Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

WHO GOES THERE? USING AN AGENT-BASED SIMULATION FOR TRACKING POPULATION MOVEMENT

Jordan Lueck, Jason H. Rife

Department of Mechanical Engineering
Tufts University
200 College Ave
Medford, MA 02155

Samarth Swarup

Biocomplexity Institute & Initiative
University of Virginia
995 Research Park Boulevard
Charlottesville, VA 22911

Nasim Uddin

Department of Civil, Construction, and Environmental Engineering
University of Alabama, Birmingham
1075 13th Street South
Birmingham, AL 35294

ABSTRACT

We present a method to apply simulations to the tracking of a live event such as an evacuation. We assume only a limited amount of information is available as the event is ongoing, through population-counting sensors such as surveillance cameras. In this context, agent-based models provide a useful ability to simulate individual behaviors and relationships among members of a population; however, agent-based models also introduce a significant data-association challenge when used with population-counting sensors that do not specifically identify agents. The main contribution of this paper is to develop an efficient method for managing the combinatorial complexity of data association. The key to our approach is to map from the state-space to an alternative *correspondence-vector* domain, where the measurement update can be implemented efficiently. We present a simulation study involving an evacuation over a road network and show that our method allows close tracking of the population over time.

1 INTRODUCTION

Microsimulations and agent-based simulations are increasingly being used to model complex social phenomena, including disaster planning (Barrett et al. 2013), crime (Rosés et al. 2018), environmental effects on health (Yang et al. 2018), infectious disease epidemics (Halloran et al. 2008), and more (e.g., Zhang et al. 2015; Rai and Henry 2016; Rammer and Seidl 2015). These simulations integrate data from multiple sources and model large numbers of agents. The focus of this type of work tends to be on explanation, forecasting, and evaluating possible interventions. In situations where the phenomena to be modeled are dynamic and of short duration, such as disaster response, this limits the application of simulations to preparedness and planning beforehand, and explanation and analysis after the fact.

In the present work, we discuss how the use of agent-based simulations can be extended to live tracking of an ongoing event. This would enable the use of simulation-based technology in real-time situations, such as an ongoing disaster response. A crucial problem in such situations is state estimation, i.e., understanding what is happening on the ground, where people are, and how conditions are evolving. A detailed agent-based simulation in principle offers a method for modeling conditions in real-time, insofar as it allows a natural

representation of individuals, behaviors, infrastructure, etc., as opposed to more stylized models that might use systems of differential equations or other mathematical formalisms. In practice, however, the discrete nature of the agent-based representation makes it a challenge to use simple filtering approaches other than to track means and variances of numbers of agents (e.g., Ward et al. 2016).

We envision a scenario where an agent-based simulation is used for live tracking of an event by continual updates to the simulation through limited sensing of the real world. We take a particle filtering approach, where states are estimated for a number of agents roughly equal to the number of individuals in the population of interest. The main challenge here is a *data association* problem, i.e., determining which observation should be associated with which agent in the simulation. If an observation were to uniquely identify an agent, this problem would be trivial, but unique identification is not generally possible for *population counting* sensors, like video surveillance cameras, that would likely be used to support evacuation and disaster-response applications. Although data association methods have been developed for radar applications where target identity is ambiguous (Avitzour 1992), such strategies tend to rely on probabilistic data association (Schoenecker, Willett, and Bar-Shalom 2014) methods that scale poorly for large numbers of targets (hundreds, thousands, or even millions of targets), as is relevant for evacuation applications. Our solution is to introduce a novel *correspondence vector* (c-domain) representation, where we perform the measurement update in a way that correlates estimated agent states efficiently, but that otherwise allows agents states to evolve independently to ensure both robust state-space exploration and computational efficiency.

For the purpose of developing our method, we rely on a simulated "ground truth". Our eventual goal is to substitute a real population for this simulated ground truth; however, the use of simulated ground truth aids in exploring the space of possible environments, agent distributions, agent behaviors and sensor distributions, important in the initial phases of algorithm development. The particle-filter estimator is a second simulation that runs in parallel with the ground-truth simulation. The only interfaces between the ground truth and the particle filter are the modeled sensors, which provide a noisy count of population at specific locations.

The paper is organized as follows. After we briefly describe our ground truth model, we present our estimation method in detail. This is followed by experiments to show the performance of our approach and a discussion of the challenges, limitations, and future directions of this work.

2 GROUND TRUTH AND SIMULATION

We first describe the "ground truth" simulation. This represents a simplified evacuation scenario. We represent a road network as a graph, where two special "exit" nodes are marked. A population of agents moves over this graph, attempting to reach the exit nodes. Some agents are parts of groups (such as families), and first try to meet up with their group members before heading towards exit nodes. This is not meant to be a realistic evacuation simulation. It is a toy test-bed, but with enough complexity that we can demonstrate the efficacy and generalizability of our method.

The main components of the simulation are:

- A population of agents,
- A road network, and
- A behavior model.

We describe each of these next, as well as the format of the resulting outputs.

2.1 Population

The agent population is organized into groups of different sizes, from 1 to 4. Groups of size 1 are referred to as individuals, and the rest are referred to as group agents. Each agent is assigned an *age* that is relevant to behavior, as we describe further below.

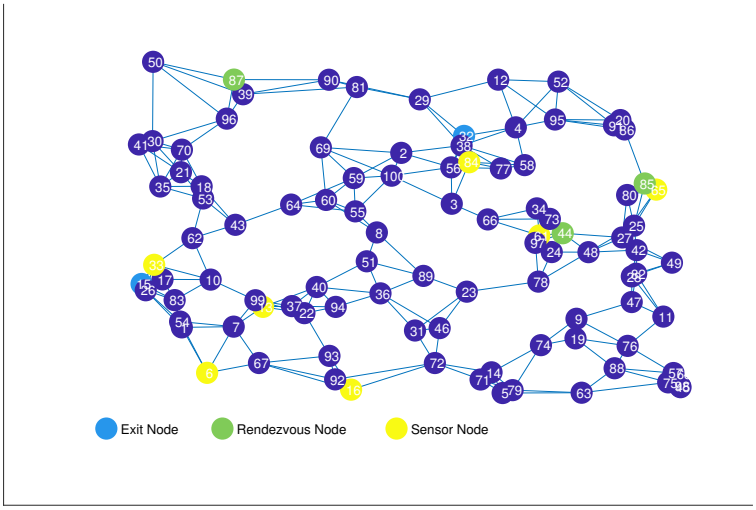


Figure 1: Map of the simulated environment, consisting of 100 nodes.

Groups correspond to families and are assigned appropriate ages and genders. In particular, children (i.e., agents with age less than 18) are always group agents.

A distribution over group sizes governs the relative numbers of individuals and group agents that are generated. In the experiments in this paper, we generated a population of 100 agents, of whom 50 were individual agents and 50 were group agents. The latter were divided into 10 groups of size 2, 6 groups of size 3, and 3 groups of size 4.

2.2 Road Network

The evacuation is assumed to be taking place over a road network. We model this as a graph embedded in two dimensions. To construct the graph, we generate a collection of random points in a square area and connect each point to its k nearest neighbors. The points correspond to the nodes in the network and, thus, each node has a corresponding (x, y) location. The road network used in the simulations presented in the following sections is shown in Figure 1. Two nodes were randomly selected as exit nodes. These are marked in light blue in Figure 1.

In our simple model, agents are assumed to be at the nodes (corresponding to intersections), and to move exactly one hop in a time step (if they choose to move at all). In reality, of course, both these assumptions are unrealistic, but they suffice for a toy model to test our method.

In the simulation, we also precompute the shortest path from each node to the closest exit node. This is helpful in implementing some of the behaviors efficiently, as described next. In the simulations used in this paper, we generated a road network with 100 nodes, where each node is connected to its four nearest neighbors.

2.3 Behavior

We implement four different behaviors:

- **Rendezvous**: In this behavior, agents belonging to a group move towards a pre-defined rendezvous location, which represents an agreed meeting point where the group will convene, even in the absence of communication. Rendezvous nodes are shown in green in Figure 1. Once agents arrive at the rendezvous, they transition to the *stay* behavior.

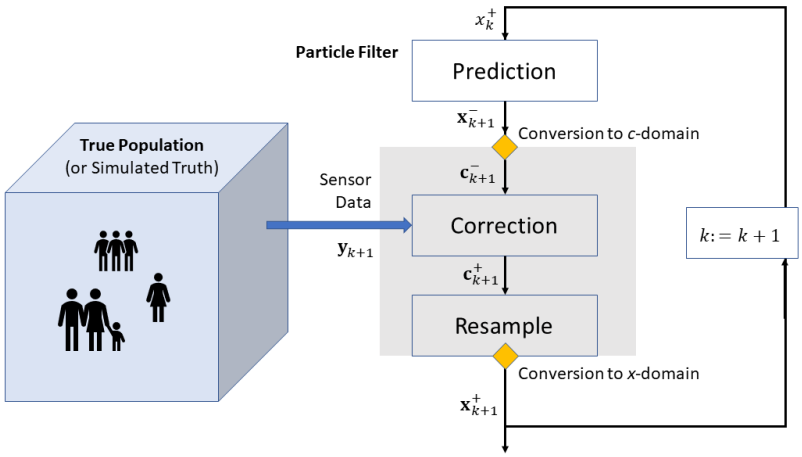


Figure 2: Block diagram showing key elements of particle filter including domain conversion.

- **Stay**: In this behavior, agents belonging to a group stay at the rendezvous node and do not move. This behavior is executed until all group members arrive at the rendezvous. Thereafter agents switch to the *evacuation* behavior.
- **Evacuation**: This is the behavior where an agent heads towards the closest exit node. The behavior is implemented efficiently by using precomputed shortest paths from all nodes to the closest exit node. Individual agents begin with this behavior.
- **Exited**: Once an agent exhibiting the *evacuation* behavior reaches an exit node, it transitions to the *exited* behavior, and remains at the exit node for the rest of the simulation.

In addition to these four behaviors, we also implement a "do-nothing" behavior, where, with a small probability, an agent in the evacuation or rendezvous behavior stays at its current node for one time step. This behavior is introduced as process noise, so that the estimator model does not perfectly predict the ground truth.

2.4 Sensing

Sensors are assumed to be deployed at a specific subset of nodes (marked in yellow in Figure 1), where they count the total population of agents at those nodes. Noisy population counts for those nodes are passed to the state estimator at each time step. The particle-filter estimator then infers agent states that cannot be directly sensed, including agent behaviors and the locations of agents not collocated with a sensor. The sensors are the only interface between the ground-truth simulation and state-estimation simulation, as shown in Figure 2. The sensor data are critical for steering the state-estimator (described in the following section), so that the state estimates do not diverge from the ground truth due to unknown initial conditions, sensor noise, and modeling errors.

3 STATE ESTIMATION AND *C*-DOMAIN CONVERSION

This section provides a mathematical model of the state estimator, including sensor modeling and data association. Data association is critical since the population-counting sensors provide only aggregate measurements and not specific information about the identity of individual agents in the true population.

3.1 State definition

In our current simulations, it is assumed the total number of agents N is known. Each agent n is described by a position state ${}^{(n)}z_k$ and a behavior state ${}^{(n)}b_k$, so that the agent's state vector is:

$$ {}^{(n)}\mathbf{x}_k = \begin{bmatrix} {}^{(n)}z_k \\ {}^{(n)}b_k \end{bmatrix} \tag{1} $$

The joint state for all agents is $X_k = \{{}^{(n)}\mathbf{x}_k | n \in [1,N]\}$. Note that both the location state and behavior state are integers, with the location state referring to the index of a node in the graph and the behavior state referring to one of the set of behaviors defined in Section 2.3.

The evolution of the joint state-space is modeled with a particle filter (Thrun, Burgard, and Fox 2005), whose structure is illustrated in Figure 2. The filter estimates the density function for the joint state distribution at each time step k as:

$$ P_{pop}(X_k) = \prod_{n \in [1,N]} P_{ag}\left({}^{(n)}\mathbf{x}_k\right) \tag{2} $$

Here the subscript pop refers to the joint state for the entire synthetic population, while the subscript ag refers to the state for each individual agent. In order to reduce the dimension of the state space and make computations efficient, correlations between agents are not modeled explicitly in the estimated distribution (2); instead, we introduce correlations through Metropolis-Hastings (M-H) resampling, as has previously been proposed for efficient multi-agent tracking, for instance by (Khan, Balch, and Dellaert 2004). In (2), the probability distribution for each agent is a mixture of distributions over a set of M particles, where each particle represents a kernel with limited local support. The mixture distribution is

$$ P_{ag}\left({}^{(n)}\mathbf{x}_k\right) = \frac{1}{M} \sum_{m \in [1,M]} P_{pa}\left({}^{(n)}\mathbf{x}_k; {}^{(m,n)}\hat{\mathbf{x}}_k\right) \tag{3} $$

Each vector ${}^{(m,n)}\hat{\mathbf{x}}_k$ is a parameter vector describing the m-th particle for the n-th agent. Each such vector consists of three parameters: ${}^{(m,n)}\hat{\mu}_k$, ${}^{(m,n)}\hat{\alpha}_k$, and ${}^{(m,n)}\hat{b}_k$. The first two parameters describe the local support for the location distribution, with μ indicating the distribution mode and α representing a distance roll-off factor. The behavior state is assigned a specific value, with no local support. Each particle's contribution to the mixture distribution is

$$ P_{pa}\left({}^{(n)}\mathbf{x}_k; {}^{(m,n)}\hat{\mathbf{x}}_k\right) = \frac{1}{{}^{(m,n)}Q_k} e^{-\frac{1}{2}\left({}^{(m,n)}\hat{\alpha}_k \, \mathrm{dist}[{}^{(m,n)}\hat{\mu}_k, {}^{(n)}z_k]\right)^2} \delta\left({}^{(m,n)}\hat{b}_k - {}^{(n)}b_k\right) \tag{4} $$

The exponential term describes a quasi-Gaussian distribution over the location graph, with the **dist** function measuring graph distance and with probability decaying as the square of graph distance. The scalar Q_k normalizes the exponential so that the location distribution integrates to unity. The δ term represents a unit indicator function, which concentrates all probability on a particular behavior state.

3.2 Prediction

Our estimator uses a conventional Bayesian prediction step, where the prediction is evaluated as an integral over the space $\mathscr{X}$ of all possible states:

$$ P(X_{k+1}|Y_k) = \int_{X_k \in \mathscr{X}} P(X_{k+1}|X_k) P(X_k|Y_k) dX_k \tag{5} $$

In our implementation, we assume that the process noise is negligible, such that the model dynamics given by $P(X_{k+1}|X_k)$ are deterministic. Though deterministic, dynamics depend on agent behavior, so there is potential for bifurcation of the state distribution in cases where behavior is uncertain. To keep

our implementation as simple as possible, we assume that the prediction step can change mode location μ and behavior b, but not the inverse-length scale α. In general terms, the previous corrected parameters (indicated by a "+" superscript) are mapped through a nonlinear function $\mathbf{g}$ to obtain the predicted parameters (indicated by a "−" superscript) at the next time step. The nonlinear mapping includes g_1, a behavior-gated model of particle motion, and g_3, a behavior-change model.

$$ {}^{(m,n)}\hat{\mathbf{x}}_{k+1}^{-} = \mathbf{g}\left({}^{(m,n)}\hat{\mathbf{x}}_{k}^{+}\right) = \begin{bmatrix} g_1\left({}^{(m,n)}\hat{\mu}_k^{+}, {}^{(m,n)}\hat{b}_k^{+}\right) \\ {}^{(m,n)}\hat{\alpha}_k^{+} \\ g_3\left({}^{(m,n)}\hat{b}_k^{+}\right) \end{bmatrix} \tag{6} $$

The location-change model g_1 advances each agent along one edge of the graph towards its target node when the behavior is *rendezvous* or *evacuation*; otherwise the agent remains at the same node. The behavior-change model g_3 is implemented as a finite state machine that cascades through a sequence of four behaviors: *rendezvous*, *stay*, *evacuation*, and *exited*. All agents that are members of a group begin in the *rendezvous* behavioral state; all singleton agents begin in the *evacuation* state. State transition occur for a given particle when the agent reaches its target and its behavior is *rendezvous* or *evacuation*. A particle transitions from *stay* to *evacuation* only after passing a transition test, in which a uniformly sampled probability is compared to a transition threshold $P_{tr}(k) = P_{tr}(k-1) * (1-P_0) + P_0$.

With the exceptions of omitting *do-nothing* and simplifying the transition to *stay*, the behavioral dynamics modeled for particles are identical to the true behavioral dynamics. In the case of the *stay* behavior, the particle model was simplified to keep the behavior update g_3 independent of other particles. Interactions between particles would otherwise be needed to implement the true transition behavior, since the true agents only transition from *stay* after their full group has reunified at a designated rendezvous node. Behavior transitions involving interactions between multiple particles and multiple agents will be left to future work. Note: the approximate transition model was tuned to provide a reasonable predictive estimate of the rate of agents exiting, a process which set $P_{tr}(1) = 0$ and $P_0 = 0.2$.

3.3 Correction

The primary novelty of our proposed Bayesian estimation scheme is in the correction (aka *measurement update*) step. We assume that sensor data are acquired from cameras that each measure the total population count at a given location. Such sensors are challenging in that they do not identify specific agents, which creates a *data association* problem similar to that found in common radar applications (Bar-Shalom, Willett, and Tian 2011) but on a larger scale. Managing data association in large scale agent-based simulations remains an unsolved problem (Long 2016).

To address this challenge, our approach is to map from the x-domain, which describes the joint state $\mathbf{x}_k$, to the c-domain, which describes the data observed by cameras using a compact correspondence vector representation $\mathbf{c}_k$. A correspondence vector represents a hypothesized association of each agent with a specific sensor, where the sensors are identified by an integer index and where the index 0 indicates that no sensor is present. Accordingly, a correspondence vector $\mathbf{c}_k \in \mathbb{Z}^N$ is constructed with an integer element (sensor ID or 0) for each agent n.

The noisy sensor measurement $\mathbf{y}_k$ is related to the correspondence vector by a sensor noise model: $P(\mathbf{y}_k|\mathbf{c}_k)$. The vector $\mathbf{y}_k \in \mathbb{Z}^L$ consists of L population-count measurements, one for each sensor $l \in [1, L]$. We assume the sensor is designed to always produce a positive measurement. A representative sensor noise distribution is the binomial distribution, which is used to model the distribution of a set of T_l random trials each with a success probability of P_d. In our implementation, we assume the performance of all sensors is identical and further assume that P_d represents the probability of detecting any one individual given that the individual is present at a location. Since we expect the detection probability to be relatively high, we set

$P_d = 0.9$. The number of trials T_l for a given sensor l is set to the sum of correspondence-vector elements equal to l.

$$T_l(\mathbf{c}_k) = \sum_n \delta\left({}^{(n)}\mathbf{c}_k - l\right) \tag{7}$$

This model captures missed-detection events, when a target is present but not recorded by the population counter. The model does not capture false-alarm events, when spurious signals artificially increment the counter. As such, the noisy measurement is biased to be always less than or equal to the actual number of agents present at a given location. (However, in future work, we will extend our models to include the effects of false alarms.) Assuming that the sensor noise distributions are independent, the probability of the set of all sensor measurements, given a particular hypothesized data-association vector, is the product of binomial distributions p_{bino} over all L sensor measurements:

$$P(\mathbf{y}_k|\mathbf{c}_k) = \prod_{l\in[1,L]} P_{bino}(\mathbf{y}_l; T_l(\mathbf{c}_k), P_b) \tag{8}$$

To leverage this sensor model for Bayesian correction, it is first necessary to convert from the x-domain to the c-domain, as illustrated in Figure 2. This conversion can be accomplished by defining the correspondence-vector distribution given prior sensing information Y_{k-1}. The set Y_{k-1} concatenates all measurements through the prior step $k-1$. In our Bayesian filter, the set Y_{k-1} is reflected in the current parameters ${}^{(n)}\hat{\mathbf{x}}_k$ obtained for each agent n via prediction:

$$P(\mathbf{c}_k|Y_{k-1}) = \prod_{n\in[1,N]} P_{ca}\left({}^{(n)}\mathbf{c}_k; {}^{(n)}\hat{\mathbf{x}}_k\right) \tag{9}$$

This product of independent distributions is derived from the joint distribution (2), and depends on the scalar distribution (P_{ca}) of the probability that the agent is at the location of the sensor identified by the correspondence vector:

$$P_{ca}\left(l; {}^{(n)}\hat{\mathbf{x}}_k\right) = \begin{cases} \tilde{P}_{ag}(s_l), & l > 0 \\ 1 - \sum_{i\in[1,L]} \tilde{P}_{ag}(s_i), & l = 0 \end{cases} \tag{10}$$

The distribution P_{ca} is either the probability that an agent is located s_l, the location of the l-th sensor, or that the agent is not co-located with any sensor, when $l = 0$. In (10), the agent-location distribution $\tilde{P}_{ag}$ is the marginal distribution obtained by integrating (3) over all values of the behavior state ${}^{(n)}\hat{b}_k$:

$$\tilde{P}_{ag}\left({}^{(n)}z_k\right) = \int_b P_{ag}\left({}^{(n)}\mathbf{x}_k\right) db_k \tag{11}$$

Once the state distribution is mapped from the x-domain to the c-domain, the Bayesian measurement update equation can be written:

$$P(\mathbf{c}_k|Y_k) = \frac{P(\mathbf{y}_k|\mathbf{c}_k)P(\mathbf{c}_k|Y_{k-1})}{P(\mathbf{y}_k)} \tag{12}$$

The right side of this equation is constructed by computing the first term in the numerator as the sensor noise model, given by (8), and the second as the mapping of the predicted states to the c-domain, given by (9). The denominator $P(\mathbf{y}_k)$ is a simple scale factor, which ensures the left-hand side integrates to one.

An important implementation detail is that the space of possible correspondence vectors $\mathbf{c}_k$, is large, so computing the full distribution (9) is combinatorially complex. With a nod to efficiency, our approach to representing (9) is to model it with a set of c-space particles ${}^{(j)}\hat{\mathbf{c}}_k$, where each c-space particle has an index j and where total number of correspondence-vector particles is M, chosen arbitrarily to be equal to the number of particles used to represent the state of each agent. The c-space particles are chosen by

building a table of the probability $\tilde{P}_{ag}$ for all N agents over all L sensor locations and augmenting with the complementary probability for the $l = 0$ case. Each c-space particle is constructed by sampling a location for each agent n from the table.

$$^{(m,n)}z_{samp} = \text{sample}\left(\tilde{P}_{ag}\left(^{(n)}z_k\right)\right) \tag{13}$$

Now define l_{samp} to be the sensor index which has position $s_{l_{samp}} = {}^{(m,n)}z_{samp}$ or zero if there is no sensor at ${}^{(m,n)}z_{samp}$. The entry of the m-th correspondence vector for the n-th agent is ${}^{(m,n)}\mathbf{c} = l_{samp}$. As a byproduct of sampling, a new set of particle parameters is also generated.

$$^{(m,n)}\mathbf{c} \implies {}^{(m,n)}\hat{\mathbf{x}}_k^+ = \begin{bmatrix} {}^{(m,n)}z_{samp} \\ {}^{(m,n)}\alpha_{samp} \\ {}^{(m,n)}\hat{b}_k^- \end{bmatrix} \tag{14}$$

In the above equation, α_{samp} is the inverse-length scale of the particle, which is unchanged if there is no sensor association $\left({}^{(m,n)}\alpha_{samp} = {}^{(m,n)}\hat{\alpha}_k^- \text{ if } {}^{(m,n)}\mathbf{c} = 0\right)$ or which becomes a spike if the particle is associated with a specific sensor $\left({}^{(m,n)}\alpha_{samp} = \alpha_{spike} \text{ if } {}^{(m,n)}\mathbf{c} \neq 0\right)$. For our implementation, we set $\alpha_{spike} = 10$.

3.4 Resampling in c-domain

Resampling guides the particle population toward the sensors and reduces the risk of particle deprivation, a phenomenon in which the estimate depends only on a small number of "good" particles. In our implementation, resampling also helps to manage the combinatorial complexity of modeling possible associations in the c-space. To this end, resampling is implemented at every time step using a Metropolis-Hastings (M-H) formulation.

Our M-H formulation operates by iteratively selecting correspondence vectors as candidates for replacement. At each M-H iteration, the candidate m^* is chosen from the set of all M correspondence vectors with uniform probability. Then a proposal vector is generated from the candidate. The proposal vector is generated by selecting one agent n^* in the candidate vector and "flipping" its correspondence, meaning that an association with a given sensor is removed ${}^{(m^*,n^*)}\mathbf{c} := 0$ or, if unassociated, an association is established ${}^{(m^*,n^*)}\mathbf{c} :\neq 0$, selecting the new sensor association with uniform probability over all sensors. The proposal $\mathbf{c}'$ is then compared to the original candidate $\mathbf{c}^*$ to determine the relative likelihood of the *posterior distribution* for either vector. This ratio is called the acceptance ratio a:

$$a = \min\left(1, \frac{P(\mathbf{y}_{k+1}|\mathbf{c}')P_{k+1}^-(\mathbf{c}')}{P(\mathbf{y}_{k+1}|\mathbf{c}^*)P_{k+1}^-(\mathbf{c}^*)}\right) \tag{15}$$

The sensor noise distribution above is evaluated using (8); the prior distribution (9). Because it is symmetric, the proposal distribution is omitted from (15). Proposal vectors are accepted if a standard uniform-distribution sample is less than the acceptance ratio a. If the proposal is accepted, then the x-domain particle (14) is updated to the appropriate μ and α; otherwise, the original candidate vector is preserved. Specifically, if a proposal is accepted that associates an agent with a specific sensor, then the inverse-distance metric ${}^{(m*,n*)}\alpha_{samp}$ is set to α_{spike}, and if a propoasl is accepted that causes an agent to move to a location not associated with a sensor, then ${}^{(m*,n*)}\alpha_{samp}$ becomes equal to the baseline value used to initialize all α.

The proposal process is repeated for a number of iterations $B+M$ where B is a burn-in period (20 iterations in our implementation). Finally, the mapping from the c-domain to the x-domain is accomplished by aggregating the pool of values (14) for all surviving c-vectors at the conclusion of M-H iterations. This

pool becomes the new set of x-domain particles input to the prediction algorithm at the next time step. Accordingly, the particle population is completely regenerated at each time step.

Importantly, M-H resampling ensures that particle behaviors are correlated since particles are chosen to conform to a given correspondence-vector. In other words, the correlation in the sensor model of (15) is the key step that inserts correlation into the joint distribution (2). Though the product of distributions in (2) suggests independence of the agent states, in fact they are correlated because of the relationships of the median-location values μ_k.

3.5 Scalability

Most steps in the proposed algorithm scale as the total number of particles MN, where N is the number of agents and M is the number of particles per agent. For instance, the prediction operation must run MN times, to update the location and behavior states of each particle.

Only one operation has higher computational complexity: the conversion from the x-domain to the c-domain. If efficiently implemented, the conversion requires MNL operations, since the probabilities of all MN particles must be evaluated at each of L sensor locations. Although domain conversion is the algorithm's most computationally expensive operation, it nonetheless provides massive computational savings over evaluating the sensor model in the x-domain, which would require managing combinatorial complexity involving $\binom{N}{\sum_l y_l}$ possible agent combinations for each possible sensor noise vector.

It is reasonable to conclude that the algorithm would scale well to larger simulations, involving more agents N and more sensors L, since the computational costs are linear in both variables. Moreover, algorithm refinements might even further reduce computational costs. For instance, sublinear scaling in N might be possible using multi-resolution particles, for instance by clustering a set of agents together and representing the entire cluster with only M particles (instead of representing each individual agent with M particles).

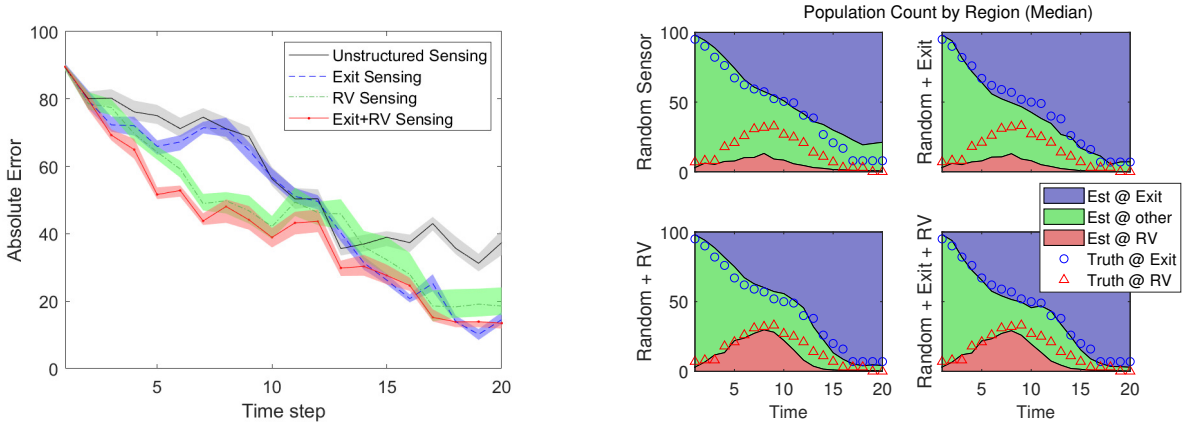
We also note that scalability of particle filtering is an active area of research, through the development of efficient resampling algorithms and parallelization (Murray et al. 2016).

4 EXPERIMENTS AND RESULTS

We applied Monte Carlo simulations to validate our methods. In the validation study, we considered a single set of truth data, which was generated for a case with 100 agents moving in an environment consisting of 100 nodes. Of the 100 nodes, two nodes were randomly selected as exits (nodes 15 and 32) and three nodes as rendezvous points (nodes 44, 85, and 87), as illustrated in Figure 1. Half the agents were clustered into groups, including 10 groups of two agents, 6 groups of three agents, and 3 groups of four agents. Each group was assigned a rendezvous point. Group members initially proceeded to the rendezvous and then waited there until all group members were present before proceeding toward the nearest exit node. The truth simulation ran for a total of 20 time steps, with agent positions updating at each time step.

Synthetic sensor measurements were generated from the truth data assuming population-counters were placed at eight distinct nodes. For each time step, a set of eight noisy sensor values was obtained by sampling (8). Because algorithm performance was sensitive to sensor placement, we considered four sensor configurations which featured: unstructured sensing (eight randomly placed sensors), exit sensing (six randomly placed sensors plus two at the exit nodes), rendezvous sensing (five randomly placed sensors plus three at the rendezvous nodes), and exit + rendezvous sensing (three randomly placed sensors plus two at the exits and three at the rendezvous nodes). To account for random sensor placement, random sensor measurement noise, and randomized initial particle placement, 101 trials were generated for each of the four configurations.

Results from the simulation indicated that tracking error steadily decreased over time, as shown in Figure 3a. The figure shows the four sensor configurations, with unstructured sensing shown in gray, exit sensing in blue, rendezvous sensing in green, and exit + rendezvous sensing in red. Absolute error is calculated at each time step as the absolute difference between the particle count (normalized by total



(a) (b)

Figure 3: Monte Carlo simulation results for four sensor configurations: unstructured, exit, rendezvous, and exit+rendezvous sensing. Left: Absolute error integrated over all locations vs. time. Right: Estimated agent populations at rendezvous nodes (red field), exit nodes (blue field), and other nodes (green field) vs. time, as compared to true counts (triangle and circle markers, for rendezvous and exit nodes respectively).

particles, $M = 50$) and the true count, summed over all nodes. For each of the four sensor configurations, a trace of median error is shown as a function of time step, with shading indicating a 50% confidence interval (one quartile on either side of the median).

As expected, the error is highest when sensor placement is fully unstructured. Likewise, the error is lowest when the sensor placement is most structured, namely, in the case when a subset of sensors is consistently placed at both the exit and rendezvous nodes. Because the sensor placement in the unstructured configuration is sparse, the sensors provide little if any useful data; in other words, the unstructured configuration is essentially an open-loop simulation using the dynamic model. At the end of the simulation, Figure 3a shows that the absolute error for the unstructured configuration is nearly 40, implying 20 misplaced agents out of 100 (with the the absolute error double counting misplaced agents, with one error for each surplus agent at one node and one error for each deficit agent at another). By comparison, the structured sensor placement results in an absolute error of about 8, implying only 4 misplaced agents out of 100.

Interestingly, the information content is richest for the rendezvous-node sensors. In other words, for this case, the simulation does best at inferring where agents are located, even when they are not collocated with a sensor. As indicated by Figure 3a, the error curve for rendezvous sensing (green) is nearly as low as that for combined rendezvous+exit sensing (red). By contrast, the error of the exit sensing configuration (blue) tracks the error of the unstructured sensing configuration (black) until the very end of the simulation, when most of the agent population has arrived at the exit nodes. Also, the rendezvous node population appears to be very sensitive to sensor configuration, as shown in Figure 3b. The figure tracks the estimated number of agents at all exit nodes (blue field), rendezvous nodes (red field), and other nodes (green field). The true counts are shown with markers: blue circles for the exit-node population and red triangles for the rendezvous-node population. The figure shows that the exit-node population is well modeled in all cases, except in the case of unstructured sensing, where the exit-node population is mis-modeled at the very end of the simulation. The figure shows the rendezvous-node population is poorly modeled in cases with unstructured and exit-only sensing. However, when rendezvous sensing is added, the estimate of the rendezvous-node population is significantly better, which implies the rendezvous sensors are important for capturing the subtle internal dynamics of the system.

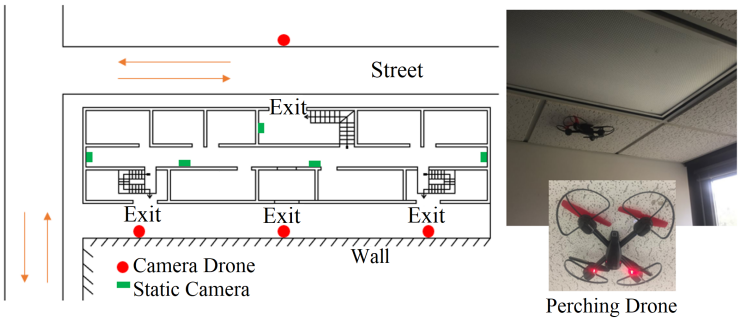


Figure 4: In future work, we propose to validate our algorithms with lab-based experiments.

5 DISCUSSION

Recent work on building complex, data-driven, high-fidelity simulations has brought to the fore several new challenges including data assimilation (Ward et al. 2016; Lloyd et al. 2016), agent-based model calibration (Lamperti et al. 2018; Singh et al. 2018), simulation analytics (Swarup et al. 2019), etc. The main impetus of this line of research is to increase the range of applicability of simulation-based methods.

The advent of the Internet of Things and the broad availability of streaming data from various kinds of sensors has made it possible to conceive of simulations that are continually updated and are able to track ongoing phenomena in real-time (Wang and Hu 2015, e.g.). This conception raises new challenges as well. There is a trade-off between fidelity and realism of the simulation on the one hand, and the data rate and computational scalability of tracking on the other hand.

In the work presented here, we are taking the first step towards addressing the application of agent-based simulations to real-time scenarios, specifically from the perspective of using a population and behavior model as a basis of tracking. Other recent approaches only model population statistics, such as the number of people in an area (Ward et al. 2016). Using our novel correspondence-vector domain representation, however, we are able to do the tracking at the individual agent level.

5.1 Extensions

A number of extensions to the current work are possible. In order to develop this approach further in the direction of real-world application, we intend to work on both theoretical and practical advancements.

On the theoretical front, we need to extend our models to include agent demographics, richer behaviors, and different sensors (e.g., GPS in cell phones). We also will consider fusing multiple types of sensors, making the approach robust to errors in the agent-based model, and extending the approach to active sensing case (where we can actively relocate sensors, e.g., using drones). Sensitivity studies are also required, to evaluate how results depend, for instance, on the number and size of groups, the total number of agents in the simulation, and the connectivity of the location graph.

On the practical front, we intend to demonstrate the method in increasingly complex situations. Before attempting tests in real-world settings, we will first consider complex simulations, such as (Barrett et al. 2013), and also experiments in a controlled, lab-based environment. For the lab environment setting, which is already constructed, a number of static cameras will be positioned to obtain population counts. Drone cameras will affix themselves to indoor or outdoor walls using a perching mechanism. This will allow population counting in hallways or at exits of a building, as illustrated in Figure 4.

ACKNOWLEDGMENTS

S.S. was supported in part by DTRA CNIMS Contract HDTRA1-17-0118. J.R. was supported in part by NSF CMMI-1903972.

REFERENCES

Avitzour, D. 1992. "A Maximum Likelihood Approach to Data Association". *IEEE Transactions on Aerospace and Electronic Systems* 28(2):560–566.

Bar-Shalom, Y., P. K. Willett, and X. Tian. 2011. *Tracking and Data Fusion*. YBS publishing Storrs, CT, USA:.

Barrett, C., K. Bisset, S. Chandan, J. Chen, Y. Chungbaek, S. Eubank, Y. Evrenosoğlu, B. Lewis, K. Lum, A. Marathe, M. Marathe, H. Mortveit, N. Parikh, A. Phadke, J. Reed, C. Rivers, S. Saha, P. Stretz, S. Swarup, J. Thorp, A. Vullikanti, and D. Xie. 2013. "Planning and Response in the Aftermath of a Large Crisis: An Agent-based Informatics Framework". In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, 1515–1526. Piscataway, NJ, USA: IEEE Press.

Halloran, M. E., N. M. Ferguson, S. Eubank, I. M. Longini, Jr., D. A. T. Cummings, B. Lewis, S. Xu, C. Fraser, A. Vullikanti, T. C. Germann, D. Wagener, R. Beckman, K. Kadau, C. Barrett, C. A. Macken, D. S. Burke, and P. Cooley. 2008. "Modeling Targeted Layered Containment of an Influenza Pandemic in the United States". *PNAS* 105(12):4639–4644.

Khan, Z., T. Balch, and F. Dellaert. 2004. "An MCMC-Based Particle Filter for Tracking Multiple Interacting Targets". In *Computer Vision - ECCV 2004*, edited by T. Pajdla and J. Matas, 279–290: Springer Berlin Heidelberg.

Lamperti, F., A. Roventini, and A. Sani. 2018. "Agent-based Model Calibration Using Machine Learning Surrogates". *Journal of Economic Dynamic & Control* 90:366–389.

Lloyd, D. J. B., N. Santitissadeekorn, and M. B. Short. 2016. "Exploring Data Assimilation and Forecasting Issues for an Urban Crime Model". *European Journal of Applied Mathematics* 27(3):451–478.

Long, Y. 2016. *Data Assimilation for Spatial Temporal Simulations Using Localized Particle Filtering*. Ph. D. thesis, Department of Computer Science, Georgia State University.

Murray, L. M., A. Lee, and P. E. Jacob. 2016. "Parallel Resampling in the Particle Filter". *Journal of Computational and Graphical Statistics* 25(3):789–805.

Rai, V., and A. D. Henry. 2016. "Agent-based Modelling of Consumer Energy Choices". *Nature Climate Change* 6:556–562.

Rammer, W., and R. Seidl. 2015. "Coupling Human and Natural Systems: Simulating Adaptive Management Agents in Dynamically Changing Forest Landscapes". *Global Environment Change* 35:475–485.

Rosés, R., C. Kadar, C. Gerritsen, and C. Rouly. 2018, July 10-15. "Agent-based Simulation of Offender Mobility: Integrating Activity Nodes from Location-based Social Networks". In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 804–812. Stockholm, Sweden.

Schoenecker, S., P. Willett, and Y. Bar-Shalom. 2014. "ML–PDA and ML–PMHT: Comparing Multistatic Sonar Trackers for VLO Targets Using a New Multitarget Implementation". *IEEE Journal of Oceanic Engineering* 39(2):303–317.

Singh, M., A. Marathe, M. V. Marathe, and S. Swarup. 2018, July. "Behavior Model Calibration for Epidemic Simulations". In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1640–1648.

Swarup, S., A. Marathe, M. V. Marathe, and C. L. Barrett. 2019. "Simulation Analytics for Social and Behavioral Modeling". In *Social-Behavioral Modeling for Complex Systems*, edited by P. K. Davis and A. O'Mahony. Wiley.

Thrun, S., W. Burgard, and D. Fox. 2005. *Probabilistic Robotics*. MIT press.

Wang, M., and X. Hu. 2015. "Data Assimilation in Agent-Based Simulation of Smart Environments Using Particle Filters". *Simulation Modelling Practice and Theory* 56:36–54.

Ward, J. A., A. J. Evans, and N. S. Malleson. 2016. "Dynamic Calibration of Agent-Based Models Using Data Assimilation". *Royal Society Open Science* 3(4):150703.

Yang, L. E., P. Hoffmann, J. Scheffran, S. Rühe, J. Fischereit, and I. Gasser. 2018. "An Agent-Based Modeling Framework for Simulating Human Exposure to Environmental Stresses in Urban Areas". *Urban Science* 2(2):Article 36:1–21.

Zhang, W., S. Guhathakurta, J. Fang, and G. Zhang. 2015. "Exploring the Impact of Shared Autonomous Vehicles on Urban Parking Demand: An Agent-Based Simulation Approach". *Sustainable Cities and Society* 19:34–45.

AUTHOR BIOGRAPHIES

JORDAN LUECK is a B.S. student in Mechanical Engineering at Tufts University. His email is jordan.lueck@tufts.edu.

JASON RIFE is an Associate Professor in the Mechanical Engineering department at Tufts University. He received his Ph.D. from Stanford University in 2004. His email is jason.rife@tufts.edu.

SAMARTH SWARUP is a Research Associate Professor at the Biocomplexity Institute & Initiative at the University of Virginia. He received his Ph.D. from the University of Illinois, Urbana-Champaign in 2007. His email is swarup@virginia.edu.

NASIM UDDIN is a Professor in the Department of Civil, Construction, and Environmental Engineering at the University of Alabama, Birmingham. He received his Ph.D. from the State University of New York, Buffalo in 1992. His email is nuddin@uab.edu.