10-2020

Reinforcement learning for zone based multiagent pathfinding under uncertainty

Jiajing LING

Tarun GUPTA

Akshat KUMAR

Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Reinforcement Learning for Zone Based Multiagent Pathfinding under Uncertainty

Jiajing Ling
School of Information Systems
Singapore Management University
jjling.2018@phdcs.smu.edu.sg

Tarun Gupta
Department of Computer Science
University of Oxford
tarun.gupta@cs.ox.ac.uk

Akshat Kumar
School of Information Systems
Singapore Management University
akshatkumar@smu.edu.sg

Abstract

We address the problem of multiple agents finding their paths from respective sources to destination nodes in a graph (also called MAPF). Most existing approaches assume that all agents move at fixed speed, and that a single node accommodates only a single agent. Motivated by the emerging applications of autonomous vehicles such as drone traffic management, we present *zone-based path finding* (or *ZBPF*) where agents move among *zones*, and agents' movements require uncertain travel time. Furthermore, each zone can accommodate multiple agents (as per its capacity). We also develop a simulator for ZBPF which provides a clean interface from the simulation environment to learning algorithms. We develop a novel formulation of the ZBPF problem using difference-of-convex functions (DC) programming. The resulting approach can be used for policy learning using samples from the simulator. We also present a multiagent credit assignment scheme that helps our learning approach converge faster. Empirical results in a number of 2D and 3D instances show that our approach can effectively minimize congestion in zones, while ensuring agents reach their final destinations.

1 Introduction

The problem of multiagent path finding (MAPF) entails multiple agents finding collision free paths in a collaborative fashion from their respective sources to destination nodes in a graph (Felner et al. 2017; Ma et al. 2017). The MAPF problem has several practical applications such as for game character movements (Ma et al. 2017), warehouse logistics (Wurman, D'Andrea, and Mountz 2008), robotics (Yu and Lavalle 2013), and vessel routing in maritime traffic (Teng, Lau, and Kumar 2017). There are multiple objectives possible in MAPF including minimizing sum of arrival time of all the agents at their destination nodes, or the makespan, which is the arrival time of the agent that reaches its destination last. Both these variants are known to be NP-hard (Yu and Lavalle 2013).

Various solution approaches (both approximate and optimal) have been developed for MAPF. Reduction based approaches translate MAPF to other well known problems such as SAT (Surynek 2017; Surynek, Kumar, and Koenig 2019), integer programming (Yu and Lavalle 2013), and constraint satisfaction (Wang et al. 2019) among others. There are several approaches that take a search based perspective of MAPF and develop multiagent search algorithms such as conflict based search (Sharon et al. 2011; 2015) and its variant (Boyarski et al. 2015). Reinforcement learning based approaches are also developed for MAPF (Sartoretti et al. 2019).

In the standard MAPF formulation, a key assumption is that each graph node can accommodate at most one agent at a time, and that agents move at fixed uniform speed. However, conflict free paths in such idealized models may not be translated to real world scenarios where there is significant uncertainty (e.g., in agent movements), partial observability and decentralized control (agents can only observe their local neighborhood), and graph nodes having capacities more than one (Ma et al. 2017; Barták, Švancara, and Vlk 2018; Surynek, Kumar, and Koenig 2019). Several recent attempts aim to generalize standard MAPF to realistic settings. Based on resource constrained activity scheduling, the MAPF problem has been extended to incorporate nodes/edges with higher capacity more than one agent, and non-uniform travel time for different edges (Barták, Švancara, and Vlk 2018; Surynek, Kumar, and Koenig 2019). However, this problem setting is still deterministic as there is no uncertainty in the movement of agents.

To handle movement uncertainty and imperfect plan execution by agents, delay probabilities are introduced in (Ma, Kumar, and Koenig 2017) in which an agent's movement to the next node succeeds with probability p, and with $1 - p$ probability agent stays at its current node. Such agent movement essentially follows a geometric distribution, which may not be general enough for several practical problems. Their solution strategy relies on modifying the plan execution during run time to avoid collisions, rather than using a full multiagent planning model such as a decentralized partially observable MDP (Dec-POMDP) (Bernstein et al. 2002). Another limitation is that their approach considers capacity of each node as one agent. Another robust plan execution strategy is presented in (Honig et al. 2016). Their approach post-processes the output of a MAPF solver to create a schedule that can be executed safely (i.e., collission-

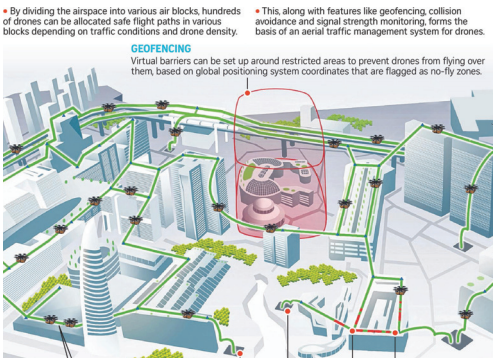
free) by (robotic) agents. Their approach considers minimum and maximum time needed by an agent to move along a graph edge, and ensures that the post-processed output is safe for all agent movements within the allowed range. Their approach also considers nodes having the capacity of one agent. A velocity planner is developed in (Singh et al. 2019). Their approach addresses uncertainty in movement, but is limited to optimizing only the temporal aspect of movement (not the spatial movement of agents), and assumes that all the agents are homogeneous. To summarize, there are generalizations of MAPF that consider graph nodes and edges having capacity of more than one agent, but may not model agent uncertainty, and there are other generalizations that model movement uncertainty, but assume nodes and edges have a capacity of one agent.

Our contributions: We make the following contributions. *First*, our approach addresses the above limitations of previous MAPF models by allowing for nodes (or *zones*) with capacity of more than one agent, modeling uncertainty (by using the highly expressive class of exponential family distributions to model variable duration movements of agents), and partial observability wherein agents take decisions based on their local observations. We model such *zone based path finding* (ZBPF) under uncertainty using a variant of the Dec-POMDP framework. Our model is also able to exploit the symmetry present in a large multiagent system where most agents are identical to each other, and influence each other via their collective state (e.g., congestion). Exploiting such collective nature of interactions increases the scalability of solution approaches.

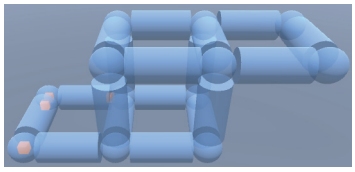
Second, we present a novel difference-of-convex functions (DC) programming (Lipp and Boyd 2016) based formulation of policy optimization. The DC perspective results in a sequence of optimization problems, each of which is easier to solve than the original policy optimization problem using samples from the domain simulator. To improve the convergence of the learning approach, we also develop value function factorization techniques that perform multiagent credit assignment for faster convergence (Chang, Ho, and Kaelbling 2004; Foerster et al. 2018).

Third, to validate our approach, we also develop a simulator for ZBPF in the *ml-agents* platform of the *Unity3D* game engine, which provides a clean interface from the simulation environment to learning algorithms (Juliani et al. 2018). Our simulator is highly expressive which can represent both 2D and 3D maps. Using our simulator, we test on several instances with varying map size and number of agents, and show that our DC programming based learning approach with credit assignment works much better than baseline approaches based on multiagent Q-learning (Fu et al. 2019).

Motivating applications: The ZBPF framework is motivated by the emerging applications of autonomous-vehicle fleet management, such as drone traffic control. Figure 1(a) shows the airspace of a city divided into multiple geofenced airblocks (which are analogous to zones). Such structured airspace can be used by hundreds of drones to safely travel to their destinations (Hio 2016). To avoid congestion, a traffic management system based on ZBPF can be used. Un-



(a)



(b)

Figure 1: (a) Airspace management for drone traffic (Hio 2016); (b) Snapshot from our simulator (best viewed electronically)

like a graph like discrete data structure, airspace is continuous. Therefore, dividing the airspace into zones which can accommodate multiple drones is necessary. Furthermore, as zones may have different lengths in different locations, agents may often take variable and uncertain amount of time to navigate them. Figure 1(b) shows conceptual representation of a zone-based network in our simulator. Connected blue cylinders/spheres are zones, and orange nodes are agents.

2 Zone Based Path Finding

A standard MAPF formulation is as follows (Sharon et al. 2015). We are given a graph $G=(V,E)$ where the set V denotes the locations where agents can move, and edges connect different locations. We have $i=1:N$ agents; an agent i has a start vertex s_i and final goal vertex g_i. A path for an agent i is a sequence of vertices where starting vertex is s_i, final vertex is g_i. More concretely, a function l_i describes agent i's path by mapping the time $(t=0:H)$ to a location; $l_i(0)=s_i$, $l_i(H)=g_i$, $l_i(t)\in V$ such that $l_i(t)$ and $l_i(t+1)$ are either identical (which means the agent stays at the same location), or $l_i(t+1)$ is the neighboring location to $l_i(t)$ connected by an edge $e\in E$ where agent must move to at time $t+1$. A solution for MAPF consists of paths for each agent such that there are no collisions (no two agents at the same vertex at the same time). Some versions of MAPF also consider edges having capacity one (or simultaneous swap of agents on a single edge is not allowed). In the standard MAPF, all the agents move at fixed uniform speed, and the problem setting is deterministic. There are multiple optimization criterion possible. In the makespan objective, the

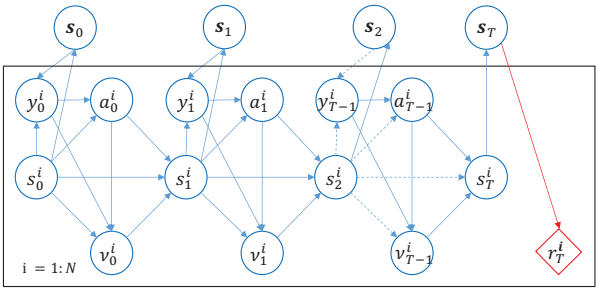


Figure 2: Dynamic Bayes Net (DBN) for T-step reward

total time until the last agent reaches its goal is minimized. In the sum-of-cost (SOC) objective, we minimize the sum of arrival time of all the agents at their goal zones. We next introduce uncertainty in MAPF and higher capacity nodes.

Zone based path finding model: We next present a model for ZBPF by using a variant of the Dec-POMDP model (Bernstein et al. 2002), which is a popular framework for multiagent sequential decision making under uncertainty and partial observability. The set of all zones z is Z. Given a directed graph $G = (V, E)$, each node is a zone, and edges represent connections between neighboring zones. We assume that each zone has a capacity C_z, which denotes the maximum number of agents a zone can accommodate at a time. If more agents are present in a zone than its capacity, it creates congestion. Crossing a zone z to a neighboring zone z' requires a minimum and maximum travel time—$t_{\min}, t_{\max}$. Our approach can also handle the case when each agent has a unique $t_{\min}, t_{\max}$ for each zone intersection, but for simplicity, we assume all zone movements have the same lower/upper limits. We also assume time is discretized, and horizon is H.

Agents' decision model: We now describe how an agent's decision making evolves in ZBPF. Instead of describing a specific formulation applied only to ZBPF, we develop a more general decision theoretic model (in figure 2) of which ZBPF is a specific instance. Figure 2 describes graphically how different agents interact with each other using a dynamic Bayes net (and plate notation). Different arrows describe how the state transition/observation/reward functions of an agent depend on global/local state, and previous action; detailed descriptions of these parameters follow next. Similar to MAPF, an agent i has a starting zone z_o^i, and a goal zone z_g^i (multiple agents are allowed to share their start and goal zones).

States: We model the behavior of each agent as follows. Let s_t^i denote the state of agent i at time t. Consider an agent i currently inside a zone z. As the navigation between zones has a variable duration (as described later), this agent can be categorized as *newly arrived* at some zone z at time t or *in-transit* through z at time t.

- [In-Transit] The state of an agent i is defined as a tuple $s_t^i = \langle z, z', \tau \rangle$, where z denotes the current zone of agent i at time t; z' denotes the next zone agent is heading to; and τ is the remaining time to reach z'.
- [Newly Arrived] When an agent i newly arrives at zone z, its next zone z' and the time-to-reach z' (say τ) are not yet determined (which will be determined by the agent's action sampled from the policy). The state s_t^i is denoted as $\langle z, \Phi, \Phi \rangle$.

Actions: The action set of an agent is $A = \{\langle z, \nu \rangle \ \forall z \in V, \nu \in \Re^d\} \cup \{\text{noop}\}$. In a given state s_t^i, only some of the actions are valid.

- If agent is in-transit, $s_t^i = \langle z, z', \tau \rangle$, only valid action is noop.
- If agent is newly arrived at a non-goal zone $z \in Z \setminus \{z_g^i\}$, or $s_t^i = \langle z, \Phi, \Phi \rangle$, then valid action set is $A = \{\langle z, \nu \rangle \ \forall z \in \text{Nb}(z), \nu \in \Re^d\}$, where $\text{Nb}(z)$ is the set of neighbor zones of z where agent can move to. There are additional conditions on possible parameter ν as explained in the transition function.
- If agent is at the start of a goal zone, or $s_t^i = \langle z_g^i, \Phi, \Phi \rangle$, then the only valid action is noop.

Intuitively, for an agent at the start of a non-goal zone z, two action components are needed—$\langle z', \nu \rangle$. First, the agent decides its next destination $z' \in \text{Nb}(z)$ to move to. Second, the agent must decide the speed at which it wants to move. To avoid modeling the control dynamics of agents, we assume that the agent decides the time to take to move to z'. This temporal movement part is controlled by the continuous action component ν (transition function shows a more precise definition). Thus, the model we present contains both discrete and continuous actions, which represents a more challenging setting than standard planning or RL.

Transition function: We assume that the movement uncertainty is captured by the transition function. Instead of assuming a specific movement model such as geometric distribution (Ma, Kumar, and Koenig 2017), we chose to model travel time uncertainty using a widely applicable yet tractable class of exponential family distributions. Give parameter(s) η, the exponential family of probability distributions is defined as those distributions whose pdf or pmf have the following general form (Bishop 2006, Chapter 2):

$$p(x|\eta) = h(x) \exp\{\eta^T T(x) - \mathcal{A}(\eta)\} \quad (1)$$

The function $T(x)$ is the *sufficient statistic*, η is the *canonical parameter*, and $\mathcal{A}(\eta)$ is the partition function (to make sure probabilities normalize to 1). Using this form, we now frame the transition function of the model in figure 2 as:

$$p(s'^i|s^i, a^i, \nu^i) = f(s^i, a^i, s'^i) \exp\{\nu^i \phi(s^i, s'^i) - \mathcal{A}(\nu^i)\} \quad (2)$$

where f is non-negative (plays the role of $h(x)$); a^i is the discrete action (analogous to the direction decision in ZBPF), and ν^i is the canonical parameter of the exponential family distribution that governs the transition function of each agent i.

The above representation of the transition function is quite general; it can represent standard categorical distribution (which are widely used as transition function for MDPs), as

well as distributions such as Gaussian. The exponential family distributions are also useful to model travel times because they are the maximum-entropy distribution consistent with given constraints (or prior information) about the underlying random variables. As per the principle of maximum entropy, among the distributions that satisfy given constraints, the least informative one must be chosen (Jaynes 1957). E.g., given that we have fixed time limits $t_{\min}$ and $t_{\max}$ on the movement of an agent between zones, an agent may decide to move in α time steps. The maximum entropy discrete distribution with a given mean (say α) and bounded support (between $t_{\min}$ and $t_{\max}$) is the binomial distribution (Harremoes 2001), which models the uncertainty in agent movement. In this case, we interpret ν as the success probability of the binomial distribution. We can similarly interpret the geometric distribution used by (Ma, Kumar, and Koenig 2017). In their approach, an agent's move action succeeds with (a given) probability p. In other words, the average time to cross an edge is $\alpha = 1/p$. The maximum entropy distribution with a given mean is indeed the geometric distribution[1]. Similarly, if time is continuous, then the agent's action ν can be interpreted as the mean of the Gaussian distribution which models the travel duration (we can assume that the agent cannot control the variance, which is known).

To summarize, given an action $\langle z', \nu \rangle$, the first action component denotes the *direction decision* (the zone z') where agent wants to go next, and ν is the *canonical parameter* of the transition function (assuming the transition function is from the exponential family). Specific transition function for ZBPF is given as:

- For a newly arrived agent i, let $s_t^i = \langle z, \Phi, \Phi \rangle$, action taken $\langle z', \nu \rangle$ (which implies agent has decided to go to zone z'), then the time taken to move to z' is sampled from the distribution $\tau \sim p^i_{\text{dur}}(\cdot; \nu, z')$, where p^i_{dur} is an exponential family distribution with finite support between $t_{\min}$ and $t_{\max}$ (that is, $\tau \in \{t_{\min}, \ldots, t_{\max}\}$). If $\tau = 1$ (i.e., only one time step to-go), then the next state is $\langle z', \Phi, \Phi \rangle$ (or agent reaches the next zone z' at time $t+1$), therefore, $p^i(\langle z', \Phi, \Phi \rangle | \langle z, \Phi, \Phi \rangle, \langle z', \nu \rangle) = p^i_{\text{dur}}(\tau; \nu, z')$. If $\tau > 1$, the next state is $\langle z, z', \tau - 1 \rangle$, and $p^i(\langle z, z', \tau - 1 \rangle | \langle z, \Phi, \Phi \rangle, \langle z', \nu \rangle) = p^i_{\text{dur}}(\tau; \nu, z')$.
- For in-transit agent i, let $s_t^i = \langle z, z', \tau \rangle$, then the only action allowed is noop. If $\tau = 1$, then the state deterministically transitions to $\langle z', \Phi, \Phi \rangle$. If $\tau > 1$, then the state deterministically transitions to $\langle z, z', \tau - 1 \rangle$. That is, the agent has not yet completed its movement action, and is still in zone z en-route to z'.

We treat an agent's goal zone as an absorbing state without any outgoing transitions. For simplicity, we assume well-formed infrastructures where an agent occupying its goal zone does not obstruct other agents from finding their paths (Cap et al. 2015).

Observation function: Each agent i observes y_t^i based on its local state s_t^i, and the global state of all agents $\boldsymbol{s}_t$. For simplicity, we assume a deterministic observation function, which can include observation about all agents present in the same zone as agent i and in neighboring zones.

Reward function: There is a penalty of $w_d < 0$ given to each agent for every time step the agent is not at its goal location. When the agent reaches its goal zone for the first time, a positive reward r is given. In a global state $\boldsymbol{s}$, if total number of agents in a zone z are more than its capacity C_z, then each agent in zone z receives a congestion penalty of $w_c < 0$. Total system reward r is sum of penalties and rewards for all the agents (or $r_t = \sum_{i=1}^{N} r_t^i$). Notice that, when there is movement uncertainty, finding a path for agents that guarantees no collisions would lead to very conservative and high cost solutions. Therefore, unlike MAPF, we use a soft penalty to encourage coordination among agents and avoid congestion.

Policy representation: An agent i has two policies. First, the policy $\pi^i(a_t^i | s_t^i, y_t^i)$ provides a distribution over discrete actions a (discrete actions are analogous to the direction decisions in ZBPF). Thus π^i is a stochastic policy. We also have a *deterministic* policy μ^i which provides the (often continuous) action component $\nu_t^i = \mu^i(s_t^i, y_t^i, a_t^i)$. These policies can also be parameterized using parameters θ.

3 Objective Function and DC Programming

We first go over the difference-of-convex functions (DC) programming framework (Yuille and Rangarajan ; Lipp and Boyd 2016). Our goal would be to reformulate the objective function as an instance of DC programming. The DC programming and *concave-convex procedure* (CCCP) (Yuille and Rangarajan) are a popular approach to optimize a general non-convex function expressed as a *difference* of two convex functions. We describe it here briefly. Consider the optimization problem:

$$\min\{g(x) : x \in \Omega\} \tag{3}$$

where $g(x) = u(x) - v(x)$ is an arbitrary function with u , v being real-valued *convex* functions and Ω being a convex set. The CCCP method provides an iterative procedure that generates a sequence of points x_k by solving the following convex program:

$$x_{k+1} = \arg\min\{u(x) - x^T \nabla v(x_k) : x \in \Omega\} \tag{4}$$

Each iteration of CCCP decreases the objective $g(x)$ and is guaranteed to converge to a local optimum (Lipp and Boyd 2016). The above formulation is derived by linearizing the convex function v at the point x_k as: $\hat{v}(x) = v(x_k) + \nabla v(x_k)^T (x - x_k)$, and exploiting the property that linearizing a convex function results in a global (valid over the entire domain of function) lower bound on v. The key benefit of CCCP is that problem (4) is often much easier to solve than the original problem (3) as the objective in (4) is convex (first term is convex, and second term is linear in x).

We now show how to exploit the DC for our ZBPF problem. Consider the DBN in figure 2. A joint trajectory is denoted as $\boldsymbol{\zeta} = \boldsymbol{s}_0, \boldsymbol{y}_0, (\boldsymbol{a}_0, \boldsymbol{\nu}_0), \boldsymbol{r}_0, \boldsymbol{s}_1, \boldsymbol{y}_1, (\boldsymbol{a}_1, \boldsymbol{\nu}_1), \boldsymbol{r}_1, \ldots$ where subscripts denote time. For deriving the DC formulation, we start by considering policy π^i for each agent is tabular, and μ^i is a deterministic function. Let π and μ denote

[1] https://en.wikipedia.org/wiki/Exponential_family

the joint-policy. We also assume deterministic observations (as highlighted earlier). The objective to optimize is:

$$J(\pi,\mu) = \sum_{T=0}^{\infty}\sum_{\boldsymbol{\zeta}_{0:T}} \gamma^T r(\boldsymbol{s}_T) b_0(\boldsymbol{s}_0)\pi(\boldsymbol{a}_0|\boldsymbol{s}_0,\boldsymbol{y}_0)$$
$$\prod_{t=1}^{T}\Big(p(\boldsymbol{s}_t|\boldsymbol{s}_{t-1},\boldsymbol{a}_{t-1},\mu(\boldsymbol{s}_{t-1},\boldsymbol{y}_{t-1},\boldsymbol{a}_{t-1}))\pi(\boldsymbol{a}_t|\boldsymbol{s}_t,\boldsymbol{y}_t)\Big) \quad (5)$$

where we assumed that the reward is discounted using γ, b_0 denotes the initial state distribution. W.l.o.g, we also assume rewards are non-negative (if they are not, we can use $r - r_{\min}$ instead of r). Exploiting the DBN structure in figure 2, we can factorize the joint transition function and reward function to get $J(\pi,\mu)$ as:

$$= \sum_{T=0}^{\infty}\sum_{\boldsymbol{\zeta}_{0:T}} \gamma^T r(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{i=1}^{N}\pi^i(a_0^i|s_0^i,y_0^i)\Big)$$
$$\prod_{t=1}^{T}\Bigg(\Big(\prod_{i=1}^{N} p^i(s_t^i|s_{t-1}^i,a_{t-1}^i,\mu^i(s_{t-1}^i,y_{t-1}^i,a_{t-1}^i))\Big)\Big(\prod_{i=1}^{N}\pi^i(a_t^i|s_t^i,y_t^i)\Big)\Bigg) \quad (6)$$

The equation (6) is our objective to maximize by optimizing joint-policies (π,μ). The above objective is non-convex. It is also not expressed in the DC form. To reformulate the above objective into a DC form, we first make a number of variable substitutions as below.

First, we substitute $\pi^i(a^i|s^i,y^i)$ with $\exp\big(\lambda^i(a^i,s^i,y^i)\big)$ in $J(\pi,\mu)$. This can always be done without affecting the optimality. Similarly, we use the the exponential family structure of the transition function (2) to get $J(\lambda,\mu)$ as:

$$= \sum_{T=0}^{\infty}\sum_{\boldsymbol{\zeta}_{0:T}} \gamma^T r(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{t=1}^{T}\prod_{i=1}^{N} f(s_{t-1}^i,a_{t-1}^i,s_t^i)\Big)$$
$$\exp\Bigg[\sum_{i=1}^{N}\Bigg(\Big(\sum_{t=0}^{T}\lambda^i(a_t^i,s_t^i,y_t^i)\Big)+\sum_{t=1}^{T}\Big(\mu^i(s_{t-1}^i,y_{t-1}^i,a_{t-1}^i)$$
$$\times\phi(s_{t-1}^i,s_t^i) - \mathcal{A}(\mu^i(s_{t-1}^i,y_{t-1}^i,a_{t-1}^i))\Big)\Bigg)\Bigg] \quad (7)$$

Above expression has non-negative combination of $\exp(\cdot)$ functions (as all rewards, initial beliefs, and functions f are non-negative). The exp function is convex in λ, but *not* convex in μ. Reason is that for exponential family distributions, the partition function $\mathcal{A}(\nu^i = \mu^i(\cdot))$ is a convex function of canonical parameters ν. However, the exponential function has the term $-\mathcal{A}(\nu^i)$, which is concave in ν^i. To circumvent this problem, we use another substitution as $\xi^i(s^i,y^i,a^i) = \mathcal{A}(\mu^i(s^i,y^i,a^i))$. We will also include this constraint as part of our constraint set Ω when writing the final DC formulation. Using these substitutions, our final expression for $J(\lambda,\mu,\xi)$ is:

$$= \sum_{T=0}^{\infty}\sum_{\boldsymbol{\zeta}_{0:T}} \gamma^T r(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{t=1}^{T}\prod_{i=1}^{N} f(s_{t-1}^i,a_{t-1}^i,s_t^i)\Big)$$
$$\exp\Bigg[\sum_{i=1}^{N}\Bigg(\Big(\sum_{t=0}^{T}\lambda^i(a_t^i,s_t^i,y_t^i)\Big)+\sum_{t=1}^{T}\Big(\mu^i(s_{t-1}^i,y_{t-1}^i,a_{t-1}^i)$$
$$\times\phi(s_{t-1}^i,s_t^i) - \xi^i(s_{t-1}^i,y_{t-1}^i,a_{t-1}^i)\Big)\Bigg)\Bigg] \quad (8)$$

Now each exp term in the above expression is convex in each of λ,μ,ξ, and non-negative combination of convex functions is also convex. We can now reformulation policy optimization in the DC programming framework as:

$$\min_{\lambda,\mu,\xi} 0 - J(\lambda,\mu,\xi) \quad (9)$$
$$\sum_{a^i} e^{\lambda^i(a^i,s^i,y^i)} = 1 \;\; \forall s^i,y^i,\forall i{=}1:N \quad (10)$$
$$\xi^i(s^i,y^i,a^i) = A(\mu^i(s^i,y^i,a^i)) \;\; \forall s^i,y^i,a^i,\forall i{=}1:N \quad (11)$$

The objective (9) is a difference of two convex function, and is therefore a DC function. Notice that the constraints (10) and (11) are non-convex. However, the DC iteration is still applicable because as highlighted earlier, the DC objective uses linearization of the function v (which is $J(\lambda,\mu,\xi)$ in our case), and linearization based bounds are valid over the entire domain of the function.

DC Iteration for ZBPF: The optimization problem (9) is no easier to solve than the original problem (6). However, as (9) is a DC program, we can solve it iteratively analogous to (4). Let the current estimate of parameters be (λ_k,μ_k,ξ_k). To find the improved estimates (λ,μ,ξ), we need to solve the below problem:

$$\max_{\lambda,\mu,\xi} \sum_{i,a^i,s^i,y^i} \nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\lambda^i(a^i,s^i,y^i)$$
$$+ \sum_{i,a^i,s^i,y^i} \nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\mu^i(a^i,s^i,y^i)$$
$$+ \sum_{i,a^i,s^i,y^i} \nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\xi^i(a^i,s^i,y^i) \quad (12)$$
$$\text{subject to constraints (10) and (11) on variables } \lambda,\mu,\xi$$

We can further simplify the above optimization problem by eliminating constraints as follows. We can replace back $\lambda^i(a^i,s^i,y^i) = \ln\pi^i(a^i|s^i,y^i)$, and substitute back $\xi^i(a^i,s^i,y^i) = \mathcal{A}(\mu^i(a^i,s^i,y^i))$ to get the following optimization problem:

$$\max_{\pi,\mu} \sum_{i,a^i,s^i,y^i} \nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\ln\pi^i(a^i|s^i,y^i)$$
$$+ \sum_{i,a^i,s^i,y^i} \nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\mu^i(a^i,s^i,y^i)$$
$$+ \sum_{i,a^i,s^i,y^i} \nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k,\mu_k,\xi_k)\mathcal{A}(\mu^i(a^i,s^i,y^i)) \quad (13)$$
$$\sum_{a^i}\pi^i(a^i|s^i,y^i) = 1; \pi^i(a^i|s^i,y^i) \geq 0 \;\; \forall s^i,y^i,\forall i \quad (14)$$

Problem (13) is much easier to solve than original problem (6). This is because the most computationally intensive step is to compute different gradients ∇J, which are with respect to old parameters (λ_k,μ_k,ξ_k), and we show that a closed form expression can be derived for them. An agent i's state, observation and action are denoted using (s^i,y^i,a^i), and (s^{-i},y^{-i},a^{-i}) denote the same for all other agents except i. Let $d^k(s^i,y^i,a^i,s^{-i},y^{-i},a^{-i})$ denote the occupancy measure or the total discounted amount of time the system was in the joint setting $(s^i,y^i,a^i,s^{-i},y^{-i},a^{-i})$ (note that

action $\boldsymbol{\nu}$ is deterministic given the joint-state, therefore not included in d^k). Expression for measure $d^k(\cdot)$ is as:

$$\sum_{t=0}^{\infty}\gamma^t P(s_t^i = s^i, s_t^{-i} = s^{-i}, y_t^i = y^i, y_t^{-i} = y^{-i}, a_t^i = a^i, a_t^{-i} = a^{-i})$$

We also denote the total discounted reward from a given joint state, action $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$ as $Q^k(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$. Both d^k and Q^k are computed as per the old policy π_k and μ_k. Using d^k, Q^k, different gradients are as (proof in the supplementary):

$$\nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) =$$
$$\sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}) Q^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}, \nu^i, \nu^{-i})$$
$$\nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) = \gamma \sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i})$$
$$\times \sum_{\boldsymbol{s}',\boldsymbol{y}',\boldsymbol{a}'} \phi(s^i, s'^i) P(\boldsymbol{s}', \boldsymbol{y}', \boldsymbol{a}' | s^i, s^{-i}, a^i, a^{-i}, \nu^i, \nu^{-i}) Q^k(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}')$$
$$\nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) = -\sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i})$$
$$\times \Big(Q^k(s^i, s^{-i}, a^i, a^{-i}, \nu^i, \nu^{-i}) - r(s^i, s^{-i}) \Big)$$

We can substitute back the gradients to our DC iteration (13), and get the following optimization problem:

$$\max_{\pi,\mu} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}} \Big[Q^k(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu}) \Big(\sum_{i=1}^{N} \log \pi^i(a^i | s^i, y^i) \Big) \Big]$$
$$+\gamma \mathbb{E}_{\substack{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu},\\ \boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}'}} \Bigg[Q^k(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}') \Bigg(\Big(\sum_{i=1}^{N} \phi(s^i, s'^i) \mu^i(a^i, s^i, y^i) \Big)$$
$$- \Big(\sum_{i=1}^{N} \mathcal{A}(\mu^i(a^i, s^i, y^i)) \Big) \Bigg) \Bigg] \tag{15}$$
$$\sum_{a^i} \pi^i(a^i | s^i, y^i) = 1; \pi^i(a^i | s^i, y^i) \geq 0 \;\; \forall s^i, y^i, \forall i \tag{16}$$

In the above problem, we have introduced only policy normalization constraints. There may be additional constraints on μ variables for the particular exponential family distribution used. The expectation over variables $(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}')$ is as per the joint transition probability of states and actions $P(\cdot | \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$. The expectation over variables $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$ is as per the corresponding occupancy measure d^k.

Planning: The optimization (15) is solvable when the policy of each agent has a convenient form (such as a tabular policy), and transition/observation function are known. Using model parameters and the old policy π_k, μ_k we can get a better policy π_{k+1}, μ_{k+1}. Note that, it is not required to solve problem (15) exactly. E.g., we can use general purpose nonlinear program solver such as SNOPT (Gill, Murray, and Saunders 2005) to approximately optimize it. Such a process is also guaranteed to monotonically increase the original objective J until convergence to a local optima. In situations, when the exact planning model is not available, or state-space is too large, we next present a gradient based approach to approximately optimize (15).

Learning: The form of problem (15) is also highly convenient for reinforcement learning. We assume that policy π_θ and μ_θ are parameterized using θ. We can compute the gradient of the DC iteration objective J' (15) as:

$$\nabla_\theta J' = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}} \Big[Q^k(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu}) \Big(\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta^i(a^i | s^i, y^i) \Big) \Big]$$
$$+\gamma \mathbb{E}_{\substack{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu},\\ \boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}'}} \Bigg[Q^k(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}') \Bigg(\Big(\sum_{i=1}^{N} \phi(s^i, s'^i) \nabla_\theta \mu_\theta^i(a^i, s^i, y^i) \Big)$$
$$- \Big(\sum_{i=1}^{N} \nabla_\theta \mathcal{A}(\mu_\theta^i(a^i, s^i, y^i)) \Big) \Bigg) \Bigg] \tag{17}$$

We can use properties of exponential families to further simplify gradient $\nabla_\theta \mathcal{A}(\nu^i = \mu_\theta^i(a^i, s^i, y^i))$ as $\nabla_\nu \mathcal{A}(\nu^i) \nabla_\theta \nu^i$. The derivative $\nabla_\nu \mathcal{A}(\nu^i)$ is nothing but the expectation of the sufficient statistic ϕ in the transition function expression (2)[2], which can also be estimated using sampling if a closed form expression is not known. The equation (17) provides a basis for computing gradients using sampling to approximately optimize (15). Once a new estimate θ^{k+1} is known, we can iteratively improve it again until convergence.

Multiagent credit assignment: Performing vanilla gradient ascent based on (17) generally suffers from poor convergence. The main reason is that in the presence of multiple agents, gradient estimates have high variance, and the contribution of an agent's action on the global reward is not clear as multiple agents' decisions affect the team reward. This issue is known as the problem of multiagent credit assignment (Chang, Ho, and Kaelbling 2004; Foerster et al. 2018). We next show how to extract a clearer signal from empirical returns that performs effective credit assignment. We assume that the joint Q-function (for a fixed policy) is approximated as follows:

$$Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu}) \approx \sum_{i=1}^{N} h_w^i(s^i, a^i, \nu^i, y^i(\boldsymbol{s})) \tag{18}$$

We focus on an agent i's contribution to the gradient expression (17). Considering first terms depending on $\nabla_\theta \pi^i$, using Q function approximation (18), we get:

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}} \Big[Q^k(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu}) \nabla_\theta \log \pi_\theta^i(a^i | s^i, y^i) \Big] \approx$$
$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}} \Big[\{ \sum_{m=1}^{N} h_w^m(s^m, a^m, \nu^m, y^m) \} \nabla_\theta \log \pi_\theta^i(a^i | s^i, y^i) \Big] \tag{19}$$

We can show that:

$$\mathbb{E}_{\boldsymbol{s}} \Big[\mathbb{E}_{\boldsymbol{a},\boldsymbol{\nu}|\boldsymbol{s}} \Big[\{ \sum_{m \neq i} h_w^m(s^m, a^m, \nu^m, y^m) \} \nabla_\theta \log \pi_\theta^i(a^i | s^i, y^i) \Big] \Big] = 0$$

(proof omitted). Therefore, (19) simplifies to:

$$\mathbb{E}_{\boldsymbol{s}} \Big[\mathbb{E}_{a^i,\nu^i|\boldsymbol{s}} \Big[h_w^i(s^i, a^i, \nu^i, y^i) \nabla_\theta \log \pi_\theta^i(a^i | s^i, y^i) \Big] \Big] \tag{20}$$

[2] https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/other-readings/chapter8.pdf

The above expression has much lower variance than estimating gradients using (19). Let us now consider terms depending on deterministic policy gradient $\nabla_\theta \mu^i$:

$$\mathbb{E}\Big[Q^k(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}')\Big(\phi(s^i, s'^i)\nabla_\theta \mu^i_\theta(a^i, s^i, y^i) - \nabla_\theta \mathcal{A}(\mu^i_\theta(a^i, s^i, y^i))\Big)\Big]$$
$$\approx \mathbb{E}\Big[\{\sum_{m=1}^{N} h^m_w(s^m, a^m, \nu^m, y^m)\}\Big(\phi(s^i, s'^i)\nabla_\theta \mu^i_\theta(a^i, s^i, y^i)$$
$$-\nabla_\theta \mathcal{A}(\mu^i_\theta(a^i, s^i, y^i))\Big)\Big] \quad (21)$$

Empirically, the above expression also resulted in poor quality credit assignment because the gradient term $\nabla_\theta \mu^i_\theta$ is affected by the value accumulated by all the agents m. We therefore used an approximation where we only add the contributions from those agents which are in the immediate neighborhood of agent i given the joint-state $\boldsymbol{s}$, denoted using $\mathrm{Nb}(i, \boldsymbol{s})$. The expression becomes:

$$\mathbb{E}\Big[\{\sum_{m \in \mathrm{Nb}(i,\boldsymbol{s})} h^m_w(s^m, a^m, \nu^m, y^m)\}\Big(\phi(s^i, s'^i)\nabla_\theta \mu^i_\theta(a^i, s^i, y^i)$$
$$-\nabla_\theta \mathcal{A}(\mu^i_\theta(a^i, s^i, y^i))\Big)\Big] \quad (22)$$

Using the above credit assignment techniques, our approach worked significantly better than the vanilla gradient based learning. To estimate $h^i_w(s^i, a^i, \nu^i, y^i)$, we used empirical returns. That is, for a given joint-trajectory of all the agents from the simulator, the total discounted reward obtained by an agent i given its current state-action-observation being (s^i, a^i, ν^i, y^i) is treated as the target for $h^i_w(s^i, a^i, \nu^i, y^i)$.

Practical considerations: As number of agents can become quite large in practical ZBPF scenarios, optimizing a unique policy for each agent would require far too many samples to learn within a reasonable time. Therefore, we use policy sharing among agents. We have one set of parameters for each zone or $\theta = \{\theta_z \ \forall z \in Z\}$. Our approach resembles a traffic management system where traffic through a zone z is controlled using $\pi_{\theta_z}, \mu_{\theta_z}$. To differentiate among agents, we also provide agent id as part of the observation to the agent. This approach is suited for settings where number of start and goal zones are relatively fewer than the number of agents. This is a reasonable assumption, for example, in drone traffic management, where most drones takeoff and land at few specialized high-rise and safe locations in the city. If each agent has a unique start-goal zone, then the problem becomes highly combinatorial, and policy sharing among agents may not be effective.

4 Experiments

We evaluate our approach on several 2D grid maps and 3D maps. Our ZBPF simulator is constructed in the cross platform Unity3D engine which provides ml-agents framework for training intelligent agents, and a simple Python API to access simulation state. We vary several parameters including grid size, number of agents, start and goal zones. We compare our learning approach with credit assignment ('DCRL') against the vanilla gradient based approach ('VPG') in (17) where Q values are estimated by total discounted global return, and multiagent Q-learning based approach for discrete-continuous hybrid action spaces ('HA') (Fu et al. 2019). We have adapted the 'HA' approach for the ZBPF setting (details in supplementary). We optimize SOC combined with penalties for congestion (more details in supplementary). For comparison, we also show SOC when all the agents follow their shortest path ('SP'). This value is the lowest SOC any approach can achieve; but it also results in high congestion. All our experiments are run on a Linux machine with 3GHz CPU and 64 GB RAM. Maximum runtime allowed was 10 hours for any approach.

2D Open Grids: Figure 3 shows the SOC comparisons among DCRL, VPG and HA. For each grid, starting and goal locations were the top and bottom rows. For each agent, we randomly selected its start and goal zones from the top and bottom rows. As a result, multiple agents had the same start and/or goal zones. This setting is also challenging for our approach—if multiple agents are in the same start zone, the zone based policy has to properly assign different route/travel time to them. If agents follow each other on the shortest path to the goal zone, it creates high congestion. The capacity of each zone was sampled uniformly from [1, 2] for 4x4 grid, [1, 3] for 8x8 grid, and [1, 4] for 10x10 grid. The $t_{\min}, t_{\max}$ for each zone were $1, 5$ respectively. The travel time distribution used was the binomial distribution, which is the maximum entropy distribution with a given support and mean. Each training episode was cutoff after 500 time steps or after 10 hours. We varied both grid sizes and number of agents ranging from 4x4 grid, 2 agents to 10x10 grid 30 agents. For each setting, we generated 5 instances, and the average SOC (along with standard deviation) is shown in figure 3. This figure clearly shows that as the problem becomes more complex (increasing grid size, number of agents), our approach DCRL provides significantly better solution quality than VPG and HA. This is because DCRL performs credit assignment much more effectively than the VPG, and HA attempts to learn Q-values over joint state and action, which may be inaccurate for large number of agents. For larger grids (e.g., 10x10) and larger density of agents (e.g., 10x10, 30 agents), several agents did not reach their goal zone when using VPG based policy and Q learning based policy. To provide an estimate of best possible SOC, we also show the SOC when all agents move as fast as possible (using $t_{\min}$ time) on their shortest path to goal (denoted using 'SP'). DCRL is worse than SP (as expected), however, not by a large margin. This is expected as DCRL make agents take a longer route or take more time traveling between zones to minimize congestion. For the ZBPF problems we have generated as above, standard MAPF solvers including based on RL (such as by (Sartoretti et al. 2019)) are not applicable in a straightforward fashion. In our instances, multiple agents can have the same start zone, which is not allowed in standard MAPF, and they also do not incorporate uncertainty in travel time and higher capacity nodes.

Table 1 shows the results comparing congestion level between our approach, VPG and HA. For each time step, if number of agents in a zone z (say N_z) are more than its capacity C_z, than $N_z - C_z$ is added to the congestion level.

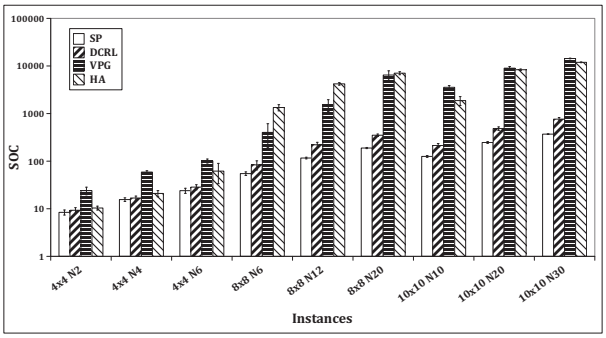


Figure 3: Sum-of-cost (SOC) comparisons among DCRL, VPG and HA (log-scale, lower is better). N# denotes number of agents.

Setting	Congestion			Stranded agents		
	DCRL	VPG	HA	DCRL	VPG	HA
4x4 N2	0	0	0	0	0	0
4x4 N4	0.11	0.02	0.17	0	0	0
4x4 N6	0.12	0.04	0.29	0	0	0
8x8 N6	0.06	0.18	0.11	0	0.40	2.08
8x8 N12	0.50	0.67	31.62	0	2.21	6.20
8x8 N20	1.74	4.15	114.70	0	11.45	10.71
10x10 N10	0.44	0.07	0.68	0	6.81	2.83
10x10 N20	0.97	3.21	130.29	0	17.25	14.18
10x10 N30	2.12	35.12	273.16	0	27.33	18.38

Table 1: Average congestion level and stranded agents comparisons

We also show, on an average, how many agents were unable to reach destination in these three approaches (or 'stranded agents'). These results clearly show that our approach is able to effectively minimize congestion over VPG and HA, and for all tested instances, agents were able to reach their destinations. The standard deviation in congestion metric was low for DCRL (less than 1 for all problems).

Figure 4 shows the learning curve for DCRL, VPG and HA for the largest grid with most number of agents (10x10, 30 agents). This figure clearly shows that with credit assignment, our approach was quite stable during learning and converged much faster and to a better quality than VPG. For this large setting, HA was unable to reduce the congestion level to an acceptable low level.

3D Maps: We also generated 3D maps ('SmallCube' and 'Twofloor') in the Unity environment (maps available in the supplementary material). SmallCube map and TwoFloor map have 30 zones and 142 zones respectively. Total number of agents were 10 and 20 for these two maps respectively. Capacity of each zone was uniformly sampled from [1, 3] and [1,4]. Agents moved from bottom left 5 zones (which were start zones) to top right 5 zones (which were goal zones) for SmallCube, and moved from bottom front zones to bottom back zones for TwoFloor map. Rest of the settings were same as for grid graphs. For such maps also, our approach DCRL worked much better than VPG and HA as expected. Table 2 shows the SOC, stranded agents and congestion level results. For SmallCube map, all agents

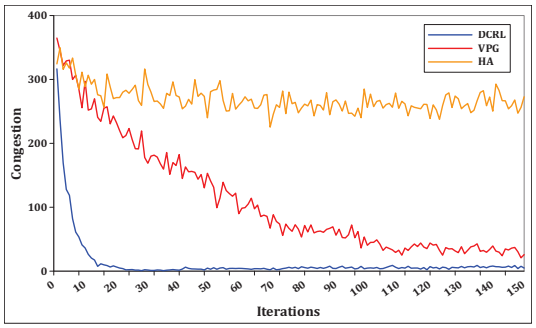


Figure 4: Learning curve (showing congestion level) of DCRL, VPG and HA on a 10x10 grid, 30 agents instance

Metric	Instance	Average		
		DCRL	VPG	HA
SOC	SmallCube	155.29	312.00	775.98
	TwoFloor	532.35	1300.73	6346.09
Stranded agents	SmallCube	0	0	0.33
	TwoFloor	0	0.2	8.66
Congestion	SmallCube	3.66	4.37	10.61
	TwoFloor	5.09	7.24	112.49

Table 2: Average SOC, stranded agents, and congestion level comparisons (over 5 instances).

were able to reach their goals in both DCRL and VPG approaches. But compared with DCRL, VPG gives higher SOC (lower SOC is better) and marginally greater congestion level. HA performed slightly worse in terms of stranded agents (which is 0.33), but also gives worst SOC and congestion level. For TwoFloor map, VPG gives reasonably higher SOC of 1300.73 versus 532.35 by DCRL. This is because there are slightly more average stranded agents (0.2) versus zero stranded agents by DCRL. Compared with DCRL and VPG, HA gives the worst SOC (6346.09), and it cannot work well for either making all agents reach goal zones or minimizing the congestion.

5 Conclusion

We addressed the problem of zone based multiagent path finding. Our work addressed the modeling of uncertainty in agent movement, and higher capacity zones, which standard MAPF solvers are unable to address in a single framework. We used a general model of movement uncertainty by modeling agent's transition function using an exponential family distribution. We developed a novel DC programming based perspective on the policy optimization problem in ZBPF, and showed how it enabled both planning and learning using a sequence of easier optimization problems. Furthermore, to make the learning approach converge faster, we developed multiagent credit assignment techniques that enabled faster and better quality learning than the standard gradient based approach without credit assignment. We demonstrated the effectiveness of our approach on a number of instances to find paths that minimized congestion, and enabled all agents to reach their goal zones.

Acknowledgments

This research is supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 2 (MOE2018-T2-1-179).

References

Barták, R.; Švancara, J.; and Vlk, M. 2018. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, 748–756.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4):819–840.

Bishop, C. M. 2006. *Pattern recognition and machine learning*. Berlin, Heidelberg: Springer-Verlag.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence*, 740–746.

Cap, M.; Novak, P.; Kleiner, A.; and Selecky, M. 2015. Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots. *IEEE Transactions on Automation Science and Engineering* 12(3):835–849.

Chang, Y. H.; Ho, T.; and Kaelbling, L. P. 2004. All learning is local: Multi-agent learning in global reward games. In *Advances in Neural Information Processing Systems*, 807–814.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Annual Symposium on Combinatorial Search, SoCS*, 29–37.

Foerster, J. N.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual Multi-Agent Policy Gradients. In *AAAI Conference on Artificial Intelligence*, 2974–2982.

Fu, H.; Tang, H.; Hao, J.; Lei, Z.; Chen, Y.; and Fan, C. 2019. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. In *International Joint Conference on Artificial Intelligence*, 2329–2335.

Gill, P. E.; Murray, W.; and Saunders, M. A. 2005. SNOPT: An SQP algorithm for large-scale constrained optimization (Reprinted from SIAM Journal Optimization, vol 12, pg 979-1006, 2002). *Siam Review* 47(1):99–131.

Harremoes, P. 2001. Binomial and poisson distributions as maximum entropy distributions. *IEEE Transactions on Information Theory* 47(5):2039–2041.

Hio, L. 2016. Traffic system for drones. https://www.straitstimes.com/singapore/traffic-system-for-drones.

Honig, W.; Kumar, T. K.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In *International Conference on Automated Planning and Scheduling*, 477–485.

Jaynes, E. T. 1957. Information theory and statistical mechanics. *Phys. Rev.* 106(4):620–630.

Juliani, A.; Berges, V.-P.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; and Lange, D. 2018. Unity: A General Platform for Intelligent Agents. *CoRR* abs/1809.0.

Lipp, T., and Boyd, S. 2016. Variations and extension of the convex-concave procedure. *Optimization and Engineering* 17(2):263–287.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hoenig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2017. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios.

Ma, H.; Kumar, T. K.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *AAAI Conference on Artificial Intelligence*, 3605–3612.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Satish Kumar, T. K.; Koenig, S.; and Choset, H. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters* 4(3):2378–2385.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *International Joint Conference on Artificial Intelligence*, 662–667.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Singh, A. J.; Nguyen, D. T.; Kumar, A.; and Lau, H. C. 2019. Multiagent decision making for maritime traffic management. In *AAAI Conference on Artificial Intelligence*, 6171–6178.

Surynek, P.; Kumar, T. K. S.; and Koenig, S. 2019. Multi-Agent Path Finding with Capacity Constraints.

Surynek, P. 2017. Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. *Annals of Mathematics and Artificial Intelligence* 81(3-4):329–375.

Teng, T.-H.; Lau, H. C.; and Kumar, A. 2017. Coordinating Vessel Traffic to Improve Safety and Efficiency. In *Internation Conference on Autonomous Agents and Multiagent Systems*, 141–149.

Wang, J.; Li, J.; Ma, H.; Koenig, S.; and Kumar, S. 2019. A New Constraint Satisfaction Perspective on Multi-Agent Path Finding: Preliminary Results. In *International Joint Conference on Autonomous Agents and Multiagent Systems*.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* 29(1):9–19.

Yu, J., and Lavalle, S. M. 2013. Planning optimal paths for multiple robots on graphs. *IEEE International Conference on Robotics and Automation* 3612–3617.

Yuille, A. L., and Rangarajan, A. The Concave-Convex Procedure (CCCP), url = http://papers.nips.cc/paper/2125-the-concave-convex-procedure-cccp, year = 2001. In *Advances in Neural Information Processing Systems*, 1033–1040.