The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

Inverse Reinforcement Learning with BDI Agents for Pedestrian Behavior Simulation

Nahum Alvarez* Itsuki Noda*

* The National Institute of Advanced Industrial Science and Technology (AIST)

Crowd behavior has been subject of study in fields like disaster evacuation, smart town planning and business strategic placing. It is possible to create a model for those scenarios using machine learning techniques and a relatively small training data set to identify behavioral. We implemented a BDI-based agent model that uses such techniques into a large-scale crowd simulator, and apply inverse reinforcement learning to adjust agents' behaviors by examples. The goal of the system is to provide to the agents a realistic behavior model and a method to orient themselves without knowing the scenario's layout, based in learnt patterns around environment features.

1. Introduction

In this paper, we present a BDI agent model that includes the use of inverse reinforcement learning (IRL from here on) for the agents' decision making process, training them with knowledge learnt from previous data. The context of our research lies in the domain of pedestrian simulation on cities, with the objective of extracting knowledge of the pedestrian flow around concrete points of the map that contain certain features, like shops or restaurants. With this information we would be able to decide which spot is best for certain type of feature and how would change pedestrian affluence if we add new features or modify the existent ones. This is done by analyzing the crowd movement flow according of map features and agent characteristics. Previously IRL techniques has been used to calculate trajectories and plan movements, but as far as we know its use in agents based simulators or feature map optimization has been sparse.

2. Related Work

One of the applications of pedestrian simulation is smart city and business location planning, where we analyze people movements and behavior in their daily city life and obtain insight of what places attract more people and how different types of locations affect their actions. Then, once we have a model of the patterns that people follow, we can simulate them in other environments to assess the effectiveness of certain spots, or predict how a future potential business or facility could perform at different locations.

Using agents for the simulation is a popular approach due to be able of generating complex behavior with simple design and also escalates well, being appropriate for large scenarios. A common strategy is to model the agents with a dual behavior system controlling two types of movement (or behavior): micro and macro movement. The first one deals with collision avoiding in the near space and adjusting the agent's velocity in the crowd's flow, and the second is the one in charge of driving the agent towards its goal, creating and updating its route and taking care of the decision making process [Torrens et al., 2012]. Our simulator handles micro movements in an autonomous way, with its own subsystem where agents adapt to the crowd flow, and also allows to control macro movements using behavior scripts. To achieve our goal, we are focused in macro movements as it is the module that creates the agent's behavior.

In order to learn behavior patterns, we can take advantage of machine learning techniques. Concretely, apprenticeship learning methods have been widely used in intelligent agents' systems to train them to perform tasks in changing environments. Simulating people's behavior and not only trajectory planning is a difficult task, as their movements are governed by hidden rules and oriented to goals that may be hidden as well. We can observe strategies to emulate different behaviors for agents in [Faccin et al., 2017] but it is a static model with pre-designed driving styles, having the problem of not being able to simulate unplanned behaviors. There are previous works where agents are given a behavior cognitive model for pedestrians like in [Luo et al., 2008], [Martinez-Gil et al., 2017] or [Crociani et al., 2016], but they are specific to its domain, escalating badly, or they take as a given the reward or utility functions that drive the agents behavior, which often is unknown in complex scenarios. Using IRL is appropriate to overcome this issue, because IRL methods work on domains where the reward function is hidden. Hence, it is ideal for modeling animal or human behavior [Ng et al., 2000]. Works that use IRL to manage agent's behavior were sparse, but some researchers are starting to use it [Šošić et al., 2017]. IRL not only allows to train agents into achieving concrete goals, but also can learn different behavior patterns, as it is shown in [Abbeel & Ng, 2004] where driving styles are learned by an agent.

3. Pedestrian Simulator

We developed an agent based pedestrian simulator called CrowdWalk to perform crowd behavior for generic scenarios [Yamashita et al., 2009]. It can simulate movements of more than 1 million agents in a large area like complex building or town blocks in a city. Maps and agents' behaviors are configurable so that we can conduct simulations with various situations of maps

Contact: Nahum Alvarez, National Institute of Advanced Industrial Science and Technology (AIST), 2-3-26 Aomi, Koto-ku, Tokyo 135-0064, Japan
Tel: 080-3301-5860, E-mail: nahum.alvarez@aist.go.jp

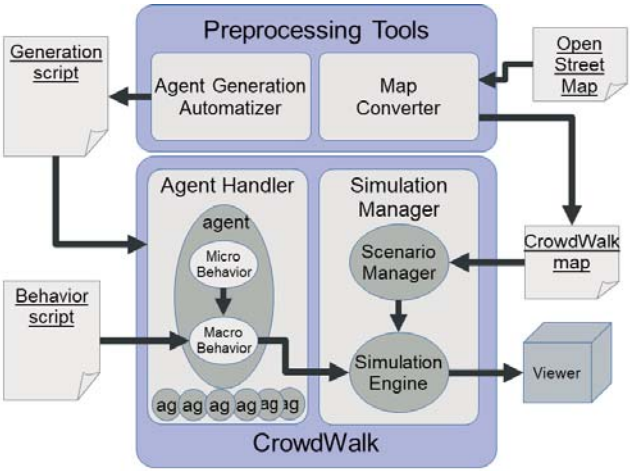

Figure 1: Architecture diagram of CrowdWalk. It has two main modules: one managing the agents and other in charge of the environment simulation.

and policies of agents. The architecture of CrowdWalk is depicted in Figure 1, and shows its principal work modules. We omitted from the diagram the modules related to the IRL process as in this section we want to describe in depth the application where we built learning agents on. We will focus in the IRL process flow and the new agents with detail in the next section.

CrowdWalk has two main working sub-systems: an Agent handler and a Simulation manager. The Agent handler contains an agent factory, that generates one agent per pedestrian. Each agent contains its own decision model that its composed by the micro and macro behavior modules. The micro behavior module in each agent takes care of micro movements automatically, calculating when an agent has to stop, walk slower or even try take other route due to be unable to continue its original path. However, the final decision of selecting other path is left to the macro behavior module, and actually, the macro behavior module is able to override any orders from the micro movements module if deems so.

The macro behavior module is the one in charge of calculating the agent's route to its goal, and updating it if necessary. This module contains some predefined behaviors, built using a nested hierarchy where the most basic behavior consists in an agent that just calculates the shortest path to the goal without taking in account the degree of agglomeration of each link, and other behaviors increase in complexity by changing the path if there is too much agent density, or avoiding roads it used previously. On the top of the most complex class, there is a special class that enables the agent to be controlled externally, the agents' Behavior Script. In our current research we created an IRL agent for this class that will be described in depth in the next section.

To create the agents, the agent handler reads from the agent generation script, a file containing the rules of agent generation containing the number of agents needed, which type of agent will they be, which agent behavior script will use (if any), which point they come from, and which point they are going to. CrowdWalk also has a tool to automatically generate this file by providing some simple rules and the map model.

The other important sub-system is the Simulation Manager, which is the one in charge of actually run the simulation. It receives as input a configuration file containing the scenario information and recreates it as a virtual environment. Internally, CrowdWalk uses a Network-based model capable of high speed simulations of large numbers of agents. The model of the map received by the Simulation Manager consists in a custom xml that describes the map in the form of a road network represented by nodes (intersections) and links (road paths) composing a graph. A link has length (how long agents need to walk from a end to another) and width (how many agents can walk in parallel), and can be two-way or one-way. Nodes and links have labels to indicate goals and other features information, like what kind of facilities are on that location. The xml model can be created automatically using a tool included in CrowdWalk that converts maps obtained from the open source software Open Street Map into our custom format. This allows us to use any possible city map in the world with no additional effort.

The Agent handler is called by the Simulation manager in order to generate the agents in the virtual environment following the rules given in the agent generation script. When running, the simulation engine represents the scenario as a graphic simulation where the agents will behave according to each one's own behavior modules, allowing pausing the simulation at any moment and inspecting every element part of it.

4. Agent Model

Reinforcement learning techniques work on domains that can be modeled by a Markov decision process (MDP, from here on after). MDP are defined by a tuple M = {S, $\mathcal{A}$, $\mathcal{T}$, γ, r}, where S is the state space of the model, $\mathcal{A}$ is the set of actions that can be performed, $\mathcal{T}$ is the transition function, which returns the probability of transition from one state to other given a concrete action, and usually is given in the form of a matrix, r is the reward function that generates a reward value from reaching a state, and γ is a discount factor, that is applied when calculating accumulated reward through consecutive actions. When working with models with an unknown reward function, IRL methods provide us a way to obtain it. In order to get r, it is also provided a set of expert trajectories T, consisting of “paths” composed of pairs of states and actions.

We developed an automatic module that converts the city map used in CrowdWalk into a MDP, ready to be used by our IRL method. This tool translates map nodes into states, and creates a possible action for each link that it has. Also, in order to optimize the model, all the nodes that only have two links are trimmed, as there are no other possible decisions once a pedestrian enter in one other than continue walking or going back. Once we have the MDP model for the map, we run the CrowdWalk IRL module, that consists on a modified version of the maximum entropy IRL method found in [Alger, 2015], adapted in order to use a variable number of possible actions on each state, as each map node has a different number of possible links to take. The input of this module is the MDP representing the current map, and a file containing the training trajectories (i.e., the pedestrian behavior) we want to train. The resulting product of the module is a file containing the optimal policy function derived from the reward function generated by the IRL algorithm. This policy function takes the form of a lookup table stored in a file, which will be used by the system's agents.

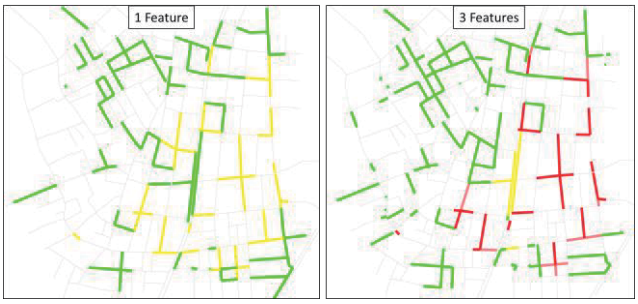

Figure 2: The rewarded links in the map when training with one and three different types of features.

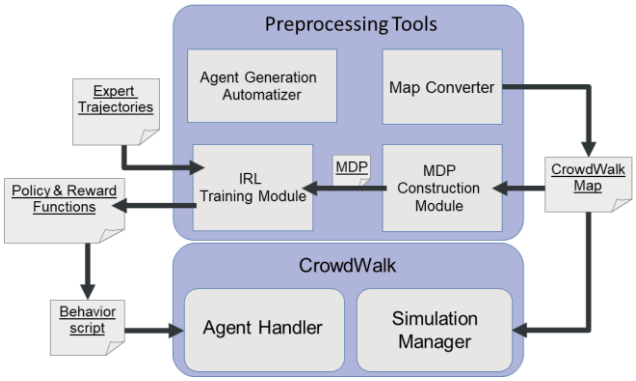

Figure 3: The IRL process integrated in the system. We added two modules to the system in order to train the agents.

An example of how the policy influence the map navigation can be seen in Figure 2. The maps shows a portion of the city of Tokyo containing features that were classified in three groups: shops, restaurants or entertainment. In the leftmost map we trained only shopping behavior, and in the rightmost map we trained all the three features. Links in red have a probability greater than 80% to be selected as the next path, in yellow when is between 40% and 80%, and in green when it is lower than 40% but greater than 20%. As we can see in the figure, when training with three groups of features, new colored links appear compared to the one-featured map, and also increases the probability of taking the links in areas with different features are overlapping. We want to note that colored links appear over the whole map, and not only in the areas covered by trained routes, which where a 60% of the map.

The generation of the MDP model and the reward and policy functions is performed before the simulation as a pre-processing task. Thus, even if this pipeline can take a long time depending of the complexity of the map, it does not represent a big impact in the simulation speed. The process flow once included in the architecture we explained in the previous section is depicted in Figure 3.

We developed a type of agent, called IRL agent, that works with an input behavior script and the IRL's resulting policy and decides which link to take on the probabilities contained in it. The agent follows a BDI architecture, having a list of desires defined in the input behavior script as goals (that should be in line with the trained behavior), and a memory where its beliefs are stored and consisting in the optimal policy obtained from the

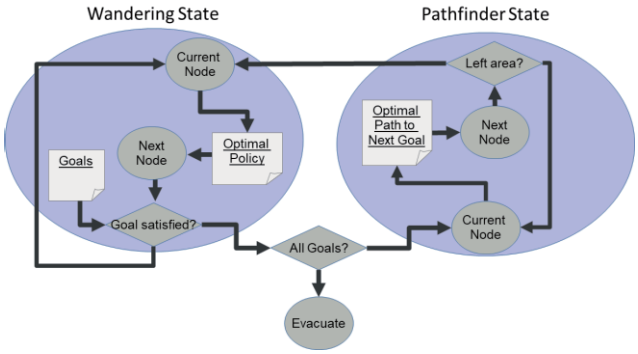

Figure 4: The algorithm followed by the IRL agents.

IRL process (stored as a lookup table), and the goals and features previously visited. The goals can be generic, like "visiting any restaurant", or concrete, like "visiting the restaurant located in the node labeled as nd00327". Each goal in the list also contains the conditions for its satisfaction, which can be reaching the goal, staying in the goal a defined time, reaching a number of goals of that type (in case of the generic goal), or a combination of them. Also, the script contains an evacuation point, where the agent will go after completing its goals. The decision making algorithm is shown in Figure 4. An agent starts in the state called "wandering", where he obtains from the policy table a list containing the probabilities of taking each one of the available links on its current node. Then the agent chooses which path it will take based on that probability list. Whenever the agent visits a goal node, it checks and updates its satisfaction conditions. If that goal is satisfied, the agent enters into another state, called "pathfinder", which makes it go to its next goal using the most optimal path. However, it leaves this state as soon as it has left the area containing the previous goal, and returns to the "wandering" state. The agents decide they left the area of a goal if they surpassed a distance threshold from that goal. Also, goals already visited do not count as new goals when returning to them (in case the agent has to visit a concrete number of them). Once all the goals have been completed, the agent enters again in "pathfinder" mode and goes straightly to its evacuation point, leaving the map.

5. Preliminary Validation

We performed a preliminary set of tests to validate if the intended behavior is observed in the agents. We selected a map from a portion of Tokyo were we located a total of 36 features, classified in three groups: shops, restaurants or entertainment. We generated by hand 3000 routes representing pedestrians making errands (we observed in previous tests that using bigger data sets did not generated better performance). Then, we performed three different sets of simulations using 3000, 6000 and 10000 IRL agents. The agents started from 20 different random points and have the goal of visiting 4 featured spots. Also, in order to compare the agents behavior, we performed a parallel simulation using other type of agent called "pathfinder agent", capable of calculate optimal routes and going directly to its goals (so it is always in the pathfinder state). We designed routes for those agents placing waypoints on certain featured

Table 1: Agent coverage of the featured nodes in the map.

Agent Type	Total Nodes Visited	Average per Agent
IRL	100%	7.11
Pathfinder	75%	2.23

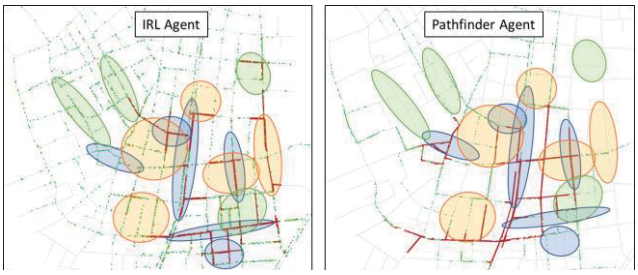

Figure 5: Comparison between the behavior of the agents. The colors used mark different areas: yellow for shops, blue for restaurants, and green for entertainment.

areas, making them to walk to those points in order and then evacuate at the final point.

Figure 5 shows a caption of the simulation with the three types of featured areas marked with different colors. As the pathfinder agents are programmed to only follow the best path, they avoid other similar featured areas that are not part of it. They also concentrate in other areas that do not contain features, but are just the optimal path to the goal (for example, the big avenue that runs vertically across the map, which is a navigational hub). On the other hand, IRL agents tend to disperse themselves when they are searching for featured areas, and concentrate in all the featured areas in the map, especially if they overlap. Even untrained areas are populated. Also, by observing the reward values of the links on the map, we can detect which places are more optimal for certain business.

We also extracted some population density numbers from our tests, which can be seen in Table 1. The IRL agents visited all the 36 featured nodes while the pathfinder agents left 9 nodes untouched. Not only that, each IRL agent visited more than three times the average number of nodes per agent than the Pathfinder agents. Naturally, the pathfinder agents only went to the featured nodes that were by chance in their path to the goal, but this means that in the real world, not all features are in the most optimal paths and agents that does not take in account behavioral patterns will ignore places that may be important.

6. Conclusions

In this paper we presented an agent-based pedestrian simulator that uses inverse reinforcement learning in order to imitate behavior patterns learned from expert's trajectories. Our goal is to provide better understanding of crowd simulation and applying it to city and business smart planning. We performed a set of preliminary experiments obtaining promising results, especially when replicating the intended behavior. Our IRL agents visit spots in the map that are ignored by agents that only calculate the most optimal route to the goal, generating more realistic behavior. We are planning to validate the behavior of our agents by comparing them with live data from real pedestrians in the next steps of our research. We think IRL techniques opened an interesting path that was not enough explored although they are well known from long ago, maybe because a lack of simulation technology.

References

[Abbeel & Ng, 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1. ACM, 2004.

[Alger, 2015] Matthew Alger. Deep inverse reinforcement learning. 2015.

[Crociani et al., 2016] Luca Crociani, Giuseppe Vizzari, Daichi Yanagisawa, Katsuhiro Nishinari, and Stefania Bandini. Route choice in pedestrian simulation: Design and evaluation of a model based on empirical observations. Intelligenza Artificiale, 10(2):163-182, 2016.

[Faccin et al., 2017] João Faccin, Ingrid Nunes, and Ana Bazzan. Understanding the Behaviour of Learning-Based BDI Agents in the Braess' Paradox, pages 187-204 Springer International Publishing, Cham, 2017.

[Luo et al., 2008] Linbo Luo, Suiping Zhou, Wentong Cai, Malcolm Yoke Hean Low, Feng Tian, Yongwei Wang, Xian Xiao, and Dan Chen. Agent-based human behavior modeling for crowd simulation. Computer Animation and Virtual Worlds, 19(3-4):271-281, 2008.

[Martinez-Gil et al., 2017] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Emergent behaviors and scalability for multi-agent reinforcement learning based pedestrian models. Simulation Modelling Practice and Theory, 74:117-133, 2017.

[Ng et al., 2000] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In Icml, pages 663-670, 2000.

[Šošić et al., 2017] Adrian Šošić, Wasiur R KhudaBukhsh, Abdelhak M Zoubir, and Heinz Koeppl. Inverse reinforcement learning in swarm systems. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pages 1413-1421. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[Torrens et al., 2012] Paul M Torrens, Atsushi Nara, Xun Li, Haojie Zhu, William A Griffin, and Scott B Brown. An extensible simulation environment and movement metrics for testing walking behavior in agent-based models. Computers, Environment and Urban Systems, 36(1):1-17, 2012.

[Yamashita et al., 2009] Tomohisa Yamashita, Shunsuke Soeda, and Itsuki Noda. Evacuation planning assist system with network model-based pedestrian simulator. In PRIMA, pages 649–656. Springer, 2009.