Enriching Graph Information for Pedestrian Behavior Learning

Nahum Alvarez, Chenyi Zhuang, Itsuki Noda

[1]National Institute of Advanced Industrial Science and Technology

Abstract: Many aspects in planning and prediction in real environments could benefit from behavior learning techniques, with pedestrian simulation for city and business planning being one of them. We developed a method for behavior learning called Contextual Action Multiple Policy Inverse Reinforcement Learning (CAMP-IRL) that allows effective pedestrian behavior simulation. However we noticed some limitations when identifying important relationships between locations in the city map. In this paper we present a graph enrichment technique devised to solve this issue and similar ones. The technique works by applying a number of transformations to the map features in order to permeate with feature information the nodes that are indirectly related to those features. Our tests showed promising results, improving the performance of the original CAMP-IRL method and opening new paths to explore.

Introduction

Pedestrian behavior simulation is a difficult task to perform due to the performance requirements and the necessary information to learn meaningful behavior patterns. Inverse Reinforcement Learning techniques help in solving those issues, as they learn from a set of observed behaviors provided by an expert and can be processed before the simulation. We developed a variant that includes contextual actions and multiple reward functions and adapted it to work with a multi-agent based pedestrian simulator. The agents in the simulator are able to navigate the map with no information other than the learned behavior patterns and obtain better results than other methods in terms of goal clear times and trajectory optimization. We called them "Contextual Action Multiple Policy Inverse Reinforcement Learning" (CAMP-IRL) agents.

However, we found several instances when only having the data from expert trajectories was not enough to obtain the desired knowledge. For example, under certain conditions, traversing certain areas of unknown layout to reach concrete goals is difficult for trained agents, whilst for humans such information should be trivial to deduct. In order to avoid such situations, we devised a method to improve the available graph information contained in the trajectory database.

This work is organized as follows: Section 2 contains a review of previous work on reinforcement learning used for agent behavior and pedestrian simulators. Section 3 describes the CAMP-IRL method and our pedestrian simulator. Section 4 presents our map enrichment method and Section 5 shows our preliminary results. Finally, section 6 contains the conclusions of our research.

Related Work

Apprenticeship learning methods have been widely used in intelligent agents' systems to train them to perform tasks in dynamic environments [1]. We can observe strategies to emulate predefined driving behaviors [2], and there are also works where agents are given a behavior cognitive model for pedestrians [3]. However, in those works the behavior model is predefined by a designer, having a low degree of flexibility, being tied to its domain, or even escalating badly.

An extra issue in simulating people's behavior is that the reward function governing their actions will be often hidden. We can avoid this problem using inverse reinforcement learning (here on after IRL) because instead a reward function, it only needs a set of observed expert demonstration behaviors. IRL works well on domains where the reward function is hidden, being appropriate to model animal and human behavior [4].

We can find many different approaches to IRL, each one with its own characteristics and issues [5]. For example, we find a linearly solvable approach in [6], with a number of constraints in the MDP definition, or the Maximum Entropy method [7], which works well when we do not have much information about the solution space. Other methods have obtained better results under certain conditions, like [8] which works on a subset of MDPs, but it does not match well with our domain, or [9] which deals

with non-linear reward functions.

Works using IRL to learn agent behavior are sparse but effective, as it is shown in [10] where driving styles are learned by an agent, or [11] where different agents work together for routing traffic.

Other works have previously dealt with extracting multiple reward functions, like [12], showing how to switch between different MDP and obtain their related policy functions and works well extracting different behaviors. The work in [13] divides the data in smaller sub-goals in order to obtain simple reward functions, and in [14] we find a hierarchical method for selecting MDP partitions with different policies for each sub-MDP, which can be interesting for domains where the agent has a number of sequential small sub-goals.

The CAMP-IRL Method

IRL techniques work on domains that can be modeled by a Markov Decision Process (MDP) but have hidden reward functions (the reward function dictates the gain from performing a given action in a given state). However human behavior is not only directed by only one goal but many, with different rewards that are managed at the same time.

We based our method in a non-parametric Bayesian approach to the problem [15] 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, used to avoid an explosion in the solution space by cutting redundant actions and allow flexibility in the domain definition.

Contextual Action Multiple Policy MDP (CAMP-MDP) consists on an MDP {S, $\mathcal{A}$, $\mathcal{T}$, γ , $\mathcal{R}$} with 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. It also composed by a super set $\mathcal{A}$ (s) of actions as a function of a state s, and $\mathcal{R}$ (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). This means that available actions are dependent of the state (i.e., contextual), because each location has a different number of possible paths to take; when translating the locations to states and paths to actions, there will be certain actions only available to certain states. Finally, each state has a set of features, which influence how the reward function is calculated.

The CAMP-IRL algorithm uses a Dirichlet process [16] to classify the trajectories into different groups we call profiles, and then a reward function is calculated for each profile using a Bayesian approach to the IRL method. However, we modified it to be able to work with the CAMP-MDP considering that each state will have a different action set. The algorithm follows the next steps 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:

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

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 random number sampled from a 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, calculated as follows:

$$g(x,y) = \frac{\exp\left(-\frac{1}{2\tau^2}\left\|x - y - \frac{\tau^2}{2}\nabla log P(\chi_k|x)P(x)\right\|\right)}{(2\pi\tau^2)^{D/2}}$$

where τ is a scaling factor.

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. This value represents the expected reward of executing that policy on a node s and it is calculated as follows:

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

In order to use this method for pedestrian prediction, we created a CAMP-IRL module that interfaces with a crowd simulator called CrowdWalk where each pedestrian is represented by an agent.

Our simulator simplifies the city map into a 1-dimensional network consisting of nodes and links. The model of the map consists in a custom xml that describes the map in the form of network where nodes represent intersections and links represent streets or paths, which in our CAMP-MDP will represent as well as states and actions, respectively. The contextual actions are created using the number of links each node has, with one action per link, being semantically different for each node; thus, the first action in certain state will be different from the first action in another one, but will have the same label.

A link also has length and width attributes influencing how long the agents need to walk from an end to another and how many agents can walk in parallel, and can be two-way or one-way. Nodes also can have features, and information describing what facilities are on that location, which are also the state features in the CAMP-MDP.

The inputs of our method are the city map in this model and a file containing the trajectories we want to train the agents with. The CAMP-IRL training algorithm 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 use it in real time. Once the training process finishes, we obtain two files: one containing the weights of the features of each discovered profile, and another containing the value of each map node (as defined in the step 4 of the algorithm) for each profile. The weight and value files will be used in the simulation by the agents created by our CAMP-IRL module agents to traverse the map using the trained behaviors.

In our experiments, the CAMP-IRL agents outperformed other types of agents that do not work with multiple policy functions or contextual actions, but we identified one issue that hindered their behavior. We observed that some useful information that should be extracted from the map and the routes was not being reflected in the learning process; in some of our experiments, one of the goals was a scarce feature that only was present in four nodes of the map; the agents were able to find it, but the wandered excessively before to do it. The main reason was that the system switching between different policies without finding any goal.

After analyzing why this was happening, we found that this situation was due to the coincidence of two factors: scarcity of the goal feature in the map and having only a few and indirect ways to reach those features. In one example, in order to reach one of its goals, agents had to cross from one area of the map to another which could only be reached by crossing three links between them, but those links were not very remarkable in terms of learned value for the selected policies to reach that feature. Thus, agents were conducted by their policies to go towards the feature, but when reaching the nearby areas of the map they could not find the crossing point which was far away. We plan to solve this issue by improving the learning process by adding enriched information to the map, trying to establish semantic relations between nodes of the map like those crossing points and the featured nodes using a method we explain in the next section.

Map Enrichment

Before training the pedestrian behaviors, we first propose a method to improve the available graph information contained in the trajectory database. The intuitive idea we have is that by enriching each graph node information by its neighbors, an agent could make better decisions when choosing the next node to move.

In our experiments, the graph we collected from the trajectory database contains the following information: (1) there are 13 categories (i.e., *hostel, books, convenience, restaurant, café, dry_cleaning, hospital, supermarket, fast_food, kindergarten, telephone, cinema* and *post_office*) describing each node; and (2) the graph structure is stored as an adjacency matrix. Therefore, the input of this method are a feature matrix $X \in \mathbb{R}^{n\times 13}$ recording the category information for all the n nodes

and an adjacency matrix $A \in \mathbb{R}^{n \times n}$ recording the graph structure. The output of this method is a new feature matrix $\hat{X} \in \mathbb{R}^{n \times k}$, where the value of k depends on how many different filters we will used in this method.

Having defined the input and output of this method, in the remainder of this section, we will introduce the method in detail. Similar to signal processing, our method consists on three steps: (1) by using discrete Fourier transform, we first transform the feature matrix X from the graph vertex domain to the graph spectrum domain that is donated as matrix X'; (2) then, we do filtering on X' in the graph spectrum domain; (3) at last, by using inverse discrete Fourier transform, we transform the filtered feature matrix from the spectrum back to the vertex domain.

1. Transform $\mathbf{X}$ in graph vertex domain to $\mathbf{X}'$ in graph spectrum domain:
   To do the graph Fourier transform, we first need to calculate the eigenvectors and eigenvalues of the graph Laplacian matrix $L = D - A$, where A is the adjacency matrix and $D = diag(\sum_{j \neq i} A_{i,j})$ is the degree diagonal matrix. After obtaining the Laplacian matrix L, we obtained its eigen- vectors and values using the following factorization:
   $$L = U \Lambda \mathrm{U}^*$$
   where all the eigenvectors are stored as columns in matrix U, the diagonal matrix Λ records all the eigenvalues and $*$ is the conjugate transpose operator. Then, the graph Fourier transform is defined as:
   $$\mathrm{X}' = \mathrm{U}^* \mathrm{X}$$
   Since each node has 13 categories in our case, by regrading X as a signal having 13 channels, the equation above maps the signal from vertex domain into the spectrum domain, i.e., X'.
2. Do frequency-based filtering on $\mathbf{X}'$:
   Then, we apply different filters to the obtained X'. Without loss of generality, if we define any filter as a function g, the frequency-based filtering process is: $\mathrm{g}(\Lambda)\mathrm{X}'$.
   Similar to signal process, the input of function g are the n eigenvalues stored in Λ, which can be regarded as *graph frequencies*. By defining different filters, we can adjust the final results. Since we do not know which eigenvalues are important for our final pedestrian simulation in advance, we constructed a filter bank to record as many filters as possible. In our experiments, we utilized heat-kernel based filters [17] and Meyer filters [18]. Heat-kernel based filters are low-pass filters (only allowing small eigenvalues to pass the filter) and Meyer filters cover all the frequency ranges, which can allow low-pass, band-pass and high-pass. In our preliminary tests, heat-kernel filters laid better results than Meyers filters, but the performance is highly dependent of the distribution of the features which may vary in other domains of application.
3. Transform X' in graph spectrum domain back to graph vertex domain, i.e. the output $\hat{X}$:
   After filtering, we finally transform the information in spectrum domain back to the vertex domain. Assuming we used two filtering functions $\mathrm{g}_1, \mathrm{g}_2$ in the second step, the final output would be calculated as:
   $$\hat{X} = [\mathrm{U}(\mathrm{g}_1(\Lambda)\mathrm{X}')] \oplus [\mathrm{U}(\mathrm{g}_2(\Lambda)\mathrm{X}')]$$
   where $\oplus$ is the matrix concatenating operator along the second dimension.

Using different filtering functions g would lead to different graph filtering results. To automatically identify which filters are better than others, is advisable to perform a cross validation process where the simulation output feeds the map enrichment method and a final simulation would automatically do the selection.

Once the map enrichment process finishes, it generates a modified map file where the nodes' features are modified and can be passed to our inverse reinforcement learning method to generate finally the behavior patterns and pass them to the CAMP-IRL agents.

Performance comparison

We compared this method of map enrichment by comparing the performance of our CAMP-IRL agents trained using a map with different types of feature filtering. Concretely we used two heat-kernel filters, one with tau = 100, tau = 1000, a Meyer filter with two bands and an addition filter without normalization which consisted on adding to each node the features of its adjacent nodes. By comparing these different filtering strategies, we want to verify whether map filtering would improve the final pedestrian simulation performance.

We tested the speed of the agents in locating 5 goal features in the map. The agents were trained using 150 trajectories belonging to different pedestrian profiles

Table 1: Clear Times of the Agents

Agent	Individual Avg.	Std. Dev.	Avg. Total Clear Time
No Enrichment	8236.84	7871.77	11:48:35
Addition Filter	3338.19	2965.07	5:21:00
Heat Kernel (Tau = 100)	3575.10	3047.51	4:44:31
Heat Kernel (Tau = 1000)	4851.17	5472.58	9:18:44
Meyer Filter	6325.51	6075.81	10:08:53

obtained using synthetic data. All of them were instances of our CAMP-IRL agents, so the only difference in the comparison is the filter applied to the map. The simulations were executed with 150 agents in the same map that was trained, and we performed 10 runs of each simulation in order to average our results.

Table 1 shows the results of our initial experiments, displaying positive results, but mixed when comparing the simple addition method with the use of the filters. We can see that any kind of filtering improves the agents' results in clearing the scenario. As we thought initially, the Meyer filter does not work well with our domain, but we may adapt it for future versions, as its ability to generate different bands for each feature we think it holds potential to be useful. The best filters we found were the addition filter, which is the simplest of them, and the heat kernel filter using a tau value of 100. There are some differences between the two, with the addition filter having the best average clear time for the individual agents, and the heat kernel having faster total clear times for the whole set of 150 agents. This means that the addition filter lowers the time required for agents that are not located in difficult locations where the paths to the goal features is very complex. On the other hand, the kernel filter may benefit more those cases instead. In general, when choosing one filter for the map treatment and in case that it can be only one, the decision will depend on the layout of the map: if there are critical features in spots with no easy access, the heat kernel would be the chosen. In other cases, the addition filter would work fine as a general solution.

However, it also appears that is necessary to perform a cross validation selection for the different filtering functions and their hyper-parameters, as the resulting performance varies greatly depending on it. In our case, Meyer filter did not work well because the features have semantic information for the agents, so dividing each feature into different bands require further training for the agents or they won't be able to select the appropriate band in each situation. In this case, the low pass filter or even simple filters like the addition filter benefit more from the domain characteristics. As a future improvement, we can think in training the model to learn which filter is better, or a switching method in order to allow multi-band filters to work with our domain.

Conclusions

This work presents an graph enriching technique that works with our Contextual Action Multiple Policy Inverse Reinforcement Learning (CAMP-IRL) method, designed to learn pedestrian behavior. This method was devised in order to solve a problem we observed in agent driven simulations with no prior knowledge of the environment layout using machine learning. We found that under certain conditions, it was very difficult for the agents to learn how to get to locations containing features that are very sparse and have only a few ways to get to them if they only use the layout information as it is provided. We proposed that by pre-processing the map before training the agents by adding richer information could solve this problem.

Our method converts a city map into a model where the states represent locations on the map and the actions symbolize movements between locations. Once this model is created, it is enriched using a filter that modifies the features value of the map nodes. Concretely, the method consists on transformation from the graph vertex domain to the graph spectrum domain and then the desired filter is applied. Finally the map is reverted again to the vertex domain.

The technique is flexible enough to allow different types of filters, and it is possible to define new ones that fit better other domains. In fact, we observed that is important to choose an appropriate filter to work with our pedestrian simulation problem, obtaining very different results depending on which one was chosen.

Once obtained the enriched map, the model is trained using the data from previously stored pedestrian trajectories. The products of the training process are a set

of behavior profiles which will be used by the agents to traverse the map, choosing the profile that fits better their goals. The CAMP-IRL agents are also able to switch profiles whenever they have to obtain a different goal or when they consider that their profile is not good enough to reach the current goal.

We prepared a set of experiments in order to compare the performance of different filters. The experiments consisted in running a number of simulations where the agents have to reach 5 goals after being trained with our CAMP-IRL method. In our tests we experimented with different heat kernel, Meyer and addition filters. We observed that in general, enriching the map information improves the agents' performance, being the best ones the addition filter and the heat kernel filter with tau = 100. Also, we found that those two filters have different advantages, with the addition filter obtaining better goal completion times for the pedestrians in general, and the heat kernel being better for pedestrians placed in locations where it is difficult to reach certain goals.

By seeing the results of our experiments, we think it is worth to work further in this method, and testing more different filters. A filter that we want concretely to develop is one able to reflect how much influence has a feature in a node, even if such feature is not actually present in it. We plan to do it by identifying feature-driven relations between nodes and influence areas for the features.

References

[1] Svetlik, M., Leonetti, M., Sinapov, J., Shah, R., Walker, N., and Stone, P. (2016). Automatic curriculum graph generation for reinforcement learning agents.

[2] Faccin, J., Nunes, I., and Bazzan, A. (2017). Understanding the Behaviour of Learning-Based BDI Agents in the Braess' Paradox, pages 187–204. Springer International Publishing.

[3] Martinez-Gil, F., Lozano, M., and Fernandez, F. (2017). Emergent behaviors and scalability for multiagent reinforcement learning-based pedestrian models. Simulation Modelling Practice and Theory, 74:117–133.

[4] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In Icml, pages 663–670.

[5] Zhifei, S. and Meng Joo, E. (2012). A survey of inverse reinforcement learning techniques. International Journal of Intelligent Computing and Cybernetics, 5(3):293–311.

[6] Kohjima, M., Matsubayashi, T., and Sawada, H. What-if prediction via inverse reinforcement learning. In Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Florida, USA, May 22-24, 2017, pages 74–79.

[7] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In AAAI, volume 8, pages 1433–1438. Chicago, IL, USA.

[8] Dvijotham, K. and Todorov, E. (2010). Inverse optimal control with linearly-solvable mdps. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 335–342.

[9] Levine, S., Popovic, Z., and Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes. In Advances in Neural Information Processing Systems, pages 19–27.

[10] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1. ACM.

[11] Natarajan, S., Kunapuli, G., Judah, K., Tadepalli, P., Kersting, K., and Shavlik, J. (2010). Multi-agent inverse reinforcement learning. In 2010 Ninth International Conference on Machine Learning and Applications, pages 395–400. IEEE.

[12] Surana, A. and Srivastava, K. (2014). Bayesian nonparametric inverse reinforcement learning for switched markov decision processes. In Machine Learning and Applications (ICMLA), 2014 13th International Conference on, pages 47–54. IEEE.

[13] Michini, B. and How, J. P. (2012). Bayesian nonparametric inverse reinforcement learning. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 148–163. Springer.

[14] Krishnan, S., Garg, A., Liaw, R., Miller, L., Pokorny, F. T., and Goldberg, K. (2016). Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. arXiv preprint arXiv:1604.06508.

[15] Choi, J. and Kim, K.-E. (2012). Nonparametric bayesian inverse reinforcement learning for multiple reward functions. In Advances in Neural Information Processing Systems, pages 305–313.

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

[17] Nicole Berline, Ezra Getzler, and Michele Vergne. Heat kernels and Dirac operators. Springer Science & Business Media, 2003.

[18] Nora Leonardi and Dimitri Van De Ville. Wavelet frames on graphs defined by fmri functional connectivity. In Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on, 2136–2139.