Inverse Reinforcement Learning for Agents Behavior in a Crowd Simulator

Nahum Alvarez(✉) and Itsuki Noda

The National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan
{nahum.alvarez,i.noda}@aist.go.jp

Abstract. Crowd behavior has been subject of study due to its applications in fields like disaster evacuation, smart town planning and business strategic placing. However, obtaining patterns from the crowd to make a working model is difficult, as it requires an enormous quantity of data from observation and analysis and is impractical in many scenarios due to logistic and legal issues. Machine learning techniques are a good tool to overcome these difficulties, using a relatively small training data set to identify patterns, allowing crowd agents to react to similar situations accordingly. We implemented a behavioral 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.

Keywords: Pedestrian simulation · Inverse reinforcement learning · Multi-agent systems

1 Introduction

Crowd movement is a topic whose study has a large number of applications in diverse domains. Naturally, to experimenting or testing scenarios with real people presents a number of logistic problems and is generally not practical, or even infeasible in certain instances. Therefore, a widely accepted solution is to use a simulator to replicate the desired scenario. Then, the simulation can be used for extracting crowd behavior patterns and predicting its movement. This could help in improving our understanding of real life tasks like city planning, disaster prevention, or business strategy. Agent based models are commonly used to perform the simulations, due to its flexibility and scalability, and allow to produce complex crowd interactions using simple action patterns. However, human behavior is a factor difficult to model: people's actions are goal-driven but those goals are not usually visible and do not follow optimized plans often. Also, scalability and performance requirements arise when we need to work in scenarios involving large numbers of humans, so even basic behaviors pose a

D. Lin et al. (Eds.): MMAS 2018, LNAI 11422, pp. 81–95, 2019.
https://doi.org/10.1007/978-3-030-20937-7_6

challenge to researchers. A possible way to solve this problem is to use machine learning techniques on available similar data in order to give the agents a way to react to new situations.

In this paper, we present a 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, and predicting which places are more appropriate for certain business. We aim to deploy a large number of agents with different profiles depending of their goal (shopping, work, entertainment) and observe how their behavior is influenced by the features in the environment. 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.

We developed a crowd simulator designed to generate pedestrian movement in city scenarios using real world city maps that originally used simple scripted agents to calculate trajectories, and we expanded it by adding a behavior module that works with IRL and is used by the agents to decide which path take. The decision process is influenced by the preexistent features in the map generating similar behavior for places with similar features. This module also allows the agents to traverse maps whose layout is not known. Also, once they have learned behavior patterns related to the map features, they can be put on a different map and behave the same way they did in the original scenario.

The rest of the present document is organized as follows: Sect. 2 contains a review of previous work on reinforcement learning used for agent behavior and pedestrian simulators and the techniques they use. Section 3 describes in detail our crowd simulator and its architecture, and Sect. 4 presents our agent model and the IRL method that generate their behavior. Section 5 contains the tests we performed to validate the system and the results we obtained from them. Finally Sect. 6 contains the conclusions of our research.

2 Related Work

Crowd simulation have been recently the object of rising interest because it can deal with a number of important problems in our society. For example, traffic simulation can be used to improve transportation systems and networks, and also in obtaining solutions to lowering car pollution [6]. Pedestrian simulation is useful to design evacuation strategies and identifying potential problems in concrete scenarios like natural disasters or terrorist incidents, like [22] or [12]. In these works we can see that not only an accurate model is needed, but a high

degree of scalability is mandatory, as simulating hundred of thousands people requires many resources in terms of computational power. Different models has been used to achieve efficient and accurate simulations: for example, [3] shows the simulation of the crowd flow in a train station during an event using the Cellular Automata model, or our own system that uses the Social Force model described in [7]. However, every model has certain limitations, like the Social Force model having issues representing realistic collision behavior at high densities due to the specification of the repulsive interaction forces, or the Cellular Automata model having issues at modeling agents at high or non homogeneous velocities [17]. Solutions of these shortcomings have been proposed by extending the models, often making them domain-specific to some degree. For example, the Social Force model was upgraded in [9] to describe detailed velocity control of pedestrians, and other works like [23] and [11] add a collision-prediction/-avoidance force model.

There is another topic where pedestrian simulators can be applied, which is the one we are interested in: city and business location planning. We want to analyze people movements and behavior in their daily city life, obtaining 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 could 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. We are interested in such kind of application and it is the objective of the present work.

Aiming to that goal, we developed CrowdWalk, a crowd simulator that uses a multi-agent model to represent pedestrians. Using agents is a popular approach due to be able of generate complex behavior with simple agent design and also escalates well, being appropriate for large scenarios. There are a wide range of works in pedestrian simulation with agents, using different techniques. Systems based in video analysis work well, like the one in [24], but in order to remain practical it narrows their domain, using tile location and pre-generated trajectories. Another 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 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 [21]. 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.

However, a basic action for an agent macro movement like calculating the most optimal route to a goal in a crowded scenario is a complex task: an a priori calculated optimal route can become much slower if an enough large number of agents take it, and this is the simplest problem that could arise; nevertheless, this is not what we intend as we only aim to replicate pedestrian behavior. Learning

such behavior patterns is an interesting question: humans usually do not take the most optimal route, and even congestion can be seen as a positive factor ("if there are many people, is because is good") as noted by [4]. In such kind of domains, 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 like [18] or [20]. As we noted, 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 [6] 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 [14,15] or [4], 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 to model animals and humans behavior [16]. Interestingly, the works that use IRL to manage agent's behavior are sparse, but are recently some works are starting to use it [19]. IRL not only allows to train agents into achieving concrete goals, but also can learn different behavior patterns, as it is shown in [1] where driving styles are learned by an agent. There are a number of algorithms to solve IRL problems, like the ones in [8] or [10]. We decided to use the maximum entropy approach [25] because it works well when we do not have much information about the solution space, as we are dealing with city scenarios with a layout a priori unknown by the agents. There are methods that perform better, like [5] which it works on a subset of MDP, but it does not match well with our domain, or [13] which could be interesting to apply in future instances of our research.

3 Pedestrian Simulator

CrowdWalk is a pedestrian simulator we developed to perform crowd behavior prediction in disaster scenarios in order to identify potential bottleneck issues when coordinating evacuation routes. Aside of this type of scenarios, CrowdWalk was actually designed for generic uses, so it is possible to create pedestrian simulation with other purposes. In CrowdWalk, each agent (pedestrian) walks on a map toward its own goal. 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 and policies of agents. The architecture of CrowdWalk is depicted in Fig. 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. First, CrowdWalk has two main working modules: an Agent handler and the Simulation

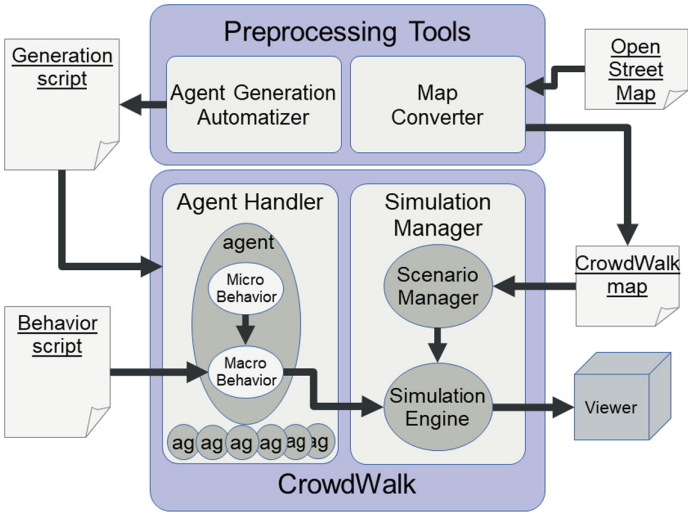

Fig. 1. The architecture diagram of CrowdWalk. It has two main modules: one managing the agents and other in charge of the environment simulation. Each one of these modules is configured by different files containing the specifications of the simulations. Some of those files can be automatized using the system preprocessing tools to some extent for easier construction.

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 speed of an agent is given in relation with the density of the link he is currently in, being slower as the link is more crowded, until a maximum capacity limit where it is not possible to continue advancing. Concretely, it is determined according to the social force model as following:

$$\frac{dv_i}{dt} = A_0 \left(v_i^* - v_i\right) - A_1 \sum_{j \in H_i} \exp\left(A_2 \left(s - \|p_i - p_j\|\right)\right)$$

where v_i, v_i^* and p_i is the current and max speed, and the current location of agent i, respectively. §H_i is the set of agents located in front of the agent i. And s is the radius of a personal space. A_0, A_1 and A_2 are constant parameters whose purpose is to adjust the formula. Through experimental observations in real scenarios we set these parameters, the personal space radius and the max speed to a default of $A_0 = 0.962$, $A_1 = 0.869$, $A_2 = 4.682$, $v_i^* = 1.023$ and

$s = 1.044$, however these parameters can be customized in the configuration files of CrowdWalk.

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 by an external script. Using this last behavior class, CrowdWalk can use external scripts defining agent's macro behavior, allowing more flexibility.

The other important module within the system is the Scenario Engine, 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. We simplified the map into a 1-dimensional network consists of nodes and links instead of 2D free space. Our main focus is to investigate phenomena and behaviors of a large scale crowd in a large area, so we need to execute exhaustive simulations with a large number of configurations. Therefore, we chose a light computational 1D model rather than a 2D model one. However, it is capable of simulate 3-dimensional structures as well, being capable of representing the internal layout of a building.

CrowdWalk uses five configuration files to setup simulation environments. The five files are described as follows:

Properties file: specifies top-level configurations of the simulation. Other configuration files listed below are specified in this file.
Map file: specifies a map for the simulation.
Generation file: specifies the rules to generate agents. Each generation rule specify agent classes, other parameters like the max speed and personal space radius, populations, goals, and generation time independently.
Scenario file: specifies a sequence of events occur during the simulation.
Fallback file: specifies the default values for the simulation parameters, like the ones in the social force model.

The model of the map consists in a custom xml that describes the map in the form of a road network represented by nodes (intersections) and links (road path) composing a graph. A link has a 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 can have *tag*s as 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[1] into our custom format. This allows us to use any possible city map in the world with no additional effort.

[1] https://www.openstreetmap.org.

The agent generation script is a file containing the rules of agent generation, i.e. what number of agents are created, which type of agents will be and if they are not any default type, which agent behavior file the system will use, 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. Finally, the agent behavior is contained in a script and describes the macro behavior of the agent. This script optional, as there are a number of predefined agents, like agents that move randomly, or agents that move directly towards their goal. For our current research we created an IRL agent we will describe in depth in the next section.

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 (like the current internal state of any agent, and any node or link).

Once CrowdWalk is running it shows a simulation of the agents traversing the city map until all the agents reach their goal point; When the agents are walking freely they are colored green, but when they have to stop or walk slower they become red, showing bottlenecks in the map. The simulator screen allows to pause the simulation and examine every agent, map node or link information. The simulation is run at accelerated time, but it shows the real world time it is taking given the velocity of the agents. When finished, it records each agent path in a log with timestamps for posterior replication or analysis purposes.

4 Agent Model

IRL techniques work on domains that can be modeled by a Markov decision process (MDP, from here on after) and are used to learn its hidden reward function. MDP are defined by a tuple $\mathrm{M} = \{\mathrm{S},\ \mathcal{A},\ \mathcal{T},\ \gamma,\ \mathrm{r}\}$, 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 applies 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 an instance of the IRL algorithm, using a modified version of the maximum entropy method found in [2]. We adapted this module 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 we want to train. It is possible to run this process using trajectories for each available behavior we want to train, but also we can run it only once training all the behaviors at the same time. For example, we can train shopping behavior using routes that go to the shops on the map, preferring shops hubs like malls or shops surrounded by other entertainment facilities, or business behavior by training the routes that working people would do, preferring wide avenues over small and crowded streets; we can do this once per each behavior or put together all of the trajectories and train them as a whole. 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. All of these actions are performed before the simulation is executed as a pre-processing task. Thus, even if this pipeline can take a long time depending of the complexity of the map (about one hour for a map for one Tokyo district, with around 2000 nodes), it does not represent a big impact in the simulation speed as the policy selection once we have this file is enough to use it in real time. The process flow once included in the architecture we explained in the previous section is depicted in Fig. 2.

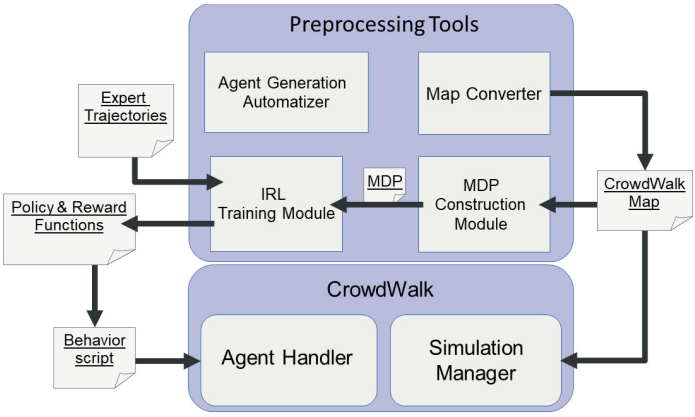

Fig. 2. The IRL process integrated in the system. We added two extra modules to the system in order to create a MDP based in the simulated scenario and train the agents with expert trajectories. This pipeline integrates naturally with the architecture of the system.

As we described in the previous section, when a simulation runs in CrowdWalk, the agents have a macro behavior module which is in charge of the decision making process by selecting a type of behavior for the agents. 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 behavior script contains a list of goals, which describes the features the agent wants to visit. They can be generic (like "visiting three restaurants") or a concrete one (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 agent has access to the optimal policy learned from the IRL process (stored as a lookup table), and also the set of rules contained in its behavior script, whose decision making process is shown in Fig. 3. The agent starts in an state called "wandering". In this state, on each map node, the agent obtains from the policy table a list containing the probabilities of taking each one of the available links on that 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 enter briefly 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 featured area containing the previous goal, and returns to "wandering". The agents decide they left the area of a goal by have a distance threshold from the goal they are leaving.

The rationale behind the pathfinder state is to avoid returning to the previously visited nodes (on a side note, goals already visited do not count when returning to them), as the nearly policies would drive back the agent to them. Currently this is controlled by the distance threshold, but in future installments of the system we will use multiple policy functions with a smart selection method between 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 two aspects of our method: first, if the intended behavior is observed in the agents, and how the policy and reward values are distributed on a featured map, and second, that the agents are capable to reach their goal points. Additionally, we aimed to identify unpredicted issues and deviations from our expectations.

First, we selected a map from a portion of Tokyo containing commercial areas, touristic spots, a train station, and residential zones. In the map there were a total of 36 features, which we classified in three groups: shops, restaurants or entertainment. We generated by hand 3000 routes representing pedestrians making errands (we observed in previous tests that training that number of

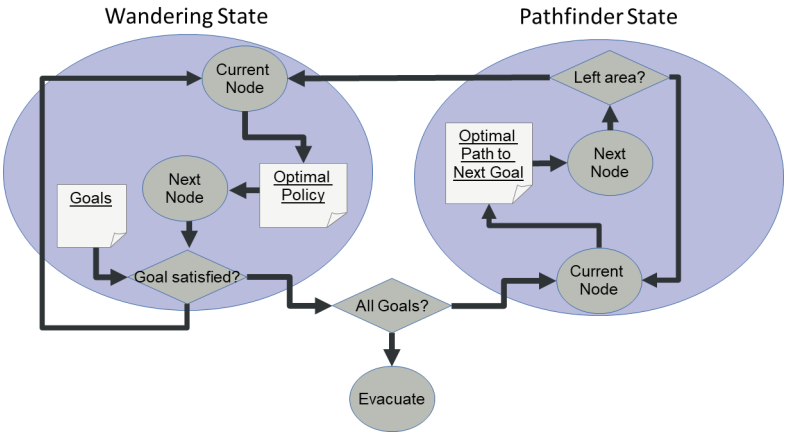

Fig. 3. The algorithm followed by the IRL agents: once they reach a map node, they use the optimal policy function to choose the next node they will go to. If they satisfy a goal, they leave the area and start wandering again. Also they check their list of goals and leave the map when all of them were visited.

sample routes results in a better performance than using bigger data sets). The routes start on 20 random points in the map, covering a 60% of it (a bit more, they cover 165 nodes from 273) and evacuate on a designated area. We are aware that for a complete validation we will need to use trajectory data from real humans. However, we are expecting to obtain a data set containing pedestrian behavior from other domains, like a department store or a fireworks festival, so we will adapt our simulator to represent these new environments and perform more definitive tests.

We prepared two different maps for training the routes: one was designed to train only shopping patterns; i.e. we deleted from the map any feature not classified as shop. The second map was prepared using the original, containing all three groups of features. We trained the routes in these two maps, and generated two different optimal policy files. Figure 4 depicts how the rewards influence in the map. 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, training with three groups of features generates new colored links but also increases the probability of taking the links in areas where different features are contained. We want to note that colored links appear over the whole map, and not only in the areas where routes were trained.

Then, we performed three different sets of simulations using 3000, 6000 and 10000 IRL agents with the two different policy files. The agents start from 20 different random points and have a goal of visit 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), as we wanted to

compare our theoretically more realistic behavior with an agent capable of reaching scripted route points. We designed routes for these agents placing waypoints on certain featured areas (but not in feature spots); the agents are programmed to walk to those points in order and then evacuate at the final point.

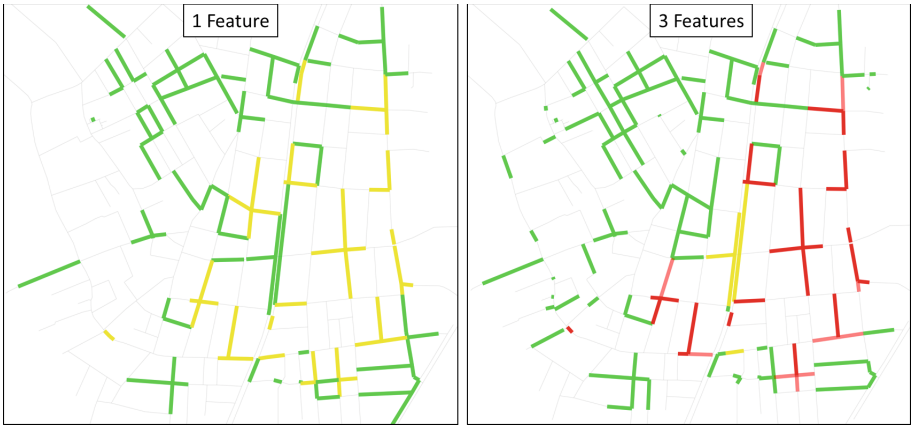

Fig. 4. The rewarded links in the map when training with one and three different types of features. The links are colored depending of their reward value, with no coloring in case of no reward, green if there is a low reward, yellow if it is moderate and red for high rewards. (Color figure online)

All the simulations showed similar results in terms of population in featured areas and number of features visited. In Fig. 5 we can observe the distribution of the pedestrians in the simulation of 6000 agents with only one feature group trained. The pathfinder agents crowd around the waypoints, but since they 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 such 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 not in featured areas. Figure 6 shows a caption of the simulation with the three types of featured areas marked with different colors. IRL agents populate almost all of the featured areas, but as the agents have a much more varied featured-driven behavior, the crowd in some of them is not as clear-cut as the in the first experiment.

We also extracted population density data from our tests with the full features in the map. The IRL agents visited all the 36 featured nodes in the map, with an average of 7.11 featured spots, whilst the pathfinder agents visited an average of 2.23, leaving 9 nodes that were untouched by them. Naturally, the pathfinder agents only went to the featured nodes that were by chance in their path to the goal, but, interestingly this means that not all business are in the most optimal paths and agents that does not take in account behavioral patterns, will ignore places that may be important. Finally, we observed space to

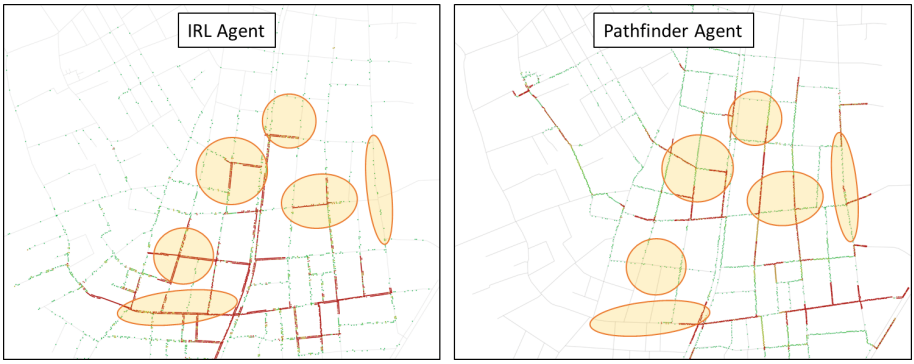

Fig. 5. Comparison between the simulations with one feature trained, both of them with 6000 agents. Featured areas are marked in yellow. (Color figure online)

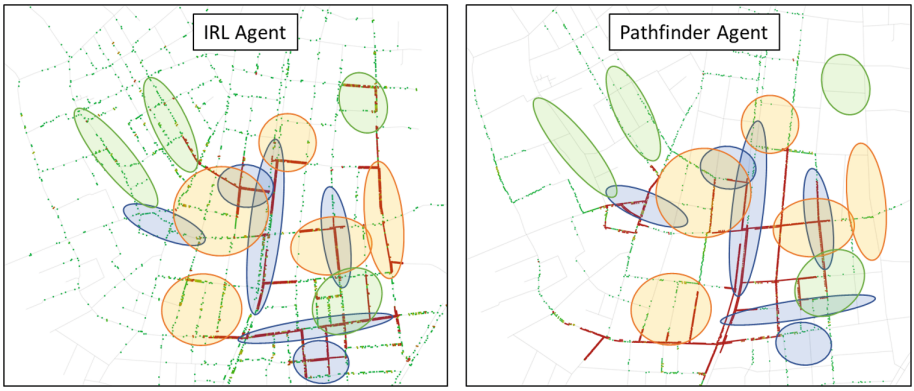

Fig. 6. Comparison between the behavior of the agents used in a simulation with the three groups of features trained, both of them with 6000 agents. The colors used mark different areas: yellow for shops, blue for restaurants, and green for entertainment. (Color figure online)

improvement, identifying a number of issues in the IRL agents. First, we noticed deadlocks emerging when there are a high number of agents, and we plan to solve that problem in the next version of the system; however, the deadlocks appear much more often using pathfinder agents. We will add reactive behavior to the agents in order make them able to detect when they cannot continue their path and even warn other agents of it.

Also, we think we could improve the agents behavior when shifting between behavior patterns, because currently switching behaviors is predefined and we think that is more a way to circumvent the fact that the agents still do not switch policy functions (and even it may not be suitable to real situations). We plan to perform an additional layer of learning to obtain a cognitive model for changing behaviors from the data used for training, and then we can manage

different policies for different behaviors. Once we finish this behavior management method, we will test if its better to have one unique policy and reward functions for multiple trained behaviors, or to maintain one per behavior, shifting the policy depending of the goal.

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. Current pedestrian or crowd simulators rarely use this approach in order to simulate the crowd behavior, preferring to use this method for optimal trajectory generation instead. However, such kind of system would have a wide array of applications in fields: for example it can be used for smart city planning by extracting the movement patterns of pedestrians and applying those patterns to create optimal paths to key areas; it can be used for disaster prevention by identifying which areas are more likely to create bottlenecks in evacuations, and avoid undesirable paths that would be nonetheless taken by fleeing pedestrians; it also can be applied to take business strategy decisions in where to place certain types of business by predicting how they can perform in attracting customers depending of their surrounding features.

We developed a module that provides behavioral learning to the agents in our crowd simulator. This module works using an inverse reinforcement learning technique by converting the domain of pedestrians walking across a city into a Markov decision process. We performed two sets of experiments obtaining promising results 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 better behavior. Also, by observing the reward values of the links on the map we can detect which places are more optimal for certain business. We observed interesting aspects in our tests. First, when the number of agents increase, pedestrian congestion was much more common with pathfinder agents than using the IRL agents, hindering their performance or even generating deadlocks at certain junctions. This is mostly due to the wandering nature of the IRL agents around zones that match their behavior pattern but are not part of optimal paths. We also observed that IRL agents have a consistent behavior independently of the number of simulated agents.

In the next steps of our research, we plan to improve the agents with further machine learning techniques, in order to teach them when to switch between different behavior patterns. We also plan to validate the behavior of our agents by comparing them with live data from real pedestrians. In order to do this and due to difficulties to track effectively massive numbers of people, we will apply our system to concrete environments more manageable, like crowd movement in controlled events or customer behavior inside of supermarkets. 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. Thus, we aim to provide better understanding of crowd simulation using IRL and usable tools for its potential applications.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning, p. 1. ACM (2004)
2. Alger, M.: Deep inverse reinforcement learning (2015)
3. Crociani, L., Lämmel, G., Vizzari, G.: Multi-scale simulation for crowd management: a case study in an urban scenario. In: Osman, N., Sierra, C. (eds.) AAMAS 2016. LNCS (LNAI), vol. 10002, pp. 147–162. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46882-2_9
4. Crociani, L., Vizzari, G., Yanagisawa, D., Nishinari, K., Bandini, S.: Route choice in pedestrian simulation: design and evaluation of a model based on empirical observations. Intell. Artif. **10**(2), 163–182 (2016)
5. Dvijotham, K., Todorov, E.: Inverse optimal control with linearly-solvable MDPs. In: Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 335–342 (2010)
6. Faccin, J., Nunes, I., Bazzan, A.: Understanding the behaviour of learning-based BDI agents in the Braess' paradox. In: Berndt, J.O., Petta, P., Unland, R. (eds.) MATES 2017. LNCS (LNAI), vol. 10413, pp. 187–204. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64798-2_12
7. Helbing, D., Molnar, P.: Social force model for pedestrian dynamics. Phys. Rev. E **51**(5), 4282–4286 (1995)
8. Herman, M., Gindele, T., Wagner, J., Schmitt, F., Quignon, C., Burgard, W.: Learning high-level navigation strategies via inverse reinforcement learning: a comparative analysis. In: Kang, B.H., Bai, Q. (eds.) AI 2016. LNCS (LNAI), vol. 9992, pp. 525–534. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50127-7_45
9. Johansson, A., Helbing, D., Shukla, P.K.: Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. Adv. Complex Syst. **10**(2), 271–288 (2007). https://doi.org/10.1142/S0219525907001355
10. Kohjima, M., Matsubayashi, T., Sawada, H.: What-if prediction via inverse reinforcement learning. In: Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, 22–24 May 2017, pp. 74–79 (2017). https://aaai.org/ocs/index.php/FLAIRS/FLAIRS17/paper/view/15503
11. Lämmel, G., Plaue, M.: Getting out of the way: collision-avoiding pedestrian models compared to the RealWorld. In: Weidmann, U., Kirsch, U., Schreckenberg, M. (eds.) Pedestrian and Evacuation Dynamics 2012, pp. 1275–1289. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-02447-9_105
12. Lämmel, G., Grether, D., Nagel, K.: The representation and implementation of time-dependent inundation in large-scale microscopic evacuation simulations. Transp. Res. Part C Emerg. Technol. **18**(1), 84–98 (2010)
13. Levine, S., Popovic, Z., Koltun, V.: Nonlinear inverse reinforcement learning with Gaussian processes. In: Advances in Neural Information Processing Systems, pp. 19–27 (2011)
14. Luo, L., et al.: Agent-based human behavior modeling for crowd simulation. Comput. Animat. Virtual Worlds **19**(3–4), 271–281 (2008)
15. Martinez-Gil, F., Lozano, M., Fernández, F.: Emergent behaviors and scalability for multi-agent reinforcement learning-based pedestrian models. Simul. Model. Pract. Theory **74**, 117–133 (2017)

16. Ng, A.Y., Russell, S.J., et al.: Algorithms for inverse reinforcement learning. In: ICML, pp. 663–670 (2000)
17. Schadschneider, A., Klingsch, W., Klüpfel, H., Kretz, T., Rogsch, C., Seyfried, A.: Evacuation dynamics: empirical results, modeling and applications. In: Meyers, R. (ed.) Extreme Environmental Events, pp. 517–550. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-7695-6_29
18. de Albuquerque Siebra, C., Botelho Neto, G.P.: Evolving the behavior of autonomous agents in strategic combat scenarios via sarsa reinforcement learning. In: Proceedings of the 2014 Brazilian Symposium on Computer Games and Digital Entertainment, SBGAMES 2014, Washington, DC, USA, pp. 115–122. IEEE Computer Society (2014). https://doi.org/10.1109/SBGAMES.2014.36
19. Šošić, A., KhudaBukhsh, W.R., Zoubir, A.M., Koeppl, H.: Inverse reinforcement learning in swarm systems. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 1413–1421. International Foundation for Autonomous Agents and Multiagent Systems (2017)
20. Svetlik, M., Leonetti, M., Sinapov, J., Shah, R., Walker, N., Stone, P.: Automatic curriculum graph generation for reinforcement learning agents, November 2016. http://eprints.whiterose.ac.uk/108931/
21. Torrens, P.M., Nara, A., Li, X., Zhu, H., Griffin, W.A., Brown, S.B.: An extensible simulation environment and movement metrics for testing walking behavior in agent-based models. Comput. Environ. Urban Syst. **36**(1), 1–17 (2012)
22. Yamashita, T., Soeda, S., Noda, I.: Evacuation planning assist system with network model-based pedestrian simulator. In: Yang, J.-J., Yokoo, M., Ito, T., Jin, Z., Scerri, P. (eds.) PRIMA 2009. LNCS (LNAI), vol. 5925, pp. 649–656. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-11161-7_52
23. Zanlungo, F., Ikeda, T., Kanda, T.: Social force model with explicit collision prediction. EPL (Europhys. Lett.) **93**(6), 68005 (2011)
24. Zhong, J., Cai, W., Luo, L., Zhao, M.: Learning behavior patterns from video for agent-based crowd modeling and simulation. Auton. Agents Multi-Agent Syst. **30**(5), 990–1019 (2016)
25. Ziebart, B.D., Maas, A.L., Bagnell, J.A., Dey, A.K.: Maximum entropy inverse reinforcement learning. In: AAAI, Chicago, IL, USA, vol. 8, pp. 1433–1438 (2008)