A Realistic Outdoor Urban Pedestrian Mobility Model

Ryan Vogt[a,*], Ioanis Nikolaidis[a], Pawel Gburzynski[b]

[a]*Department of Computing Science*
2-21 Athabasca Hall
University of Alberta
Edmonton, Alberta
Canada T6G 2E8
Phone: +1 (780) 492-2285
Fax: +1 (780) 492-6393
[b]*Olsonet Communications Corporation*
51 Wycliffe Street
Ottawa, Ontario
Canada K2G 5L9

Abstract

Existing generative mobility models, used to produce mobility data for simulated agents in, e.g., wireless network simulations, suffer from a number of limitations. Most significantly, existing models are not representative of actual human movement. We introduce a new mobility model 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. Our model is implemented as the publicly available Destination-Based Space Syntax Simulator (DBS3). We use DBS3 to demonstrate which mobility model parameters affect wireless network simulations: centrality bias significantly affects network simulation results in non-grid-based urban centres, whereas the pathfinding metric affects results more in grid-based urban centres; distance decay has minimal impact on our wireless network metrics.

Keywords: Generative mobility model, Space Syntax, wireless network simulation

1. Introduction

Simulation studies of wireless network protocols require a source of mobility data for the agents in each experiment. Trace data obtained from tracking actual movement of real people is an appropriate source of such mobility data. However, it is impractical 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

*Corresponding Author
Email addresses: rvogt@ualberta.ca (Ryan Vogt), nikolaidis@ualberta.ca (Ioanis Nikolaidis), pawelg@ualberta.ca (Pawel Gburzynski)

Also available as *Simulation Modelling Practice and Theory 26 (2012) 113–134*

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,[1] 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 [1] is one such generative model. Agents moving according to this model proceed in straight lines to randomly chosen destinations 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 [2], 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 [3].

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 [4]. 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 [3]. Despite these issues, the random waypoint mobility model continues to be used [5, 6].

The key problem that this paper addresses 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 simulation 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. 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*. This paper adopts the view on how configured space influences human mobility as it is articulated in the theory of Space Syntax [7]. 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

[1] 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.

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 lots 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 the number of changes in direction of travel, or the magnitude of change in direction of travel, that we must make to move between the two destinations.

Based on these two key results, we built the Destination-Based Space Syntax Simulator (DBS3). DBS3 1.1.1 is a freely available,[2] high-performance mobility simulator for agents in outdoor urban environments. We begin in Section 2 by describing how DBS3 is distinct from related work on realistic mobility models. Section 3 describes the design of our new mobility model. It also describes the implementation details of DBS3, in particular the new multi-expansion A* search (MEA*) used to produce optimal non-Euclidean-distance pathfinding. Section 4 then verifies the correctness of the mobility model by showing high correlation between the mobility patterns produced in DBS3 and observed pedestrian patterns in downtown Edmonton. Finally, we conclude and discuss future work in Section 5.

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. [8] 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 (measured by Euclidean distance) between origin and destination pairs form the set of possible routes used by their generative model. That approach is subsequently extended [9] 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 predictibility given the current time of day [10].

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. [11] 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 interests, 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 [13] 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.

[2] Available on-request; please contact the authors by email.

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, 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" [14]. 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. The second effort, by Kostakos et al. [15] and Kostakos [16], 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.

3. Mobility Model Design and Implementation

3.1. Agents and Maps

The mobility model used by DBS3 is an instance of a random trip [2] 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 3.2, the pathfinding algorithms in Section 3.3, and simulation initialization in Section 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 paper. In general,

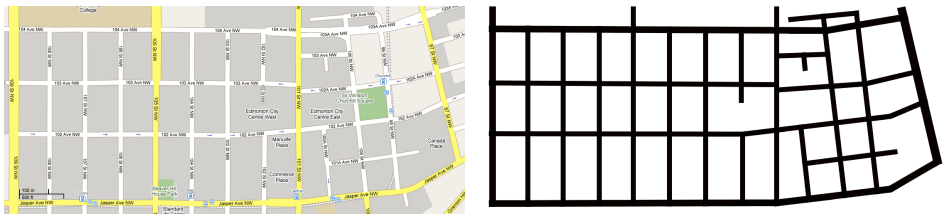

Figure 1: A map of downtown Edmonton, Canada from Google Maps 2010, and the associated street outlines used as input to DBS3.

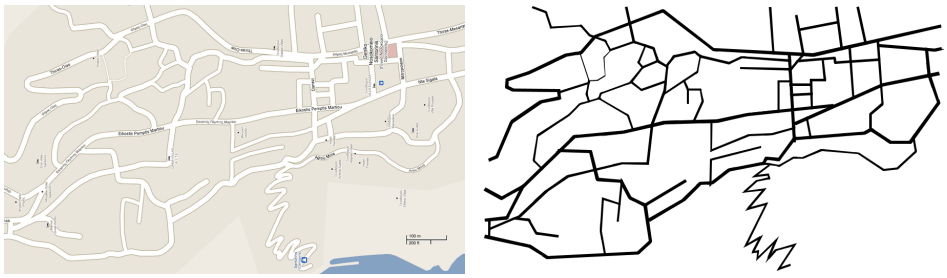

Figure 2: A map of Fira, Greece from Google Maps 2010, and the associated street outlines used as input to DBS3.

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 [17, 18, 19]. 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. [8], we represented any curved street as a sequence of straight streets intersecting at angles that are neither overly sharp nor overly shallow.

The first map used in this paper is grid-like downtown Edmonton, Canada, illustrated in Figure 1. The second map we use is the more organic and graph-theoretically interesting Fira, Greece, shown in Figure 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 when mobile, S_i, 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 $\mathcal{S}$ [21]). But more importantly, we feel that our design is more realistic, 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 [22, 23]. 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 [9]. 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 [9]. 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 $\mathcal{T}$. Complete details of how we generate bounded normal and bounded log-normal random values are presented in Appendix A, 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.

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 [24]. 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 paper is to formalize the interplay of centrality bias and distance decay into an algorithm that can be used to select destinations for agents in a mobility simulation based on their 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.

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. As such, 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

$$\mathrm{P}(s,d) = \frac{\mathrm{L}(d)}{\mathrm{I}(d)^{\alpha} \cdot \mathrm{D}(s,d)^{\delta}} \ , \tag{1}$$

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 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. 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,

$$\mathrm{D}(s,d) = \mathrm{T}(s,d) + 1 \ . \tag{2}$$

With reference to Equation 1, 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 [24], 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." 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

$$\mathrm{I}(d) = \frac{\sum_{i \mid \mathrm{T}(i,d) \leq \rho} (\mathrm{L}(i) \cdot \mathrm{D}(i,d))}{\sum_{i \mid \mathrm{T}(i,d) \leq \rho} \mathrm{L}(i)} \ . \tag{3}$$

Again with reference to Equation 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 1–3 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 1; to select a new destination, an agent calls GetNewDestination(*curLoc*), where *curLoc* is its current location.

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 1 The destination-selection algorithm in DBS3

function GETNEWDESTINATION(*curLoc*)
  $s \leftarrow$ the street on which *curLoc* is located
  $n \leftarrow$ the number of streets on the map
  for $i \leftarrow 1$ **to** n **do**
    $d \leftarrow$ street number i on the map
    $A[i] \leftarrow \mathrm{P}(s, d)$, computed as per Equation 1
  $index \leftarrow$ PROPORTIONALINDEX(A)
  $str \leftarrow$ street number *index* on the map
  $pnt \leftarrow$ a uniform random point on *str*
  return point *pnt* on street *str*

function PROPORTIONALINDEX(*array*)
  $n \leftarrow$ the length of *array*
  $sum \leftarrow \sum_{1 \le i \le n} array[i]$
  $cdf[1] \leftarrow array[1]/sum$
  for $i \leftarrow 2$ **to** n **do**
    $cdf[i] \leftarrow array[i]/sum + cdf[i-1]$
  $r \leftarrow \mathcal{U}(0, 1)$ ▷ Uniform random value in $[0, 1]$
  return $\min\{i \mid r \le cdf[i]\}$

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 paper to suggest a specific ρ value for any appreciably larger maps. It is sufficient to point out that ρ expresses what humans find acceptable walking distance, yet the notion of distance is subjective (for example, potentially dependent on the number of "features" encountered and memorized during locomotion along a given path [25]). Hence, we prefer to 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).

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* [26].

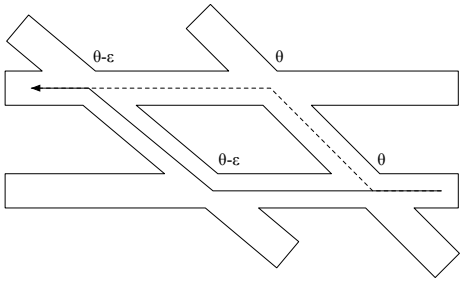

Figure 3: Two potential paths from a source to a destination, labelled with the angle changes incurred during each change in direction.

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 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 3 instead of the streets 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 equivalently 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 of all possible paths from *src* to *dst* constrained by these rules.
4. Choose path $h \in \mathbb{H}$, where h is chosen according to one of the three rules:
   (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 [27] 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 like MEA*. 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 along the chain of previous points stored at each point in the path.

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 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. In the minimal-angle variant of MEA*, if one inbound path h_1 can be reoriented (i.e., rotated) to the other path h_2, and the acquired cost of h_1 plus the cost of 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. The minimal-distance and minimal-turn variants

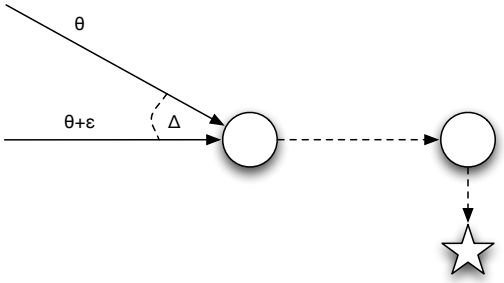

Figure 4: Two potential paths to the destination (the star) converging on a shared intermediate point (the left circle). The upper path has both a smaller acquired cost of $\theta < \theta + \epsilon$ and a smaller estimated final cost (assuming E estimates the angle change required to reach the destination 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.

of MEA* degenerate into classic A* by saving at most one inbound path to each point.

The core of the MEA* algorithm is formalized in pseudocode in Algorithms 2 and 3. 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 4. The admissible 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.
- For a minimal-angle search, E is firstly the angle change that would be incurred in turning to face directly to the destination. Ties are broken using the Euclidean length of straight lines from the crumbs 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. The pseudocode for this function has been omitted for brevity; but, for the minimal-angle search, that would be the crumb with the lowest *angleLB* value (breaking ties with the lowest *distLB* value if necessary). If the crumb

Algorithm 2 The core of the MEA* pathfinding algorithm

```
function FindPath(src, dst)
    queue, pts, cache ← {}, {}, {}
    for each street s containing point src do
        nc ← Crumb(src, s, null, dst)
        AddToCache(nc, src, s, cache)
        Add nc to queue
    while True do
        cur ← PollMinimalLB(queue)
        if cur.obsolete = true then
            continue
        else if cur.pLoc = dst then
            return ReconstructPath(cur)
        else if cur.street contains point dst then
            Add Crumb(dst, cur.street, cur, dst) to queue
        else
            for each i ∈ GetIntersections(cur, dst) do
                Expand(cur, i, dst, queue, pts, cache)

procedure Expand(cur, i, dst, queue, pts, cache)
    nextStr ← i.intersectingStreet
    for each p ∈ GetRandomPoints(i, pts) do
        nc ← Crumb(p, nextStr, cur, dst)
        for each old ∈ GetCache(p, nextStr, cache) do
            obs ← CheckObsolete(old, nc)
            if obs = ObsoleteNew then
                nc.obsolete ← True
                break
            else if obs = ObsoleteOld then
                old.obsolete ← True
                RemFromCache(old, p, nextStr, cache)
        if nc.obsolete = False then
            Add nc to queue
            AddToCache(nc, p, nextStr, cache)
```

Algorithm 3 Some shared helper functions in the MEA* pathfinding algorithm

function GetRandomPoints(i, pts)
  if $\nexists \{intr, p_1, p_2, p_3, p_4\} \in pts$ with $intr = i$ **then**
    $p \leftarrow$ a random point in i
    $p_1, p_2, p_3, p_4 \leftarrow p$ mirrored into each quadrant of i
    Add $\{i, p_1, p_2, p_3, p_4\}$ to pts
  return $\{p_1, p_2, p_3, p_4\}$ where $\{i, p_1, p_2, p_3, p_4\} \in pts$

procedure AddToCache($crumb$, $point$, $street$, $cache$)
  Add $\{point, crumb\}$ to $cache$

procedure GetCache($point$, $street$, $cache$)
  return $\{c \mid \{point, c\} \in cache\}$

procedure RemFromCache($crumb$, $point$, $street$, $cache$)
  Remove $\{point, crumb\}$ from $cache$

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 fewer turns 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 crumb's acquired Euclidean distance is less than or equal to that of the new crumb, and the new crumb will obsolete the old crumb if the new crumb 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 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. Specifically, 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

Algorithm 4 The minimal-angle versions of some MEA* helper functions

```
function Crumb(physicalLoc, street, prevCrumb, dst)
    c ← a new, empty crumb data structure
    c.pLoc ← physicalLoc
    c.street ← street
    c.prev ← prevCrumb
    c.obsolete ← False
    if c.prev = null then
        c.in ← null
    else if c.pLoc = c.prev.pLoc then
        c.in ← c.prev.in
    else
        c.in ← vector from c.prev.pLoc to c.pLoc
    c.angle ← accumulated angle change thus far
    c.dist ← accumulated distance thus far
    c.angleLB ← total angle change if going straight to dst
    c.distLB ← total distance if going straight to dst
    return c

function CheckObsolete(old, nc)
    if old.in = null then
        return ObsoleteNew
    else if nc.in = null then
        return ObsoleteOld
    Δ ← the angle between old.in and nc.in
    if (old.angle + Δ < nc.angle) or
       (old.angle + Δ = nc.angle and old.dist ≤ nc.dist) then
        return ObsoleteNew
    else if (nc.angle + Δ < old.angle) or
            (nc.angle + Δ = old.angle and nc.dist ≤ old.dist) then
        return ObsoleteOld
    return ObsoleteNone
```

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 4.

Regarding one crumb obsoleting another, it is also worth noting the following from a performance perspective.

Remark 1. If a new crumb obsoletes an old crumb during a vanilla MEA* minimal-distance, minimal-turn, or minimal-angle search, 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,

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

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 5.

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 1. Note that the runtime for the minimal-turn pathfinder was the shortest because of the extra optimization that DBS3 performs using

Algorithm 5 The StreetCut improvement to minimal-angle MEA*

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

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

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

function CheckObsolete(old, nc)
    if old.in = null then
        return ObsoleteNew
    else if nc.in = null then
        return ObsoleteOld
    Δ ← the angle between old.in and nc.in
    λ ← Euclidean distance from old.pLoc to nc.pLoc
    if (old.angle + Δ < nc.angle) or
       (old.angle + Δ = nc.angle and old.dist + λ ≤ nc.dist) then
        return ObsoleteNew
    else if (nc.angle + Δ < old.angle) or
            (nc.angle + Δ = old.angle and nc.dist + λ ≤ old.dist) then
        return ObsoleteOld
    return ObsoleteNone
```

Table 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.

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%

the precomputed number of turns between all street pairs. 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 Remark 1 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 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. More details about the conditions that can lead to post-polling obsolescences are presented in Appendix B.

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 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 t as $t \to \infty$, we would expect to see some agents moving and some agents paused, and those agents that are moving would be expected to be travelling between two destinations that are further away than the average distance between destinations on the map [28]. 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 on steady state before drawing any mobility data from the simulation [2, 3]. 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 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 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 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; so, from the point of view of a user, DBS3 simulations begin in steady state. Future work will investigate how the methods used to initialize random trip mobility models in steady state [2] could be adapted to work in DBS3.

4. Verification of DBS3

In the previous section, we described the parameters given to a DBS3 simulation on a certain map: the centrality bias and distance decay exponents, as well as the pathfinding algorithm. However, the question of what values a DBS3 user should choose remains. In this section, we will verify that when proper values are chosen for those parameters, DBS3 will generate mobility data that is highly correlated to the mobility patterns of real people in an urban environment (additional effects of these parameters on wireless networking simulations are described in Appendix C).

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 $0 \le \alpha, \delta \le 2$, to the observed distribution of pedestrians in Edmonton. This comparison is summarized in Figure 5, 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 brevity).

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.

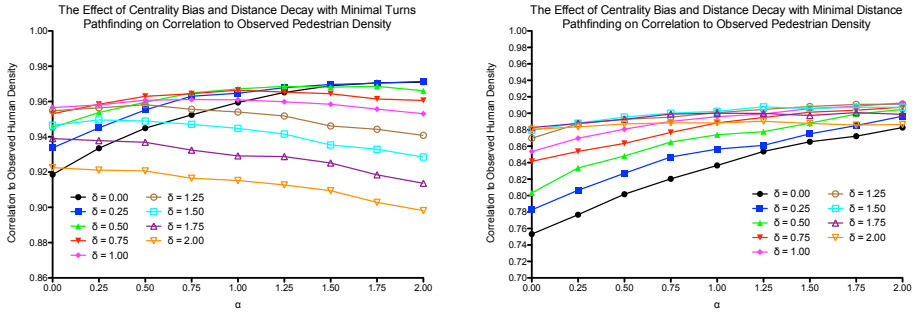

Figure 5: The effect of the centrality bias exponent, α, and the distance decay exponent, δ, on the correlation between DBS3's steady state distribution of pedestrians (using two different pathfinders) and the observed distribution of pedestrians in Edmonton.

Additionally, DBS3's mobility model is more accurate than the traditional random waypoint model with pathfinding that minimizes Euclidean distance. Setting $\alpha = \delta = 0$ eliminates 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 5, 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.

5. Conclusion

In this paper, we presented a new generative mobility model inspired by the findings of Space Syntax. This model, implemented as the publicly available DBS3 (information on how to write client programs that use mobility data from DBS3 is available in Appendix D), 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.

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 non-trivial 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.

Acknowledgements

The authors' research is funded in part by the Natural Sciences and Engineering Research Council of Canada, and iCORE and Alberta Advanced Education & Technology.

References

[1] D. B. Johnson, D. A. Maltz, Dynamic source routing in ad hoc wireless networks, Mobile Computing 353 (1996) 153–181.

[2] J.-Y. Le Boudec, M. Vojnović, Perfect simulation and stationarity of a class of mobility models, in: Proceedings of the 24th Annual IEEE Infocom Conference, pp. 2743–2754.

[3] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, Wireless Communications and Mobile Computing 2 (2002) 483–502.

[4] J. Yoon, M. Liu, B. Noble, Random waypoint considered harmful, in: Proceedings of the 22th Annual IEEE Infocom Conference, pp. 1312–1321.

[5] C. N. Ververidis, G. C. Polyzos, A routing layer based approach for energy efficient service discovery in mobile ad hoc networks, Wireless Communications and Mobile Computing 9 (2009) 655–672.

[6] X. Xiang, X. Wang, Y. Yang, Stateless multicasting in mobile ad hoc networks, IEEE Transactions on Computers 59 (2010) 1078–1090.

[7] B. Hillier, Space is the machine: a configurational theory of architecture, Cambridge University Press, 1996.

[8] J. Yoon, B. D. Noble, M. Liu, M. Kim, Building realistic mobility models from coarse-grained traces, in: Proceedings of the 4th International Conference on Mobile Systems, Applications, and Services, pp. 177–190.

[9] M. Kim, D. Kotz, S. Kim, Extracting a mobility model from real user traces, in: Proceedings of the 25th Annual IEEE Infocom Conference.

[10] C. Song, Z. Qu, N. Blumm, A.-L. Barabási, Limits of predictability in human mobility, Nature 327 (2010) 1018–1021.

[11] W.-J. Hsu, K. Merchant, H.-W. Shu, C.-H. Hsu, A. Helmy, Weighted waypoint mobility model and its impact on ad hoc networks, ACM SIGMOBILE Mobile Computing and Communications Review 9 (2005) 59–63.
[12] M. Feeley, N. Hutchinson, S. Ray, Realistic mobility for mobile ad hoc network simulation, in: Proceedings of the 3rd International Conference on Ad-Hoc Networks and Wireless, pp. 324–329.
[13] J. Kim, V. Sridhara, S. Bohacek, Realistic mobility simulation of urban mesh networks, Ad Hoc Networks 7 (2009) 411–430.
[14] N. S. Dalton, R. C. 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, pp. 359–363.
[15] V. Kostakos, T. Nicolai, E. Yoneki, E. O'Neill, H. Kenn, J. Crowcroft, Understanding and measuring the urban pervasive infrastructure, Journal of Personal and Ubiquitous Computing 13 (2009) 355–364.
[16] V. Kostakos, Space Syntax and pervasive systems, in: B. Jiang, X. Yao (Eds.), Geospatial Analysis and Modeling of Urban Structure and Dynamics, Springer, 2010, pp. 31–52.
[17] J. Peponis, J. Wineman, M. Rashid, S. Kim, S. Bafna, On the generation of linear representations of spatial configuration, Environment and Planning B: Planning and Design 25 (1998) 559–576.
[18] A. Turner, Depthmap: A program to perform visibility graph analysis, in: Proceedings of the 3rd International Symposium on Space Syntax, pp. 31.1–31.9.
[19] J. Wineman, J. Turner, S. Psarra, S. K. Jung, N. Senske, Syntax2d: An open source software platform for Space Syntax analysis, in: Proceedings of New Developments in Space Syntax Software, pp. 23–26.
[20] Google, Google maps, `http://maps.google.com/`, 2010.
[21] J. Yoon, M. Liu, B. Noble, Sound mobility models, in: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, pp. 205–216.
[22] L. F. Henderson, The statistics of crowd fluids, Nature 229 (1971) 381–383.
[23] L. F. Henderson, D. J. Lyons, Sexual differences in human crowd motion, Nature 240 (1972) 353–355.
[24] B. Hillier, S. Iida, Network and psychological effects in urban movement, Lecture Notes in Computing Science 3693 (2005) 475–490.
[25] D. R. Montello, The perception and cognition of environmental distance: Direct sources of information, in: S. C. Hirtle, A. U. Frank (Eds.), Proceedings of COSIT '97, pp. 297–311.
[26] K. Daniel, A. Nash, S. Koenig, A. Felner, Theta*: Any-angle path planning on grids, Journal of Artificial Intelligence Research 39 (2010) 533–579.
[27] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions of System Science and Cybernetics 4 (1968) 100–107.
[28] W. Feller, An Introduction to Probability Theory and Its Applications, volume 2, Wiley, New York, 2nd edition.
[29] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, H. Yu, Advances in network simulation, IEEE Computer 33 (2000) 59–67.
[30] P. Gburzynski, I. Nikolaidis, Wireless network simulation extensions in SIDE/SMURPH, in: Proceedings of the 2006 Winter Simulation Conference, pp. 2225–2233.

Appendix A. Random Number Generation

DBS3 generates random agent speeds and pause times within finite, bounded ranges $[min, max]$ according to one of three distributions: uniform, normal, or log-normal. Uniform random values are generated in the obvious fashion. Normal and log-normal distributions, however, 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 $[min, max]$.

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 trun-

cated 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 0.06134 or larger than approximately 44.31 are rejected.[3] 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 distribution's truncation.

Pseudocode for how these bounded values are generated is provided as Algorithm 6. 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 6 Generation of bounded, random values in DBS3

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

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

function GenerateLogNormal(min, max)
    lb ← 0.06134160902282682206561692291493027192911068582928926771205592947
    ub ← 44.31383316742639161294271979219893347698550976490274713986352064 04
    repeat
        g ← G(0, 1)                               ▷ Normal random with μ = 0, σ = 1
        l ← e^g
    until lb ≤ l ≤ ub
    p ← (l − lb)/(ub − lb)
    return min + (max − min) · p
```

Appendix B. Post-Polling Obsolescence in Minimal-Angle StreetCut Pathfinding

When StreetCut is disabled in MEA*, or when either the minimal-distance or minimal-turn metrics are used, any crumb that is tagged as obsolete is guaranteed either to still be in the priority queue or to have not yet been added to the priority queue. Hence, the crumb will never be expanded, improving the running time of the MEA* search. When StreetCut is enabled and minimal-angle pathfinding is performed, however, post-polling obsolescences become possible — that is, crumbs that have already been expanded out of the queue could be flagged as obsolete. This situation is not ideal, because a post-polling obsolescence represents a missed

[3] 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.

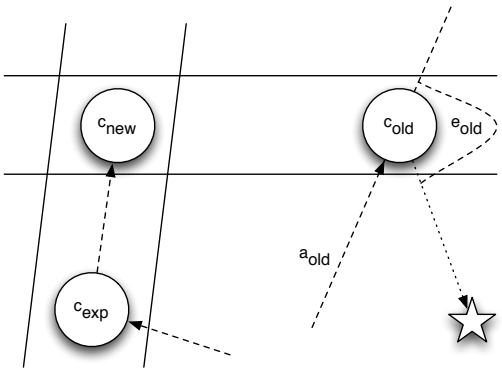

Figure B.6: The new crumb, c_{new}, is on the same street as the old crumb, c_{old}, but in a different location on the street. The star is the destination, and the angle change incurred by walking in a straight line to the destination is the admissible heuristic, illustrated for c_{old} as e_{old}.

opportunity for optimization. However, as shown in Section 3.3, post-polling obsolescences are rare in practice, and they do not affect the correctness of the search. In this section, we illustrate why post-polling obsolescences are rare.

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 B.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

$$a_{old} + e_{old} < a_{exp} + e_{exp} \tag{B.1}$$

(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, that never occurs in practice, so for simplicity we ignore that case here). Because c_{new} obsoletes c_{old}, we also know that

$$a_{new} + \Delta_{new,old} < a_{old} \ , \tag{B.2}$$

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 B.2 by noting that $a_{new} = a_{exp} + \Delta_{exp,new}$, yielding

$$a_{exp} + \Delta_{exp,new} + \Delta_{new,old} < a_{old} \ . \tag{B.3}$$

Then, we rearrange Equation B.1 to determine that

$$a_{old} < a_{exp} + e_{exp} - e_{old} \ . \tag{B.4}$$

By combining Equations B.3 and B.4, we determine that

$$a_{exp} + \Delta_{exp,new} + \Delta_{new,old} < a_{exp} + e_{exp} - e_{old} \ . \tag{B.5}$$

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

$$\Delta_{exp,new} + \Delta_{new,old} + e_{old} < e_{exp} \ . \tag{B.6}$$

While a post-polling obsolescence will occur if and only if Equations B.1 and B.2 are true, the necessary (but not sufficient) condition illustrated in Equation B.6 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.

Appendix C. Mobility Parameters and Wireless Networking Simulations

There are three choices that parameterize a DBS3 simulation: centrality bias, distance decay, and the pathfinding metric. But what effect 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, one begins 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, one begins 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:

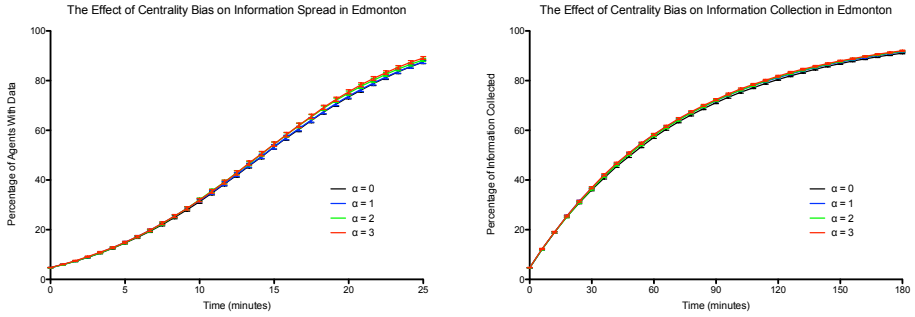

Figure C.7: The effect of the centrality bias exponent, α, on the rates of information spread and information collection in Edmonton. The distance decay exponent, δ, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

1. Of the m agents, one begins 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 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 C.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.[4] 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 C.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 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 C.9 and C.10, increasing δ

[4]For brevity, the token passing graphs have been omitted, since they were consistently similar to the information collection graphs.

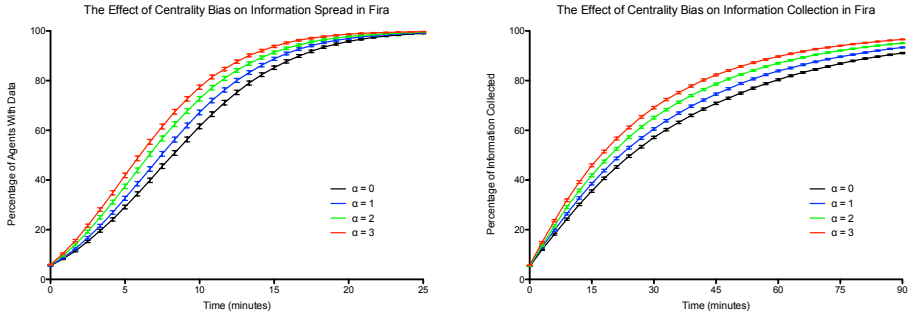

Figure C.8: The effect of the centrality bias exponent, α, on the rates of information spread and information collection in Fira. The distance decay exponent, δ, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

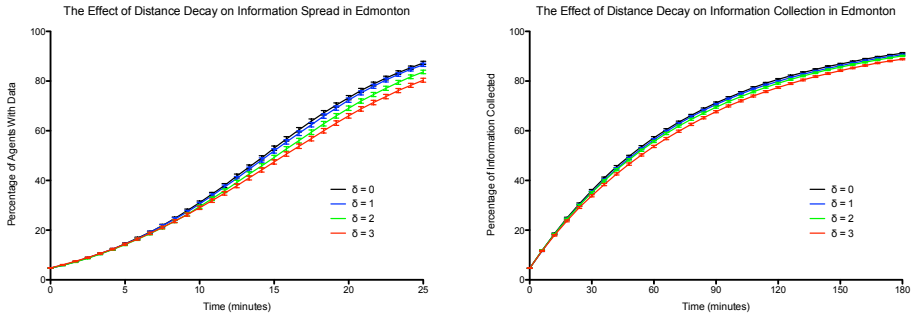

Figure C.9: The effect of the distance decay exponent, δ, on the rates of information spread and information collection in Edmonton. The centrality bias exponent, α, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

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 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 dis-

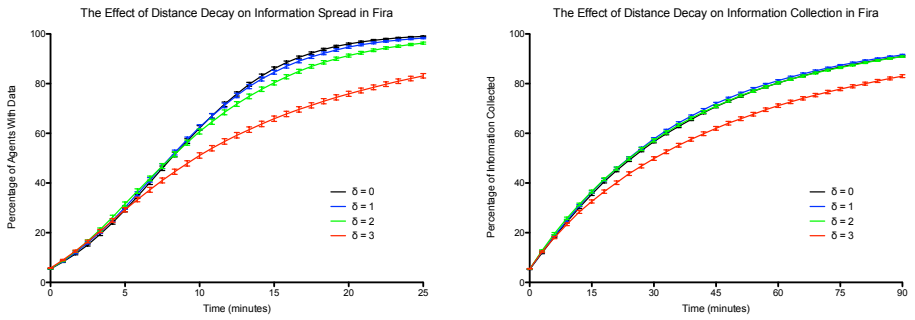

Figure C.10: The effect of the distance decay exponent, δ, on the rates of information spread and information collection in Fira. The centrality bias exponent, α, was fixed at zero. Error bars represents 95% confidence intervals around the mean.

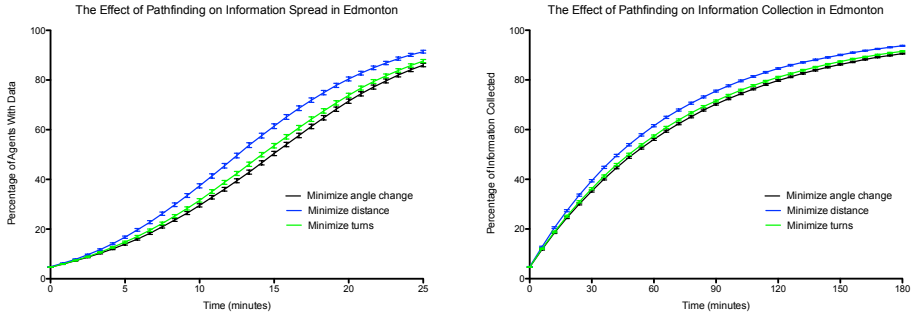

Figure C.11: The effect of the pathfinding algorithm on the rates of information spread and information collection in Edmonton. The exponents α and δ were fixed at zero. Error bars represents 95% confidence intervals around the mean over 2500 trials.

tance travelled will make many additional turns, imparting a diagonal component to the movement of an agent travelling across the map. This convergence of agents towards the central part of the grid speeds the rate of information dissemination and collection. As illustrated in Figure C.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 C.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 theorized that the two more realistic

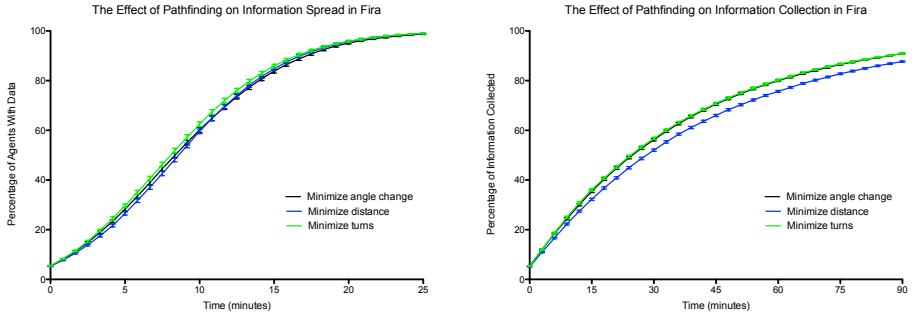

Figure C.12: The effect of the pathfinding algorithm on the rates of information spread and information collection in Fira. The exponents α and δ were fixed at zero. Error bars represents 95% confidence intervals around the mean over 2500 trials.

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 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 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. For consistency with Hillier and Iida [24], which measured pedestrian density in London by observing pedestrian flow, we eliminated the pause times from DBS3 during this steady-state distribution computation.

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 C.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 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-

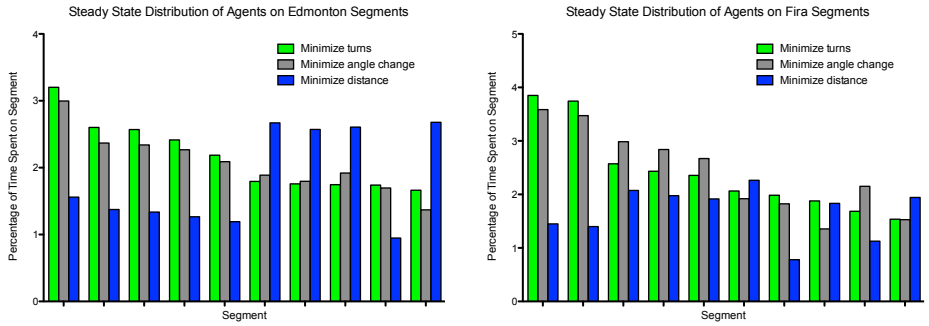

Figure C.13: The effect of the pathfinding algorithm on the steady state distribution of agents across ten fixed segments in Edmonton and Fira. The ten segments chosen were those segments inhabited by the highest percentage of agents in steady state when the minimizing-turns pathfinder was used. The exponents α and δ were fixed at zero.

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.

Appendix D. 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 [29] and SMURPH/SIDE [30], 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,

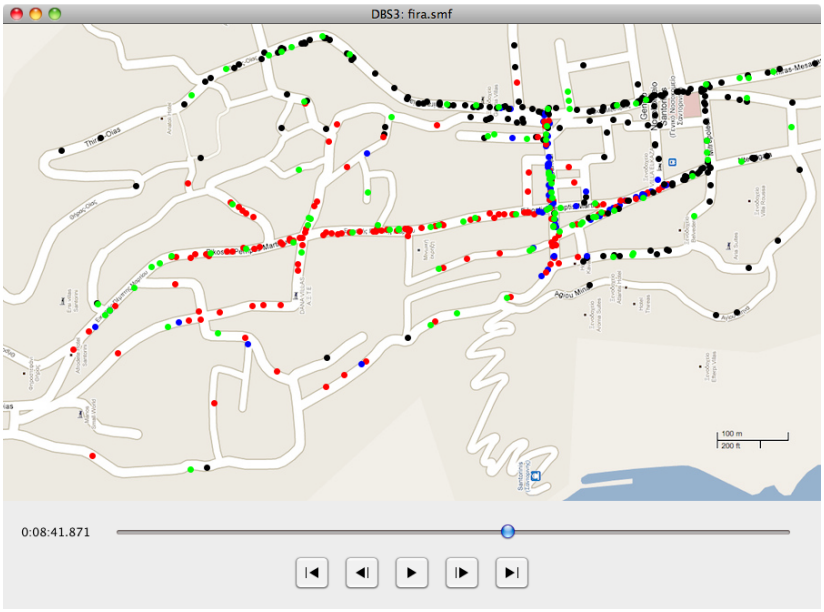

Figure D.14: A simulation of a highly virulent airborne disease shown in DBS3's GUI, which is displaying agent states computed by an external MVISP client. Agents are shown in the uninfected (black), incubating (blue), contagious (red), or vaccinated (green) states as a virus with a one-minute incubation time and a two-metre infection range spreads among the mobile agents in Fira, Greece.

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 D.14 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 D.15, which shows how client 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

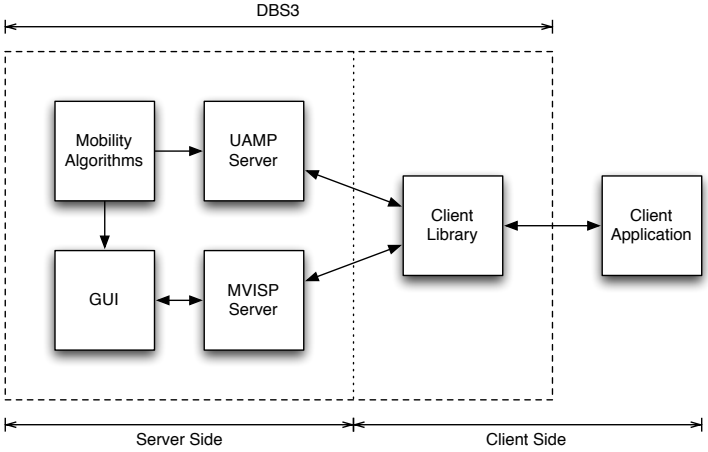

Figure D.15: The overall architecture of DBS3. The server side is responsible for computing agent mobility and sending those results to clients on-request via a UAMP server. Alternately, individual simulations can be displayed in the GUI, optionally using the MVISP server to receive state changes from an external client. Clients that utilize mobility data sent by DBS3 can be written quickly using the provided UAMP / MVISP client library.

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.