Graph Neural Networks for Decentralized Path Planning


1 INTRODUCTION

Efficient and collision-free navigation in multi-agent systems is fundamental to advancing mobility, where collision-free paths are generated to lead agents from their origins to designated destinations. Current approaches can be classified as either coupled or decoupled, depending on the structure of the state space that is searched. While coupled approaches are able to ensure the optimality and completeness of the solution, they involve centralized components, and tend to scale poorly with the number of agents [7, 8]. Decoupled approaches, on the other hand, compute trajectories for each agent separately, and re-plan only in case of conflicts [10–12]. This can significantly reduce the computational complexity of the planning task, but generally produces sub-optimal and incomplete solutions.

Current approaches can be classified as either

\[ \mathcal{A}(X_t; S_t) = \sum_{k=0}^{K-1} S_k^T X_t A_k \]  

(1)

where \( \{A_k\}_k \) is a set of \( F \times G \) matrices representing the filter coefficients combining different observations. \( S_k^T X_t = S_k(S_k^{T-1} X_t) \) is computed by means of \( k \) communication exchanges with 1-hop neighbors, and is actually computing a summary of the information located at the \( k \)-hop neighborhood. The operation \( S_k X_t \) represents a linear combination of neighboring values of the signal due to the sparsity pattern of \( S_k \).

Action Policy: A local MLP (weight-sharing) is trained to predict action \( u_t \) taken by robot \( i \), which is computed by a softmax over the probability distribution of motion primitives, includes up, left,
Reach Goal
b: Flowtime increase ($\delta_{FT}$), as a function of the number of robots. For each panel, we vary the number of communication hops ($K \in \{1, 2, 3\}$), where $K = 1$ corresponds to no communication ($\gamma = 0$). The rows represent the number of robots on which each model was trained, and columns represent the number of robots at test time. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0, and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate $\gamma$ was scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to $10^{-5}$.

Results: Figures 2a and 2b show results for the success rate and flowtime increase, respectively, as a function of the number of robots. We train a model for $N \in \{4, 6, 8, 10, 12\}$, and test it on instances of the same robot team size. In each experiment, we vary the number of communication hops ($K \in \{1, 2, 3\}$). Note that for $K = 1$ there is no communication involved. In both figures, we see a drop in performance for larger teams, but this drop is more pronounced for the non-communicative GNN ($K = 1$). There is a small but noticeable improvement as we increase the number of robots at test time. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0, and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate $\gamma$ was scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to $10^{-5}$.

Results: Figures 2a and 2b show results for the success rate and flowtime increase, respectively, as a function of the number of robots. We train a model for $N \in \{4, 6, 8, 10, 12\}$, and test it on instances of the same robot team size. In each experiment, we vary the number of communication hops ($K \in \{1, 2, 3\}$). Note that for $K = 1$ there is no communication involved. In both figures, we see a drop in performance for larger teams, but this drop is more pronounced for the non-communicative GNN ($K = 1$). There is a small but noticeable improvement as we increase the number of robots at test time. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0, and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate $\gamma$ was scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to $10^{-5}$.

Results: Figures 2a and 2b show results for the success rate and flowtime increase, respectively, as a function of the number of robots. We train a model for $N \in \{4, 6, 8, 10, 12\}$, and test it on instances of the same robot team size. In each experiment, we vary the number of communication hops ($K \in \{1, 2, 3\}$). Note that for $K = 1$ there is no communication involved. In both figures, we see a drop in performance for larger teams, but this drop is more pronounced for the non-communicative GNN ($K = 1$). There is a small but noticeable improvement as we increase the number of robots at test time. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0, and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate $\gamma$ was scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to $10^{-5}$.

Results: Figures 2a and 2b show results for the success rate and flowtime increase, respectively, as a function of the number of robots. We train a model for $N \in \{4, 6, 8, 10, 12\}$, and test it on instances of the same robot team size. In each experiment, we vary the number of communication hops ($K \in \{1, 2, 3\}$). Note that for $K = 1$ there is no communication involved. In both figures, we see a drop in performance for larger teams, but this drop is more pronounced for the non-communicative GNN ($K = 1$). There is a small but noticeable improvement as we increase the number of robots at test time. The generalization performance of the network is visualized by a heatmap, which maps performance values into a color range from purple to red, where purple indicates the best performance and red indicates the worst performance.

GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0, and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate $\gamma$ was scheduled to decay from $10^{-3}$ to $10^{-6}$ within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to $10^{-5}$.
REFERENCES


