7D-04

Behavioral Agent Simulation using Contextual Action Multiple Policy Inverse Reinforcement Learning

Nahum Alvarez   Itsuki Noda

National Institute of Advanced Industrial Science and Technology (AIST)

Introduction

In this paper, we present a model for agent-based crowd simulation to emulate observed behavior using a novel technique we called contextual action multiple policy inverse reinforcement learning (CAMP-IRL). Crowd simulation has been subject of study due to its applications in the fields of disaster evacuation, smart town planning and business strategic placing. A known issue in people behavior simulation is that scripted simulations are often limited in their flexibility, so as a possible solution we can learn from data obtained in real scenarios using machine learning techniques and generate behavior patterns. We implemented a behavioral agent model into a large-scale crowd simulator. Applying our CAMP-IRL method to agents allows them to obtain reaction cues to different behavior patterns obtained from training data, generating different trajectories depending of their goals and the environment. Our method also provides a way to switch dynamically between behaviors and to navigate through unknown layouts, thus being a robust way for agents to behave realistically.

The Pedestrian Simulator

Our simulator generates a 2D version of the city where the pedestrians move across the map as a graph with links representing streets and nodes crossings. Once the simulator is running it shows a simulation of the agents traversing the city, walking until they reach an evacuation point determined by their configuration. The agents are represented by colored dots that change their tone between green when they are walking freely and red when they have to stop or walk slower.

In the simulator, an agent handler is used to generate the agents in the virtual environment and contains a module in charge of the agents' behavior, known as the CAMP-IRL module. The module contains two separated parts: one part is run before the simulation, which contains the CAMP-IRL method itself, and the other part takes control of the agents' behavior during the simulation. The next sections will describe in depth each one of the module's parts.

The CAMP-IRL Learning Process

IRL techniques work on domains that can be modeled by a Markov Decision Process (MDP) but have hidden reward functions (the reward function dictates what reward we can obtain from performing a concrete action when being in a concrete state). Hence, it is ideal to model human behavior, which usually is reward driven using unknown reward functions. However human behavior is not only directed by only one goal but many, with different rewards that are managed at the same time, IRL has potential to learn different behavior patterns, but need some adaptation as works with single rewards and well defined actions. Thus we based our method in a nonparametric Bayesian approach to the problem [1] extracting a number of clusters from the data, obtaining different reward and policy functions for each one of them. Also, we adapted the MDP to be able to work with contextual actions for the agents, used to avoid an explosion in the solution space.

We define Contextual Action Multiple Policy MDP (CAMP-MDP) as an MDP {S, $\mathcal{A}$, $\mathcal{T}$, γ , $\mathscr{R}$} using the standard definition of S as the set of states, the transition function $\mathcal{T}$ (s, a, s’) from one state to another by executing an action, and γ as the discount factor. We also defined the super set $\mathcal{A}$ (s) of actions as a function of a state s, and $\mathscr{R}$ (s,a) as a super set of Reward functions where s is a state from S and a is an action from the set $\mathcal{A}$ (s). The actions are contextual and the same action for different states will be likely different. The reason for this is that in our domain, each map node has a different number of possible links to take, so when translating nodes to states and links to actions, each link of the map would be converted into a different action. In order to avoid a combinatorial explosion in the solution space, we convert the actions directly related to links into contextual actions that represent different links depending on the node we are currently in. The CAMP-IRL algorithm is based on the Dirichlet process mixture model Bayesian IRL, but we adapted it to be able to work with the CAMP-MDP considering that each state will have a different action set. A Dirichlet process [2] is used to classify the trajectories into different groups which we call profiles, and then the reward is calculated for each profile using a Bayesian approach to the IRL method. The algorithm uses the next algorithm and formulas:

1. Initialize the profile set C containing K elements and the reward set $\{r\}_{k=1}^{K}$
    - I. The initial clusters (profiles) and their reward function are randomized. The reward function consists in a weight vector containing the weights of all the map features.
    - II. An initial policy is generated randomly from each reward. This policy consists in a vector containing the optimal action to perform for each node, and it

is obtained by calculating the value of performing the most optimal action a from the available actions in the state s, following the next function:

$$V^*(s) = max_{a\in\mathcal{A}(s)}\ \mathcal{R}(\mathrm{s,a}) + \gamma \sum_{s'\epsilon S} \mathcal{T}(\mathrm{s,a,s'})\, V^*(s')$$

2. For each element *m* in the trajectory set, select a new class candidate c_m^* using the following rule:

I. If the trajectory has no assigned class, generate a new one, and a reward function for it.

II. If it has one, obtain the most populated profile.

III. Assign the trajectory to the new class with probability

$$\frac{P(\chi_m|c_m^*)}{P(\chi_m|c_m)}$$

3. For each class k:

I. Create a weight vector candidate

$$r_k^* = r_k + \frac{\tau^2}{2}\nabla \log\big(P(\chi_k|r_k)P(r_k)\big) + \tau\alpha$$

where τ is a scaling factor and α is a number sampled from multinomial distribution (0,1).

II. Update the weight and value vectors with probability

$$\frac{P(\chi_k|r_k^*)P(r_k^*)g(r_k^*,r_k)}{P(\chi_k|r_k)P(r_k)g(r_k,r_k^*)}$$

Being the function g the gradient from the Langevin algorithm [1].

4. Repeat the process from (2) until convergence. Once finished, it is possible to use the obtained set of optimal policies for each profile to calculate the value vector as follows:

$$V^{\pi}(s) = \mathcal{R}(\mathrm{s},\pi) + \gamma \sum_{s'\epsilon S} \mathcal{T}(\mathrm{s},\pi,\mathrm{s'})\, V^{\pi}(s')$$

The value represents the expected reward of executing that policy on a node s.

The inputs of the CAMP-IRL method are the city map and a file containing the trajectories we want to train. The map is converted into a CAMP-MDP and the results are two files: one containing the weight of each map feature for each one of the discovered profiles, and another containing the value of each map node for each profile. These two files will be used by the CAMP-IRL agents to decide which path to take, and also to select the behavior profile they should have. This method is performed before the simulation as a pre-processing task, so even if it can take a long time depending of the complexity of the map it does not represent a big impact in the simulation speed as the decision process of the agents once we have these files is enough fast to be used in real time.

Behavioral Agents Model

Once the simulation starts, the agent handler creates the agents using our CAMP-IRL Agent Controller. The CAMP-IRL agents use three input files that direct their behavior; the first two files are the weight and value files from the CAMP-IRL process, and the third one consists in a goal database containing the locations they "want" to visit. This file contains an evacuation point to go after completing the goals.

Whenever an agent enters in a node, it checks if it is a goal. In case it is not a goal, the agent compares the value of the nodes connected to the current one and select randomly one node within the best value range. This range consists in the nodes that are inside a threshold from the highest valued node. Once the best node to go is selected, the agent moves to it.
Since the nodes' values in the file are related to a profile, the agent has to choose initially a profile. The selection of the initial profile is a two stepped process that happens when the agent enters the map. First, the agent obtains from the weight file the profiles that has the highest weight for the features associated to its goals. It does not only select the highest value, but the values that are within a threshold from it are also selected. The selected profiles will form the agent's profile list. Once completed this step, the agent chooses the profile from its list with the highest value in the current node (i.e. the first node the agent steps in).

If the agent entered in a node containing a goal, then the agent enters in an state of waiting, representing that the agent is satisfying its goal. The goal satisfaction time is given by the training process, who in parallel with the CAMP-IRL process, executed a linear regression method to learn from the waiting times present in the trajectories, giving a set of parameters to estimate such time depending the features present in the node. Once the goal is satisfied (meaning this that the goal satisfaction time has passed since the agent stopped), the agent verify if it has remaining goals. If all the goals have been satisfied, the agent evacuates by going directly to the evacuation point. If there are still unsatisfied goals, the agent proceeds to select a new profile in the same way that it chose one initially. However, before selecting one it updates its profile's list, as it has fewer goals now, so some profiles are not useful anymore. After deleting those profiles related with the satisfied goals, the agent chooses a profile form its list in the same way that it did initially.

As a final note, the agent also has a timeout in case it spends too much time wandering across the map without reaching any goal. If the timeout finishes, the agent will select a new profile from the whole set, with the only condition that it has to be different than the previous one. This represents the agent deciding that its previous actions where not advancing it to the goal, and it has to "explore" the map.

References

[1] Jaedeug Choi and Kee-Eung Kim. 2012. Nonparametric Bayesian Inverse Reinforcement Learning for Multiple Reward Functions. In Advances in Neural Information Processing Systems. 305–313.

[2] Radford M Neal. 2000. Markov chain sampling methods for Dirichlet process mixture models. Journal of computational and graphical statistics 9, 2 (2000), 249–265.