Stochastic Activity Authoring with Direct User Control

Aline Normoyle*
University of Pennsylvania

Maxim Likhachev †
Carnegie Mellon University

Alla Safonova ‡
University of Pennsylvania

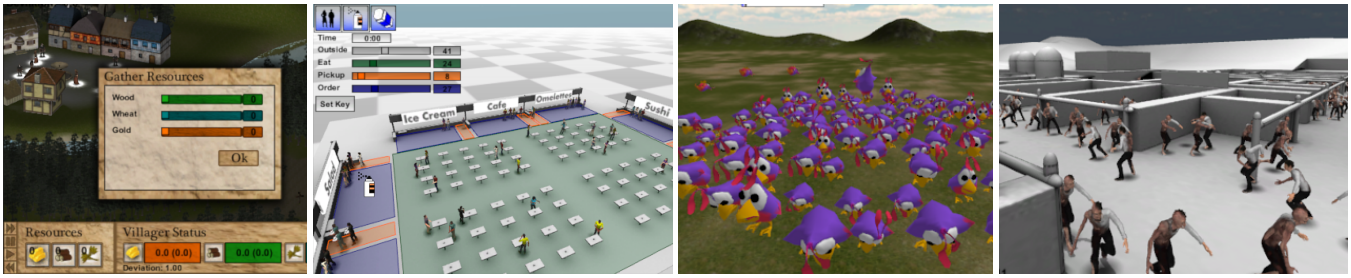


Figure 1: *Population control examples. From left to right: (1) A player interactively controls a population for gathering resources. The workers gather resources at random, while maintaining the player's instructions. (2) A game level designer can directly set up the activities of a food court so that peak times appear busiest, without manually tweaking random rates of entry and exit. (3) Bird behaviors are authored so that the proportion pecking for worms remains constant even though birds randomly switch between other activities. (4) In a large mall environment, wandering zombies explore closer areas more often than further ones while remaining evenly distributed across the environment.*

Abstract

Crowd activities are often randomized to create the appearance of heterogeneity. However, the parameters that control randomization are frequently hard to tune because it is unclear how changes at the character level affect the high-level appearance of the crowd. We propose a method for computing randomization parameters that supports direct animator control. Given details about the environment, available activities, timing information and the desired high-level appearance of the crowd, we model the problem as a graph, formulate a convex optimization problem, and solve for a set of stochastic transition rates which satisfy the constraints. Unlike the use of heuristics for adding randomness to crowd activities, our approach provides guarantees on convergence to the desired result, allows for decentralized simulation, and supports a variety of constraints. In addition, because the rates can be pre-computed, no additional runtime processing is needed during simulation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—[Animation];

Keywords: stochastic modeling, crowd authoring, optimization

*e-mail: alinen@seas.upenn.edu
†e-mail: maxim@cs.cmu.edu
‡e-mail: alla@seas.upenn.edu

http://dx.doi.org/10.1145/2556700.2556714

1 Introduction

Crowd activities are often randomized to create the appearance of heterogeneity. However, the parameters that control this randomization are frequently hard to tune because it is unclear how changes at the character level affect the high-level appearance of the crowd. For example, suppose an AI designer is using a finite state machine (FSM) to control characters in a food court. Suppose also that the animator wants to randomize the state transitions so that the final crowd appears to be made of unique individuals. The high-level appearance of the crowd will depend on numerous factors, such as the duration of each state, the distances agents must walk between various locations, and the order in which states are executed (e.g., agents must order food before they can pick it up). Tweaking the script parameters to produce a specific effect (for example, to create times of day when the food court appears busy) is tedious and time-consuming: the animator must repeatedly tweak the FSM parameters and re-run the simulation to preview the result. Furthermore, these parameters need to be re-configured whenever the environment, activity properties, or scenario changes. The crowd designer must understand the control scripts and understand which properties influence the crowd's high-level behavior. Lastly, the need for extensive parameter tweaking makes it impossible to respond in real-time to dynamic changes, for example the changes made by a game player at runtime.

In this work we investigate a method for computing random parameters that supports direct control of the crowd simulation. Given the layout of the environment and constraints on activities (such as duration, ordering, and preferences), we model agent decisions with a stochastic graph and formulate a series of convex optimization problems which compute transition rates guaranteed to satisfy the input constraints. Once computed, the new rates can be plugged into the agent simulation at no additional runtime cost.

This approach has a number of advantages.

- The distribution and the behavior of the crowd are guaranteed to converge to the specified distribution and specified constraints, regardless of the initial state of the crowd. For example, it is trivial for some agents to switch to a high-detail behavioral model in response to a player, and then resume its default behavior afterwards.

- Constraints are satisfied simultaneously even with non-trivial dependencies between activities.
- The complexity of solving the optimization problem depends on the number of activities/locations rather than the number of agents. For many scenarios, the number of modeled activities is small enough that we can pre-compute probabilities on the fly to account for changes dynamically.
- The simulation of agents according to transition rates can be easily decentralized (e.g. each agent chooses activities on its own without having to consult a centralized controller).
- Because the rates can be pre-computed, no additional runtime processing is required.
- Because our problem formulation is convex, we are guaranteed to compute transition rates as long as the input constraints are feasible (the author has immediate feedback on constraint feasibility without needing to run the simulation).

To summarize, the main contribution of this paper is a new way to compute randomization parameters for crowd simulation which supports direct control of the crowd by a user. The techniques in this paper help with computing agent-level parameters, can be used to author stochastic crowd behaviors, and enable new interfaces for controlling groups of characters, as in real-time strategy games. Our approach takes into account the spatial and timing properties of agent activities and provides guarantees on the high-level behavior of the crowd. We discuss the properties, advantages and limitations of our approach, and demonstrate its ease of use, scalability and robustness for controlling groups of agents having randomized behaviors.

2 Related Work

Randomization is a frequently used technique in games. [Mark 2011] gives a good overview, including a description of the use of distributions for selecting agent reactions. [Brockington 2002] describes the use of random walks for creatures in Never Winter Nights. In Grand Theft Auto 4, pedestrians use weighted random walks for visible characters. Additionally, stochastic spawn rates are often used to populate regions near the payer, for example, in [Admiza 2001], cars turned randomly and were replaced as necessary to keep cars close to the player.

In crowd systems, random variables are often used to add heterogeneity to behaviors according to a distribution, for example, the aleatoric actions in CAROSA [Allbeck 2010; Stocker et al. 2010], the action distribution features in [Musse and Thalmann 2001; Shoulson and Badler 2011], the distribution tools used by Industrial Light and Magic and Dreamworks [Thalmann et al. 2004], and the stochastic state machine transitions used in [Curtis et al. 2011]. Frequently, the crowd designer specifies the transition rates directly which are then sampled at the agent level. Alternatively, one may use a centralized controller to maintain a given distribution macroscopically by explicitly moving agents from over-populated areas to under-populated areas [Shoulson and Badler 2011]. Our technique provides a unified framework for directly computing transition rates which simultaneously handles multiple activity constraints. Our framework automatically handles cases which would normally require iterative, manual tuning. Lastly, because rates can be precomputed, there is no additional runtime cost.

Our method for computing rates is based on a stochastic model of agent decision-making. Similar stochastic models have been proposed for crowd simulation. For example, [Sewall et al. 2011] describes a hybrid traffic model based on Poisson processes which couples an agent-based simulation with a macroscopic traffic flow simulation. [Sunshine-Hill and Badler 2010] shows how a stochastic crowd simulation can be used as the basis for efficiently computing AI level-of-detail on demand. [Liu et al. 2012] introduces a technique for computing population distributions across activities based on a negative/positive feedback loop. In urban planning and occupational modeling, [Lovas 1994] models pedestrians using a stochastic network and [Wang et al. 2011] uses a two state Markov chain model to model entry and exit behavior from a building. In [Stylianou et al. 2004] stochastic processes are used to create large populations in urban environments. In [Sung et al. 2004] agent behaviors are influenced by action choice distributions embedded in objects in the environment. Many of these works demonstrate the massive scalability potential of stochastic modeling for crowds, but our focus is primarily authoring: how can we directly compute randomization parameters which when sampled at the agent-level, result in a desired high-level distribution?

This work is inspired by [Berman et al. 2009], which studies how to distribute a group of autonomous robots across different tasks, and [Boyd et al. 2003], which showed how to compute the fastest mixing Markov chain as a convex optimization problem. Our activity formulation is based on Markov chains (for a good introduction, please see [Ross 1996]) which have well-known steady state properties. We show how to use this work in the context of crowd simulation and extend this work to support a variety of constraints such as ordering of activities, duration of activities and preferences.

3 Methodology

In this section, we describe our approach. First, agent activity constraints are modeled as a graph. Given this graph, we solve for transition probabilities which satisfy a variety of constraints.

3.1 Activity modeling

Suppose we have a group of background agents who wish to perform some set of activities A. Activities might be buying food and eating at a food court, gathering resources for a real-time strategy game, controlling the proportions of idling behaviors in a group, or wandering in a large mall at night.

Suppose also that there are restrictions on when different activities can occur. For example, characters may need to order food before picking it up. To model these dependencies, let the activities $a_i \in A$ be modeled as nodes in a directed graph such that an edge e_{ij} exists if and only if a_i can be performed after a_j. Each edge will have a transition rate k_{ij} associated with it, which represents the probability with which an agent performing activity a_j transitions to a_i (Figure 2). For example, deterministic transitions will have $k_{ij} = 1$. Suppose we have M activities in the set A. We define the $M \times M$ matrix $\mathbf{K}$ as our stochastic activity transition matrix, where the elements k_{ij} correspond to rates between activities. Columns of $\mathbf{K}$ represent the expected outgoing flow from node a_i per unit time. Rows of $\mathbf{K}$ represent the expected incoming flow into node a_i per unit time. An element k_{ii} is non-zero only if a character may repeat activity a_i before switching to a new activity.

The aggregate state of our system can be represented as the proportion of characters performing each activity. Specifically, if our scenario contains N characters, let $n_i(t)$ represent the number of agents performing activity i at time t and let $x_i(t) = n_i(t)/N$ represent the corresponding fraction of agents performing activity i at time t. The state of the system is then given by $\mathbf{x}(t) = [x_1(t) \dots x_M(t)]^T$. Each iteration, the system is expected to change according to $\mathbf{K}$, e.g. $E[\mathbf{x}(t+1)] = E[\mathbf{K}\mathbf{x}(t)]$. Define the initial state of the system as x_0.

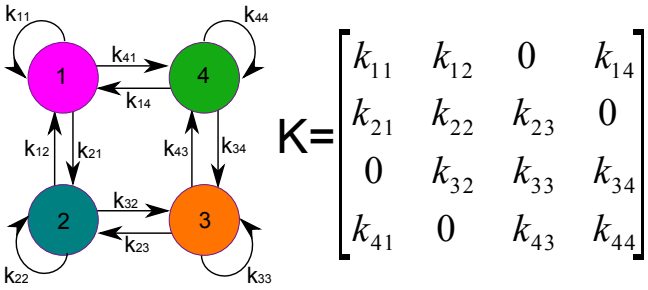


Figure 2: *Activity Modeling. Activity dependencies are modeled with a directed graph. In this example, it is not possible to perform activity 1 and 3 consecutively, nor activity 2 and 4 consecutively. Rows of the transition matrix correspond to incoming edges. Columns correspond to outgoing edges. The state of the system $x(t)$ is the proportion of agents performing each activity. For example, if every agent is performing activity 1, the state of the system is $x(t) = [1, 0, 0, 0]^T$. The desired steady state of this system, $\mathbf{x}^d$, is the user-desired proportion of agents in each activity. Given the graph and $\mathbf{x}^d$, we wish to solve for the corresponding transition rates k_{ij}.*

Regardless of the initial state, the system state $x(t)$ is guaranteed to converge to a unique steady state, $\mathbf{x}^d = \mathbf{K}\mathbf{x}^d$, provided the activity graph satisfies two properties [Ross 1996]. First, the activity graph must be strongly connected (e.g. a path exists between any two nodes). We enforce this property by adding a special activity, called "outside-system". For example, in systems where agents enter and exit, we introduce an "outside-system" node with a connection to the start and a connection from the end. Agents who are "outside-system" are not visible to the viewer. In general, the edges of "outside-system" provide paths between each pair of nodes. Second, at least one activity must not repeat on a fixed period. To understand this restriction, imagine a graph with two nodes, a_0 and a_1, each connected only to the other (e.g. no self-loops). Each frame, all agents performing a_0 will switch to a_1 and all agents at a_1 will switch to a_0. The distribution of agents can only oscillate in this case. Lastly, note that agents will generally continue to switch activities even after the steady state is reached. This behavior is desirable for producing heterogeneous-looking populations.

The system state $\mathbf{x}(t)$ succinctly describes the high-level behavior of the system. Thus, if a designer gives us a desired high-level state, $\mathbf{x}^d = [x_1^d \dots x_M^d]^T$, our goal is to compute the corresponding matrix $\mathbf{K}$ which converges to $\mathbf{x}^d$. Following the method described by [Berman et al. 2009], we define $\dot{x}_i(t)$ as the rate of change at activity a_i at time t, given by

$$\dot{x}_i(t) = \sum_{\forall j|(i,j)\in E} k_{ij}x_j(t) - \sum_{\forall j|(j,i)\in E} k_{ji}x_i(t) \qquad (1)$$

The first term represents the percentage of agents starting activity a_i and the second term represents the percentage of agents finishing activity a_i and starting to do something else. The rates at which agents change activities correspond to the $M \times M$ matrix $\dot{\mathbf{K}}$ such that $\dot{\mathbf{x}}_t = \dot{\mathbf{K}}\mathbf{x}_t$ where each entry of $\dot{\mathbf{K}}$ is given by,

$$\dot{k}_{ij} = \begin{cases} -\sum_{\forall l|(l,j)\in E} k_{lj} & \text{if } i = j; \\ 0 & \text{if } e_{ij} \notin E; \\ k_{ij} & \text{if } i \neq j \text{ and } e_{ij} \in E. \end{cases} \qquad (2)$$

By construction, each column of the matrix $\dot{\mathbf{K}}$ sums to zero. It can be shown that this system, described in terms of rates, also converges to a unique, stable equilibrium $\mathbf{x}^d$, which corresponds to the system steady state when the rate of change is zero [Berman et al. 2009], e.g., $\mathbf{0} = \dot{\mathbf{K}}\mathbf{x}^d$.

Given this formulation, we can find some matrix $\mathbf{K}$ which converges to $\mathbf{x}^d$ using the following constraints

$$\begin{aligned} &\dot{\mathbf{K}}\mathbf{x}^d = \mathbf{0} \\ &\dot{\mathbf{K}}^T\mathbf{1} = \mathbf{0} \\ &\mathbf{K}^T\mathbf{1} = \mathbf{1} \\ &k_{ij} \geq 0 \\ &k_{ij} = 0 \text{ if } e_{ij} \notin E \end{aligned} \qquad (3)$$

Because the above formulation is convex, we are guaranteed to find a solution so long as the graph structure and constraints are feasible. Typical infeasible cases occur when activities cannot repeat because such activities require agents to switch to a new activity after completion. Thus, distribution constraints which require some proportion to stay in that activity state become impossible. To solve for $\mathbf{K}$, we use the optimization package CVX [Grant and Boyd 2013] with SDPT3 [Toh et al. 1999] in Matlab.

Although each matrix $\mathbf{K}$ is guaranteed to converge to a unique, steady state $\mathbf{x}^d$, several choices of $\mathbf{K}$ can converge to the same $\mathbf{x}^d$. Thus, we can further tailor our choice of $\mathbf{K}$ to incorporate additional design considerations.

3.2 Activity Durations

Activity durations can greatly affect the high-level distribution of a crowd when they vary greatly between activities. If they are not modeled, the computed rates can fail to converge to the desired distribution.

We consider several models of activity duration (Figure 3). In the first, the amount of time each agent spends performing an activity a_i follows an exponential distribution with mean $1/(1 - k_{ii})$ (Figure 3(a)). Constraints on the mean of this duration can be achieved by either adding a soft constraint to the objective function,

$$\min \sum_{t_i \in \text{TimingConstraints}} | \, k_{ii} - \frac{(t_i - 1)}{t_i} \, | \qquad (4)$$

, or adding hard constraints of the form $k_{ii} = (t_i - 1)/t_i$.

We also model activities having a fixed duration (Figure 3(b) and Figure 3(c)). These cases arrive frequently in animation, where the duration corresponds to the length of a character motion clip. Here, we model the duration as a chain of nodes, where each node represents a unit of time δt. Thus, if the activity takes t units of time, we create a chain consisting of $t/\delta t$ subnodes and divide the desired distribution evenly among them. $\mathbf{K}$ is then solved as before. If the activity can repeat, we add an edge from the last subnode in the chain to the first.

3.3 Modeling travel time between activities

In cases where some activities are far apart, the time needed for a character to travel to their destination can significantly affect the high-level appearance of the simulation (See figure 5). In this case, our approach is to maintain the relative proportions of agents in each activity, but model the proportion of agents in transition. Define the proportion performing activities as $\alpha \in (0, 1)$ and the proportion in transition as $1 - \alpha$. The desired state of the system becomes $[\alpha\mathbf{x}^d \quad (1 - \alpha)\mathbf{y}^d]^T$, where $\alpha\mathbf{x}^d$ is the desired proportion of characters performing each activity and $(1 - \alpha)\mathbf{y}^d$ is the desired proportion transitioning between activities.

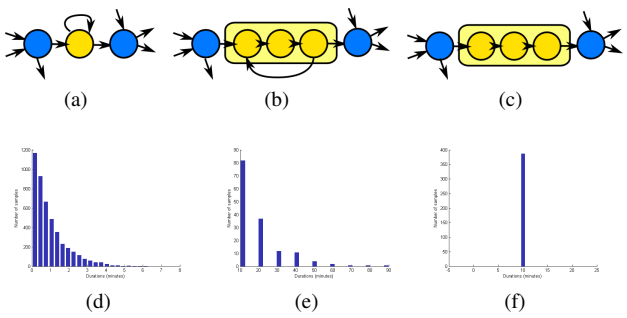


Figure 3: *Activity Durations. The graph structure can model different activity durations. Histograms below each graph shows the result of simulating agents whose activity durations correspond to the graph. The x-axis corresponds to duration of activity (minutes). The y-axis corresponds to counts of activities having each duration. 3(a) shows how to model an activity duration which follows an exponential distribution. The mean duration is a function of the transition rate for repeating the same activity. 3(b) shows how to model an activity duration which is constant, but can repeat. In this case, we model the node duration as a chain of subnodes, where each transition corresponds to a unit amount of time δt. 3(c) shows how to model a constant activity duration. In this case, we do not have a transition that allows us to repeat the activity.*

To model travel time, we replace each edge e_{ij} with a node y_i which has a single incoming edge from x_j and a single outgoing edge into x_i. To keep the graph size smaller, edges which correspond to short travel distances do not need to be expanded, and pairs of activities which share start and end locations may share the same y_i. The rate of change at each activity $\dot{x}_i(t)$ is now,

$$\dot{x}_i(t) = \alpha \left(\sum_{\forall j \mid (i,j) \in E} k_{ij} x_j(t) - \sum_{\forall j \mid (j,i) \in E} k_{ji} x_i(t) \right) - (1-\alpha) \left(\sum_{\forall j \mid (i,j) \in E} y_j(t) - \sum_{\forall j \mid (j,i) \in E} y_i(t) \right) \quad (5)$$

Similarly to activity durations, travel time is then modeled by expanding each node (section 3.1.1). In the above formulation, both the rates k_{ij} and the agents in transition y are unknown. Both may be solved for simultaneously by substituting the above rate equation for the one given in section 3.1.

3.4 Preferences between activities

A designer may want characters to prefer some activities over others while still maintaining a desired high-level appearance. For example, agents might prefer closer locations to further ones, or prefer to gather a resource from a safe location over an unsafe one. Such constraints can be achieved by weighting the probability for preferred activities to be higher than others, specifically, by adding hard constraints of the form $k_{ik} \leq k_{ij}$, or soft constraints to our objective function of the form $\mid k_{ij} - k_{ik} - c \mid, c \geq 0$.

3.5 Mapping activities to locations

So far, we have considered the set of activities (e.g. gold mining, farming, etc.) without considering the actual locations, or sites, where they are performed. Although it is possible to use the approach above to explicitly model both activity and location (each graph node could represent an activity and location tuple), such details can be easily abstracted from the designer by computing how activities should map to sites. Because the number of locations will often be much larger than the number of activities, this can greatly simplify authoring for the designer. Additionally, this approach allows the designer to apply the same activity model to new environments as well as dynamically support changes to same environment, for example, if a location is destroyed.

Let $\mathbf{x}^d$ now represent the desired spatial distribution of activities over specific locations. Each element of $\mathbf{x}^d$ will correspond to a tuple (a_i, ℓ_k), where activity a_i can be performed at location ℓ_k. We will refer to these tuples as sites. Examples of sites might be shopping at a grocery store or sleeping in a park. Let the number of sites in our scenario be S.

Let $\mathbf{a}^d$ be the corresponding desired distribution in terms of activities across all sites. First, assume $\mathbf{a}^d$ is fully specified (if not, we can distribute evenly among the remaining activities). We can initialize each element x_j^d of $\mathbf{x}^d$ as

$$x_j^d = \frac{a_k}{\mid A_k \mid} \quad (6)$$

where a_k is the activity that can be performed at this site and A_k is the set of sites that support activity a_k. For example, a_k might correspond to the activity "shopping" and the set A_k might consist of 10 stores where agents can shop. The initial density of shoppers in each store is $a_k/10$. If the user simply wants to distribute the desired proportion of agents across all sites evenly, we are done. However, in many cases, we would like to automatically weight the distribution across sites based on properties on the environment.

Thus, we define a SxS weighting matrix $\mathbf{W}$ to encode how activities should be distributed across sites, e.g. the elements of $\mathbf{W}$ encode which sites should have the greater proportion of agents. Elements in $\mathbf{W}$ that correspond to nonexistent edges in our graph are set to zero. We compute $\mathbf{x}^d$ iteratively by re-weighting the proportions at each site according to $\mathbf{W}$ and then projecting back onto the desired distribution across activities, e.g.

$$\mathbf{x}^d = \frac{\mathbf{W}\mathbf{x}^d}{\mathbf{c}^T \mathbf{x}^d} \quad (7)$$

where the elements c_i in $\mathbf{c}$ are the sum of each column i of $\mathbf{W}$. Dividing the vector $\mathbf{W}\mathbf{x}^d$ by the scalar $\mathbf{c}^T\mathbf{x}^d$ renormalizes the sum of elements in $\mathbf{x}^d$ to one. This process converges to a new $\mathbf{x}^d$ with sites weighted according to $\mathbf{W}$. We then renormalise $\mathbf{x}^d$ again to ensure that the given activity distribution is maintained,

$$\mathbf{x}^d = \mathbf{a}^d \, \mathrm{Diag}(\mathbf{x}^d) \, \mathrm{Diag}(\mathbf{A}\mathbf{x}^d)^{-1} \quad (8)$$

where the SxS matrix $\mathbf{A}$ encodes which sites belong to the same activity: element a_{ij} in $\mathbf{A}$ is one if site j belongs to the same activity as site i, and zero otherwise. Additionally, we can easily encode that sites are unavailable by explicitly setting the corresponding row in $\mathbf{W}$ to all zeros.

We can then solve for transition rates which satisfy the distribution of sites in the same way as before (section 3.1)

3.6 Using computed rates

Once rates are computed, the corresponding agent simulation is guaranteed to converge to a desired high-level distribution regardless of its initial state. In other words, agents may operate independently, selecting their next activity randomly according to the distributions in each column of $\mathbf{K}$. Thus, if an agent has completed

activity a_i, it can select a new activity by sampling from the distribution given in the i'th column of $\mathbf{K}$.

Additionally, the exponential distribution may be used to simultaneously model both the decision of where to travel next and the amount to time to spend at the next activity.

$$T_{ji} \sim P_{ji}^{exp}(t \mid \lambda_{ji}) = \begin{cases} \lambda_{ji} e^{-\lambda_{ji} t} & t \geq 0 \\ 0 & t < 0 \end{cases} \quad (9)$$

where $\lambda_{ji} = k_{ji}$. The choice of location will correspond to the site x_j with the soonest transition time.

$$\min_{j|e_{ji} \in E} T_{ji} \quad (10)$$

The advantage of this approach is that activities are automatically staggered in time. Otherwise, an additional parameter to offset agent choices is needed to avoid characters switching activities in unison at the same discrete timesteps.

4 Applications and Examples

The following examples, created with the Unity Game Engine [Unity 2013], show how the techniques in this paper can be applied to applications in games and animation.

Bird Behaviors

The methods in this paper can remove the need for tweaking crowd parameters. In this example, we animate a flock of birds who may peck, sleep, or jump at random. Animation clips for each activity have durations of 2 seconds, 8 seconds, and 1 second respectively and birds must travel to sleep (which takes 7 seconds). Figure 4 shows a state machine for controlling the bird's behaviors.

Although randomized, we want to the behavior of the birds overall to give the impression of pecking for worms. The purpose of the other activities is to add heterogeneity to the flock's appearance. However, because the sleeping animation is much longer than the others and involves a change of location, it is not intuitive to choose rates that maintain the majority of birds pecking. Suppose the animator begins by setting transition rates such that after finishing an activity, 90% of the time, birds switch to pecking and 10% of the time, birds switch to sleeping or jumping. One might expect that the high-level distribution would be $[0.90, 0.05, 0.05]$. In fact, the simulated proportion of pecking birds turns out to be far less (Figure 4(b)). The resulting visual appearance is also much different, resulting in too many birds sleeping and too many birds in flight (Figure 5).

Our approach can compute correct transition rates directly by modeling the durations (section 3.1.1) and travel time (section 3.1.2). By specifying that only 5% of birds should be in transition, the steady state becomes 85.5% pecking, 4.75% sleeping, and 4.75% jumping. The desired steady state is now maintained using the new rates shown in Figure 4(d). In a simulation of 100 birds (10 runs, 1000 time steps each), the average deviation from the specified distribution (where deviation is computed as $\|\mathbf{x}_t - \mathbf{x}^d\|$) is 0.04 with our approach, as opposed to 0.24 without our approach.

Food Court

The methods in this paper can be integrated into intuitive authoring tools for crowds. In this example, we animate the activities of a food court over the course of a day. Suppose that agents using the food court must order food before picking it up to eat it, that agents may get seconds, and that activities take some duration to complete (Figure 6). In this scenario, ordering food takes 10 minutes; picking up food takes 2 minutes; and eating takes 20 minutes. Because

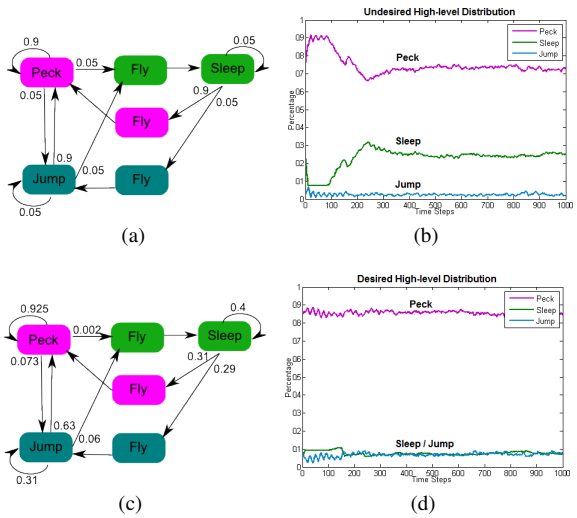


Figure 4: *Bird finite state machines. 4(a) 4(b) The top row shows a finite state machine for controlling bird behaviors which doesn't maintain the user's desired distribution. 4(c) 4(d) Our approach which models timing can directly compute rates which maintain a desired distribution.*

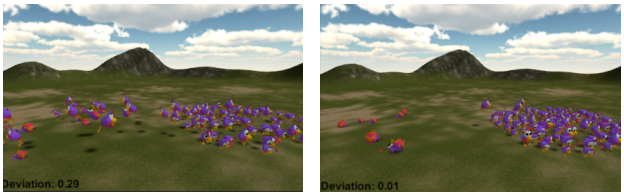


Figure 5: *Modeling timing. In this example, birds either peck, sleep, and jump. Left, a straight-forward approach where the next activity is selected according to the desired high-level distribution $[0.90, 0.05, 0.05]^T$ results in birds accumulating in the sleeping area. Right, by modeling the timing of the scenario correctly, we can directly compute the transition rates corresponding to the desired high-level behavior.*

the travel times between activities is much shorter than the durations for each activity, we do not need to model travel time for this example. For bookkeeping, the counts of agents performing each activity include agents transitioning to that activity.

A user does not need to know the underlying timing and spatial constraints to author characters using the food court. Instead, the numbers of agents performing each activity can be specified at a high-level using sliders and then automatically mapped to kiosks in the environment using the method in section 3.1.4. Additionally, the user can tweak the proportions in particular locations on the map using a crowd brush [Ulicny et al. 2005]. Once the designer is happy with the high-level appearance, she can associate the scene with a time of day and then click a button to compute random transition rates for maintaining the given appearance.

We compare using our computed rates against manually selected rates based on the desired high-level distribution. When activities can be performed in any order and take the same amount of time, there is no need to compute rates. Sampling directly from the desired distributions yields the desired high-level distribution. However, when activities take different durations and must be performed

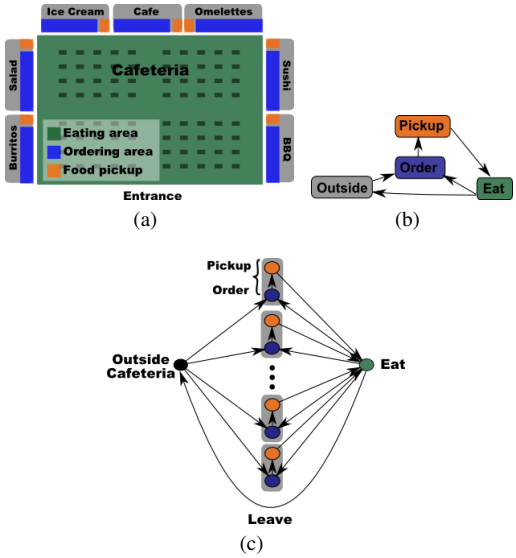


Figure 6: *Food court scenario. 6(a) shows the spatial layout of a central eating area surrounded by seven buffet-style food stations. 6(b) shows the activity graph for this scenario. Agents must order food and pick it up before eating. 6(c) shows the spatial graph which corresponds to the scenario map.*

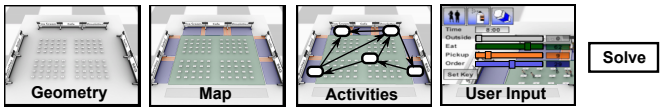


Figure 7: *Authoring keyframes. A user can author keyframes without knowing the details of the scenario. Our algorithm for computing rates depends on the geometry, map of where activities can be performed, and activity timing and sequence constraints. However, the user does not need to be aware of these details to author a scene. Instead, she can use high-level sliders and crowd brushes to set a desired high-level appearance for the crowd. At the click of a button, our technique computes rates for maintaining the given distribution.*

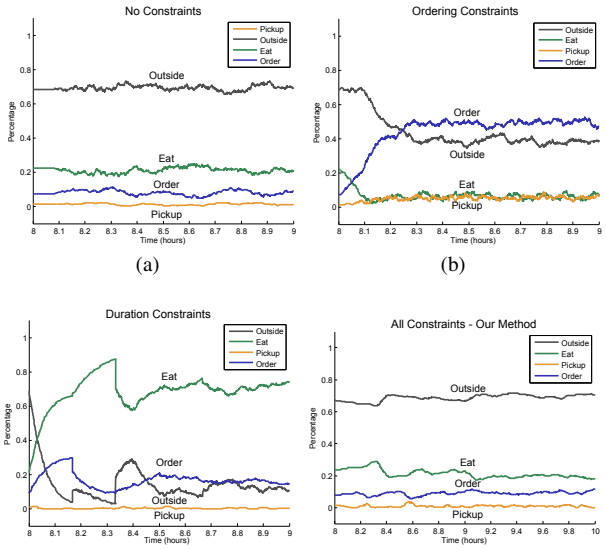


Figure 8: *Food court scenario. These graphs show changes in activity over time. The x-axis is the simulation time in hours. The y-axis is the proportions of agents who are either eating, ordering food, picking up food, or outside. 8(a) If activities can be executed in any order and have the same duration, there is no need to compute rates: sampling directly from the desired distribution yields the same high-level distribution (average deviation: 0.03). 8(b) However, if the sequence of activities matter, it is unclear how rates should be changed to yield a desired high-level appearance. In this example, we set the transition probability for invalid transitions to zero and renormalized. The high-level distribution now accumulates agents in the food ordering activity (average deviation: 0.43). 8(c) Similarly, differences between activity durations will also affect the high-level distribution. Here, the rates are the same as in 8(a), but the durations of each activity is different. Agents now accumulate in the longest activity, eating (average deviation: 0.73). 8(d) Our proposed method can directly compute valid rates which take into account both sequences and durations simultaneously (average deviation: 0.03).*

in sequence, it is unclear how rates affect the high-level appearance, and the computation of rates becomes necessary (Figure 8).

Wandering Zombies

In this example, we use the techniques from section 3.1.4 to distribute wandering behaviors across a large mall environment consisting of 69 locations. Because of the large number of spatial locations, setting densities across the entire mall by hand would be tedious. In this section, we show the result of automatically distributing agents across sites based on the desired activities set by the user.

In the first example (Figure 9(a)), we weight the distribution of agents across mall locations so that densities are highest in areas where sites are close together. To compute the elements of W, we use the travel distances between each pair of locations using a navigation mesh. The start and end points are located at the center (determined by bounding box) of each location. We then define our weights as

$$W_{ij} = \begin{cases} 0, e_{ij} \notin E \\ 0, \text{Distance}(x_i, x_j) > \text{Threshold}_{\max} \\ D_{\max} - \text{Distance}(x_i, x_j), \end{cases} \quad (11)$$

where $D_{\max}$ is the maximum distance between two locations in the environment and $\text{Threshold}_{\max}$ can be used to set a maximum distance for comparing locations.

In the second example (Figure 9(b)), we weight the distribution of agents across mall locations so that more agents are located in large areas than in small areas. In this case, the weights of $\mathbf{W}$ are the ratio of the areas between the target location and the start location.

$$W_{ij} = \begin{cases} \text{Area}(x_i)/\text{Area}(x_j) \\ 0, e_{ij} \notin E \end{cases} \quad (12)$$

In the third example (Figure 10), we distribute wandering behaviors evenly across all locations, but add preferences for closer locations over further ones using hard constraints (section 3.1.3). To keep the number of additional constraints smaller, we don't consider each pair of locations, but instead use a threshold around each location that separates near locations from far ones, e.g. $k_{ij} >= k_{kj}$ if site x_i is closer than our threshold distance and site x_j is farther than our threshold distance.

Medieval Gatherers

In this example, we show how the methods in this paper could be used to interactively control a group of characters, for example, as

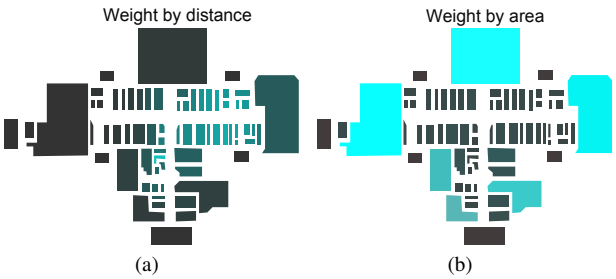


Figure 9: *Computing spatial distributions automatically. This figure shows maps of the mall environment used for our zombie demo. The shaded areas on the map indicate locations where zombies can wander: lighter regions have more zombies than darker regions. Left, we distribute wandering behaviors based on distance. Regions with many places close together will have more zombies than spread out regions. Right, we distribute wandering behaviors based on the region's area. Regions with more floor space will have more zombies than smaller regions.*

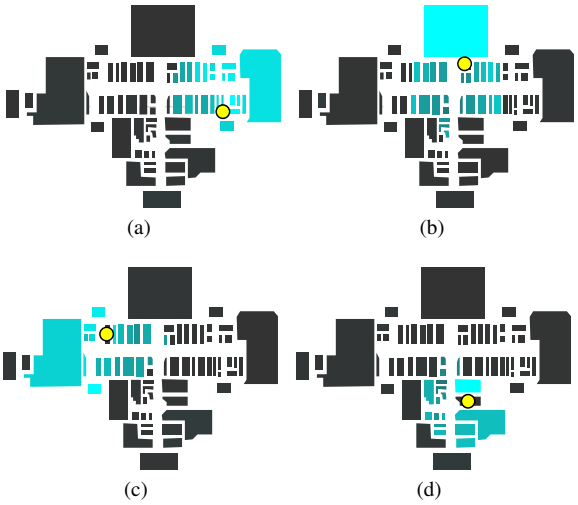


Figure 10: *Modeling preferences for closer locations. This figure shows maps of the mall environment used for our zombie demo. Locations in the mall are shaded to show how often they were visited from a starting location. For this demo, zombies were distributed evenly across the environment, but we added hard constraints so that zombies are more likely to visit closer locations than further ones. The start location is indicated with a circle. Lighter areas indicate locations more frequently visited from the start location.*

in a real-time strategy game. Here, the player assigns different proportions of agents to collect different resources, either wheat, gold, or wood. The underlying model for this demo is simple so that we may solve quickly for new rates (Figure 11). Rather than model the travel times separately, we lump the round trip travel time with the resource collection time into a single duration for modeling.

5 Performance

Because rates can be precomputed, no additional computations need to be performed at runtime, unless one wants to dynamically modify the distributions. We collected performance statistics with a Intel Core2 Duo 2.1 GHz with 4GB of RAM. The cost of computing rates for the demos in the previous sections can be found in Table 1. On the same machine, sampling for the next activities based on rates took no more than 1 msec. In general, the computation time increases with the number of nodes and number of constraints in the transition graph.

6 Discussion

We investigate a method for computing random parameters that supports direct animator control. Using these techniques can reduce the need for manually tweaking stochastic behaviors, can facilitate intuitive tools for crowd authoring, and can support interactively controlling characters in real-time.

The methods in this paper are best applied to large numbers of agents, such as background characters in open-world video games, or for controlling populations in aggregate, such as in real-time strategy games. In these cases, the high-level appearance of background characters is more important than simulating the internal processes of each agent and the stringent runtime and memory requirements favors lightweight and highly scalable approaches such as stochastic models.

However, the factors which make stochastic models appealing for large groups of background agents make them inappropriate for simulating small numbers of detailed agents. First, the accuracy of the model, e.g. how closely we match a desired distribution, becomes more accurate as the number of agents increase. In our demos, 100+ agents had good convergence. Second, it is difficult to build state machines for extremely large numbers of actions (future work could look at hierarchal stochastic models) and furthermore, this technique does not support reactive behaviors, such as those in response to the player. To support reactions, an agent could switch to a detailed AI script as necessary and then return to the more basic model. Small disturbances will not have a big effect on the aggregate appearance of the crowd.

Additionally, this model makes assumptions that an agent can always reach a destination, all interactions with other agents (such as tellers, shop keepers) are deterministic, and that resources are always available. Environmental changes can be handled by recomputing transition matrices to support more dynamic group behaviors.

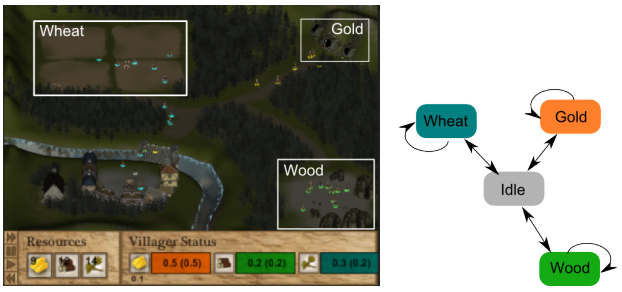
We only consider a subset of the types of distributions used for stochastic agent modeling. Future work could also look into modeling arbitrary distributions for durations and conditional probabilities between states.In the future, we may investigate how to apply these techniques to AI level-of-detail and for dynamically computing spawn entry and exit rates in moving regions around the player. We investigated this approach for authoring the appearance of crowds, but it might also be applied to sound authoring. For example, a sports game might use this approach to control the proportions of cheers, whistles, and shouts in response to game events. Other future work could verify the potential of this approach for massive scalability. Many massively multiplayer online games do not contain dynamic background characters. The robustness of this method and ability to run decentralized has the potential to let designers easily script daily routines for agents with minimal manual effort and little increased computational cost.

Acknowledgements

The authors wish to thank everyone who provided feedback and support for this project: Jan Allbeck, Norm Badler, Benedict Brown, Penfei Huang, Stephen Lane, Yusuf Sahillioglu, Ben Sunshine-Hill, and Rossana Queiroz. We also thank Fannie Liu and Corey Novich for their help with assets and videos. This work was

Demo	Activities	Sites	Nodes	Edges	Constraints	Time (s)
Birds	3	3	32	38	72	0.769
Food court - breakfast	4	16	105	134	499	8.76
Food court - lunch/dinner	4	16	105	134	499	8.65
Food court - coffee	4	16	105	134	499	8.73
Food court - closed	4	16	105	134	499	10.95
Zombies (no distance preferences)	2	69	69	4761	4899	3.32
Zombies (distance preferences)	2	69	69	4761	33876	651.23
Medieval	4	9	20	36	155	0.5

Table 1: *Precomputation times for our demos. Modeling distance preferences for the zombie mall environment takes much longer to compute because of the large number of additional constraints.*



(a) (b)

Figure 11: *Medieval gatherers. 11(a) shows the demo environment. There is a main village where people idle, 4 wheat fields for gathering wheat, 1 lumber yard, and 3 gold mines. The user can control the resource gathering across sites by assigning agents to activities. 11(b) shows the activity graph. Each gathering activity is executed as a self contained FSM which continuously fetches resources and brings them back to the village.*

supported by NSF Grant IIS-1018486 and ONR MURI DR-IRIS N00014-09-1-1052.

References

ADMIZA, J. 2001. AI madness: Using AI to bring open-city racing to life. *Game Developer Magazine* (January).

ALLBECK, J. M. 2010. CAROSA: A tool for authoring NPCs. In *Motion in Games*, Springer, 182–193.

BERMAN, S., HALÁSZ, A., HSIEH, M. A., AND KUMAR, V. 2009. Optimized stochastic policies for task allocation in swarms of robots. *Trans. Rob. 25* (August), 927–937.

BOYD, S., DIACONIS, P., AND XIAO, L. 2003. Fastest mixing markov chain on a graph. *SIAM REVIEW 46*, 667–689.

BROCKINGTON, M. 2002. Level-of-detail AI for a large role-playing game. In *AI Game Programming Wisdom*, Charles River Media, 419–425.

CURTIS, S., GUY, S. J., ZAFAR, B., AND MANOCHA, D. 2011. Virtual Tawaf: A case study in simulating the behavior of dense, heterogeneous crowds. In *IEEE Workshop on Modeling, Simulation and Visual Analysis of Large Crowds*.

GRANT, M., AND BOYD, S., 2013. CVX: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, Sept.

LIU, W., LAU, R., AND MANOCHA, D. 2012. Crowd simulation using discrete choice model. In *Virtual Reality Workshops (VR), 2012 IEEE*, 3 –6.

LOVAS, G. G. 1994. Modeling and simulation of pedestrian traffic flow. *Transportation Research Part B: Methodological 28*, 6, 429–443.

MARK, D., 2011. Using randomness in AI: Both sides of the coin. GDC AI Summit.

MUSSE, S. R., AND THALMANN, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 152–164.

ROSS, S. 1996. *Stochastic Processes*. Wiley and Sons, second edition.

SEWALL, J., WILKIE, D., AND LIN, M. C. 2011. Interactive hybrid simulation of large-scale traffic. *ACM Transaction on Graphics (Proceedings of SIGGRAPH Asia) 30*, 6 (December).

SHOULSON, A., AND BADLER, N. I. 2011. Event-centric control for background agents. In *International Conference on Interactive Digital Storytelling*, ICIDS'11, 193–198.

STOCKER, C., SUN, L., HUANG, P., QIN, W., ALLBECK, J., AND BADLER, N. 2010. Smart events and primed agents. *Proc. Intelligent Virtual Agents (IVA)*.

STYLIANOU, S., FYRILLAS, M. M., AND CHRYSANTHOU, Y. 2004. Scalable pedestrian simulation for virtual cities. In *Symposium on Virtual reality software and technology*, VRST '04.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3, 519–528.

SUNSHINE-HILL, B., AND BADLER, N. 2010. Perceptually realistic behavior through alibi generation. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

THALMANN, D., HERY, C., LIPPMAN, S., ONO, H., REGELOUS, S., AND SUTTON, D., 2004. Crowd and group animation. ACM SIGGRAPH Course Notes.

TOH, K., TODD, M., AND TUTUNCU, R. 1999. SDPT3 — a matlab software package for semidefinite programming. *Optimization Methods and Software 11*, 545–581.

ULICNY, B., DE HERAS CIECHOMSKI, P., AND THALMANN, D. 2005. CrowdBrush: interactive authoring of real-time crowd scenes. In *Symposium on Computer Animation (SCA)*.

UNITY, 2013. www.unity3d.com.

WANG, C., YAN, D., AND JIANG, Y. 2011. A novel approach for building occupancy simulation. *Building Simulation 4*, 149–167.