1 are only for small values of n.

Algorithm 9: Algorithm for calculating $k_{\mathcal{A}}$

```
Input: Plist, Blist, Alist, k, loop;
for k = 0, 1, ..., loop − 1 do
    Prob = 0;
    for m = 0, 1, ..., k do
        for l = 1, 2, ..., k − m do
            IndexList = Combinations(m, l, k);
            Sum = 0;
            for j = 0, 1, ..., length(IndexList)−1 do
                Product = 1;
                for i = 0, 1, , ..., l − 1 do
                    a = IndexList[j][i];
                    b = IndexList[j][i + 1];
                    Aa = Alist[a];
                    Ab = Alist[b];
                    Product = Product ∗ Plist[b − a][Aa, Ab];
                end
                Sum = Sum + Product;
            end
        end
        Prob = Prob + Blist[m][Alist[m]] ∗ (1 + (−1)^l ∗ Sum);
    end
    if Prob >= 0.5 then
        return k;
    end
end
```