Autonomous Robots manuscript No.
(will be inserted by the editor)

ALAN: Adaptive Learning for Multi-Agent Navigation

Julio Godoy · Tiannan Chen · Stephen J. Guy · Ioannis Karamouzas · Maria Gini

Received: date / Accepted: date

Abstract In multi-agent navigation, agents need to move towards their goal locations while avoiding collisions with other agents and obstacles, often without communication. Existing methods compute motions that are locally optimal but do not account for the aggregated motions of all agents, producing inefficient global behavior especially when agents move in a crowded space. In this work, we develop a method that allows agents to dynamically adapt their behavior to their local conditions. We formulate the multi-agent navigation problem as an action-selection problem and propose an approach, ALAN, that allows agents to compute time-efficient and collision-free motions. ALAN is highly scalable because each agent makes its own decisions on how to move, using a set of velocities optimized for a variety of navigation tasks. Experimental results show that agents using ALAN, in general, reach their destinations faster than using ORCA, a state-of-the-art collision avoidance framework, and two other navigation models.

Keywords Multi-agent navigation · Online learning · Action selection · Multi-agent coordination

Julio Godoy
Department of Computer Science, Universidad de Concepcion
Edmundo Larenas 219, Concepcion, Chile
E-mail: juliogodoy@gmail.com

Tiannan Chen, Stephen J. Guy and Maria Gini
Department of Computer Science and Engineering, University of Minnesota
200 Union Street SE, Minneapolis, MN 55455, USA

Ioannis Karamouzas
School of Computing, Clemson University
100 McAdams Hall, Clemson, South Carolina, SC 29634, USA

1 Introduction

Real-time goal-directed navigation of multiple agents is required in many domains, such as swarm robotics, pedestrian navigation, planning for evacuation, and traffic engineering. Conflicting constraints and the need to operate in real time make this problem challenging. Agents need to move towards their goals in a timely manner, but also need to avoid collisions with each other and the environment. In addition, agents often need to compute their own motion without any communication with other agents.

While decentralization is essential for scalability and robustness, achieving globally efficient motions is critical, especially in applications such as search and rescue, aerial surveillance, and evacuation planning, where time is critical. Over the past twenty years, many decentralized techniques for real-time multi-agent navigation have been proposed, with approaches such as Optimal Reciprocal Collision Avoidance (ORCA) [5] being able to provide guarantees about collision-free motion for the agents. Although such techniques generate locally efficient motions for each agent, the overall flow and global behavior of the agents can be far from efficient; agents plan only for themselves and do not consider how their motions affect the other agents. This can lead to inefficient motions, congestion, and even deadlocks.

In this paper, we are interested in situations where agents have to minimize their overall travel time. We assume each agent has a preferred velocity indicating its desired direction of motion (typically oriented towards its goal) and speed. An agent runs a continuous cycle of sensing and acting. In each cycle, it has to choose a new velocity that avoids obstacles but is as close as possible to its preferred velocity. We show that by intelligently selecting preferred velocities that account for the global state of the multi-agent system, the time efficiency of the entire crowd can be significantly improved compared to state of the art algorithms.

In our setting, agents learn how to select their velocities in an online fashion without communicating with each other. To do so, we adapt a multi-armed bandit formulation to the preferred velocity selection problem and present ALAN (Adaptive Learning Approach for Multi-Agent Navigation). With ALAN, agents choose from a set of actions, one at each time step, based on a combination of their goals and how their motions will affect other agents. We show how critical the set of available actions is to performance, and we present a Markov Chain Monte Carlo learning method to learn an optimized action space for navigation in a variety of environments. Together with a scheme that guarantees collision-free motions, these features allow ALAN agents to minimize their overall travel time. [1]

Main Results. This paper presents four main contributions. First, we formulate the multi-agent navigation problem in a multi-armed bandit setting. This enables each agent to decide its motions independently of the other agents. The other agents influence indirectly how an agent moves, because they affect the reward the agent receives. The independence of the choices made by each agent makes the approach highly scalable. Second, we propose an online action selection method inspired by the Softmax action selection technique [48], which achieves the exploration exploitation tradeoff. Third, we propose a Markov Chain Monte Carlo method to learn offline an optimized action set for specific navigation environments, as well as an action set optimized for multiple navigation scenarios. Last, we show experimentally that our approach leads to more time efficient motions in a variety of scenarios, reducing the travel time of all agents as compared to ORCA, the Social Forces model for simulating pedestrian dynamics [19], and the pedestrian model for collision avoidance proposed in [27].

This work is an extended version of [12], which introduced a multi-armed bandit formulation for multi-agent navigation problems. Compared to [12], here we reduce ALAN's dependency on parameters, present an offline approach to learn an optimized action set, and include an extended experimental analysis of ALAN.

The rest of the paper is organized as follows. In Section 2, we review relevant related work. In Section 3, we provide background on collision avoidance methods, especially on ORCA which is used in ALAN. In Section 4, we present our problem formulation for multi-agent navigation. ALAN and its components are described in Section 5, while our experimental setup and performance metric are described in Section 6, where we also present the scenarios we use to evaluate our approach, and experimental results. Section 7 presents our Markov Chain Monte Carlo method for learning action spaces for different navigation environments. A thorough experimental analysis of the performance of ALAN is in Section 8, where we also discuss its applicability in multi-robot systems. Finally, we conclude and present future research plans in Section 9.

2 Related Work

Extensive research in the areas of multi-agent navigation and learning has been conducted over the last decade. In this section, we present an overview of prior work most closely related to our approach. For a more comprehensive discussion on multi-agent navigation and learning we refer the reader to the surveys of Pelechano et al. [38] and Buşoniu et al. [7], respectively.

2.1 Multi-Agent Navigation

Numerous models have been proposed to simulate individuals and groups of interacting agents. The seminal work of Reynolds on *boids* has been influential on this field [43]. Reynolds used simple local rules to create visually compelling flocks of birds and schools of fishes. Later he extended his model to include autonomous agent behaviors [42]. Since Reynolds's original work, many crowd simulation models have been introduced that account for groups [4], cognitive and behavioral rules [10,44], biomechanical principles [15] and sociological or psychological factors [37,14,40]. Recent work models the contagion of psychological states in a crowd of agents, for example, in evacuation simulations [50]. Our approach, in contrast, does not make assumptions about the psychological states of the agents, therefore it is more generally applicable.

An extensive literature also exists on modeling the local dynamics of the agents and computing collision-free motions. Methods that have been proposed to prevent collisions during navigation can be classified as *reactive* and *anticipatory*.

In reactive collision avoidance, agents adapt their motion to other agents and obstacles along their paths. Many reactive methods [43,42,18,29,41] use artificial repulsive forces to avoid collisions. However, these techniques do not anticipate collisions. Only when agents are sufficiently close, they react to avoid collisions. This can lead to oscillations and local minima. Another limi-

[1] Videos highlighting our work can be found in `http://motion.cs.umn.edu/r/ActionSelection`

tation of these methods is that the forces must be tuned separately for each scenario, limiting their robustness.

In anticipatory collision avoidance, agents predict and avoid potential upcoming collisions by linearly extrapolating their current velocities. In this line, *geometrically based* algorithms compute collision-free velocities for the agents using either sampling [52,39,28, 36] or optimization techniques [5,13].

We focus on minimizing the travel time of the agents, but other metrics have been studied. For example, the work in [46,54,26] minimizes the total length of the path of the agents by formulating the path planning problem as a mixed integer linear program. Coordinating the motion of a set of pebbles in a graph to minimize the number of moves was studied in [32].

2.2 Reinforcement Learning

Many learning approaches used for robots and agents derive from the reinforcement learning literature [7]. Reinforcement Learning (RL) addresses how autonomous agents can learn by interacting with the environment to achieve their desired goal [47]. An RL agent performs actions that affect its state and environment, and receives a reward value which indicates the quality of the performed action. This reward is used as feedback for the agent to improve its future decisions. Different approaches have been proposed to incorporate RL when multiple agents share the environment (see [7,31,51] for extensive overviews).

In multi-agent RL algorithms, agents typically need to collect information on how other agents behave and find a policy that maximizes their reward. This is expensive when the state space is large and requires a significant degree of exploration to create an accurate model for each agent. Hence, approaches that model the entire environment focus on small problems and/or a small number of agents. To reduce complexity, some approaches focus on the local neighborhood of each agent [55,56]. By considering a local neighborhood, the state space of each agent is reduced. To completely avoid the state space complexity, the learning problem can be formulated as a multi-armed bandit problem [47], where the agents use the reward of each action to make future decisions. In multi-armed bandit problems, it is critical to balance exploiting the current best action and exploring potentially better actions [2,33].

2.2.1 Action Selection Techniques

A variety of approaches aim at balancing exploration and exploitation, which is critical for online learning problems such as ours.

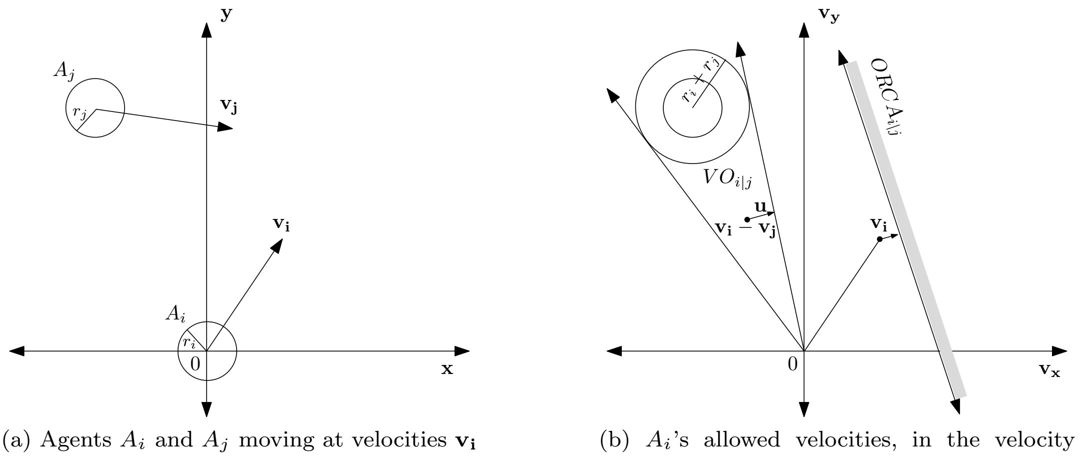
A simple approach is ϵ-greedy, which selects the highest valued action with probability 1-ϵ, and a random action with probability ϵ, for $0 \leq \epsilon \leq 1$. The value of ϵ indicates the degree of exploration that the agent performs [48]. Because of its probabilistic nature, ϵ-greedy can find the optimal action, without taking into account the difference between the values of the actions. This means that ϵ-greedy does the same amount of exploration regardless of *how* much better the best known action is, compared to the other actions.

Another widely used action-selection technique is the upper confidence bounds (UCB) algorithm [3]. UCB is a deterministic method that samples the actions proportionally to the upper-bound of the estimated value of their rewards (based on their current average reward) and their confidence interval (computed using a relation between the number of times each action was selected and the total number of action taken so far by the agent). Unlike ϵ-greedy, UCB considers the value of all actions when deciding which one to choose. However, it does unnecessary exploration when the reward distribution is static (i.e., the best action does not change).

A method that combines the probabilistic nature of ϵ-greedy and that accounts for the changing reward structure is the Softmax action selection strategy. Softmax biases the action choice depending on the relative reward value, which means that it increases exploration when all actions have similar value, and it reduces it when some (or one) action is significantly better than the rest. The action selection method we use is based on the Softmax strategy, due to these properties.

2.3 Learning in Multi-Agent Navigation

Extensive work has also been done on learning and adapting motion behavior of agents in crowded environments. Depending on the nature of the learning process, the work can be classified in two main categories: offline and online learning. In offline learning, agents repeatedly explore the environment and try to learn the optimal policy given an objective function. Examples of desired learned behaviors include collision avoidance, shortest path to destination, and specific group formations. As an example, the work in [22] uses inverse reinforcement learning for agents to learn paths from recorded training data. Similarly, the approach in [49] applies Q-learning to plan paths for agents in crowds. In this approach, agents learn in a series of episodes the best path to their destination. A SARSA-based [48] learning algorithm has also been used in [34] for offline learning of behaviors in crowd simulations. The approach in [8] analyzes different strategies for sharing policies between agents to speed up the learning



(a) Agents A_i and A_j moving at velocities $\mathbf{v_i}$ and $\mathbf{v_j}$, respectively

(b) A_i's allowed velocities, in the velocity space

Fig. 1: (a) Two agents, A_i and A_j, moving towards a potential collision. (b) The set of allowed velocities for agent i induced by agent j is indicated by the half-plane delimited by the line perpendicular to $\hat{\mathbf{u}}$ through the point $\mathbf{v_i} + \frac{1}{2}\mathbf{u}$, where $\mathbf{u}$ is the vector from $\mathbf{v_i} - \mathbf{v_j}$ to the closest point on the boundary of $VO_{i|j}$

process in crowd simulations. In the area of swarm intelligence, the work in [23] uses evolutionary algorithms for robotics, learning offline the parameters of the fitness function and sharing the learned rules in unknown environments.

Offline learning has significant limitations, which arise from the need to train the agents before the environment is known. In contrast, the main part of our work is an online learning approach. In online approaches, agents are given only partial knowledge of the environment, and are expected to adapt their strategies as they discover more of the environment. Our approach allows agents to adapt online to unknown environments, without needing explicit communication between the agents.

3 Background

In this section, we provide background information on the method that agents employ to avoid collisions.

3.1 ORCA

The Optimal Reciprocal Collision Avoidance framework (ORCA) is an anticipatory collision avoidance that builds on the concept of Velocity Obstacles [9], where agents detect and avoid potential collisions by linearly extrapolating their current velocities. Given two agents, A_i and A_j, the set of velocity obstacles $VO_{A_i|A_j}$ represents the set of all relative velocities between i and j that will result in a collision at some future moment. Using the VO formulation, we can guarantee collision avoidance by choosing a relative velocity that lies outside the set $VO_{A_i|A_j}$. Let $\mathbf{u}$ denote the minimum change in the relative velocity of i and j needed to avoid the collision. ORCA assumes that the two agents will *share* the responsibility of avoiding it and requires each agent to change its current velocity by at least $\frac{1}{2}\mathbf{u}$. Then, the set of feasible velocities for i induced by j is the half-plane of velocities given by:

$$ORCA_{A_i|A_j} = \{\mathbf{v} \;|(\mathbf{v} - (\mathbf{v}_i + \frac{1}{2}\mathbf{u})) \cdot \hat{\mathbf{u}}\},$$

where $\hat{\mathbf{u}}$ is the normalized vector $\mathbf{u}$ (see Fig. 1). Similar formulation can be derived for determining A_i's permitted velocities with respect to a static obstacle O_k. We denote this set as $ORCA_{A_i|O_k}$.

In a multi-agent setting, ORCA works as follows. At each time step of the simulation, each agent A_i infers its set of *feasible* velocities, FV_{A_i}, from the intersection of all permitted half-planes $ORCA_{A_i|A_j}$ and $ORCA_{A_i|O_k}$ induced by each neighboring agent j and obstacle O_k, respectively. Having computed FV_{A_i}, the agent selects a new velocity $\mathbf{v}_i^{\text{new}}$ for itself that is closest to a given preferred velocity $\mathbf{v}_i^{\text{pref}}$ and lies inside the region of feasible velocities:

$$\mathbf{v}_i^{\text{new}} = \underset{\mathbf{v} \in FV_{A_i}}{\arg\min} \|\mathbf{v} - \mathbf{v}_i^{\text{pref}}\|. \tag{1}$$

The optimization problem in (1) can be efficiently solved using linear programming, since FV_{A_i} is a convex region bounded by linear constraints. Finally, agent i updates its position based on the newly computed velocity. As ORCA is a decentralized approach, each agent computes its velocity independently.

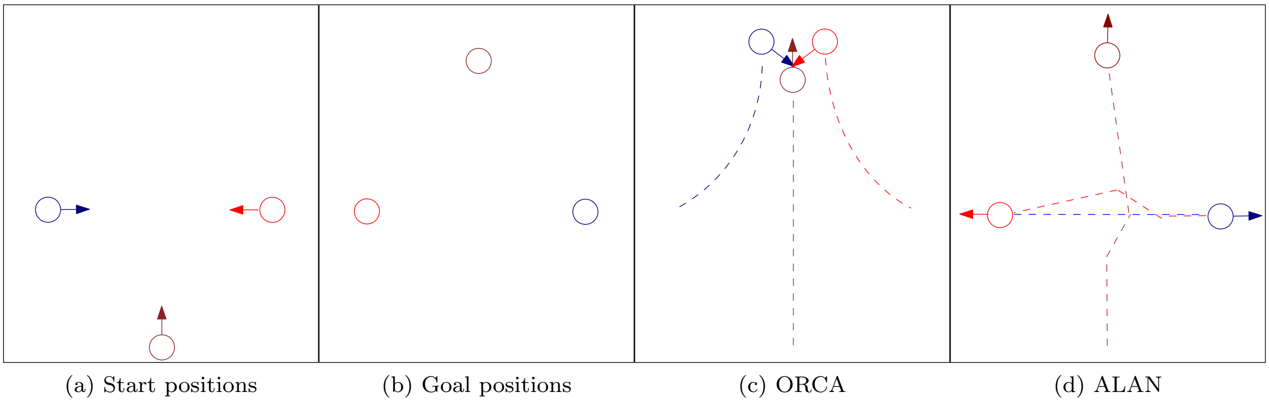

(a) Start positions (b) Goal positions (c) ORCA (d) ALAN

Fig. 2: Three agents cross paths. (a) Initial positions of the agents. (b) Goal positions of the agents. (c) When navigating with ORCA, the agents run into and push each other resulting in inefficient paths. (d) When using ALAN the agents select different preferred velocities which avoid local minima, resulting in more efficient paths.

In addition, each agent typically uses its goal-oriented velocity $\mathbf{v}_i^{\text{goal}}$ as the preferred velocity given as input to ORCA in (1). We refer the reader to [5] for more details.

3.2 Limitations of ORCA

Although ORCA guarantees collision-free motions and provides a locally optimal behavior for each agent, the lack of coordination between agents can lead to globally inefficient motions. For an example, see Fig. 2. Here, because the agents follow only their goal-oriented preferred velocity, they get stuck in a local minimum resulting in the trajectories shown in Fig. 2(c). If instead the agents behaved differently, for instance, by selecting a different $\mathbf{v}^{\text{pref}}$ for a short period of time, they might find a larger region of feasible velocities. This might indirectly help to alleviate the overall congestion, benefiting all agents. Our proposed approach, ALAN, addresses this limitation, by allowing agents to adapt their preferred velocity in an online manner, hence improving their motion efficiency. An example of the trajectories generated by our approach can be seen in Fig. 2(d).

4 Problem Formulation

In our problem setting, given an environment and a set A of agents, each with a start and a goal position, our goal is to enable the agents to reach their goals as soon as possible and without collisions. We also require that the agents move *independently* and without explicitly communicating with each other. For simplicity, we model each agent as a disc which moves on a 2D plane that may also contain a set of k static obstacles $\mathcal{O}$ (approximated by line segments in all our experiments).

Given n agents, let agent A_i have radius r_i, goal position $\mathbf{g}_i$, and maximum speed $\upsilon_i^{\max}$. Let also $\mathbf{p}_i^t$ and $\mathbf{v}_i^t$ denote the agent's position and velocity, respectively, at time t. Furthermore, agent A_i has a preferred velocity $\mathbf{v}_i^{\text{pref}}$ at which it prefers to move. Let $\mathbf{v}_i^{\text{goal}}$ be the preferred velocity directed towards the agent's goal $\mathbf{g}_i$ with a magnitude equal to $\upsilon_i^{\max}$. The main objective of our work is to minimize the travel time of the set of agents A to their goals, while guaranteeing collision-free motions. To measure this global travel time, we could consider the travel time of the last agent that reaches its goal. However, this value does not provide any information of the travel time of all the other agents. Instead, we measure this travel time, $TTime(A)$, by accounting for the average travel time of all the agents in A and its spread. Formally:

$$\begin{aligned} TTime(A) = {} & \mu\,(TimeToGoal(A)) \\ & + 3\ \sigma\,(TimeToGoal(A)) \end{aligned} \tag{2}$$

where $TimeToGoal(A)$ is the set of travel times of all agents in A from their start positions to their goals, and $\mu(\cdot)$ and $\sigma(\cdot)$ are the average and the standard deviation (using the unbiased estimator) of $TimeToGoal(A)$, respectively. If the times to goals of the agents follow a normal distribution, then $TTime(A)$ represents the upper bound of the $TimeToGoal(A)$ for approximately 99.7% of the agents. Even if the distribution is not normal, at least 89% of the times will fall within three standard deviations (Chebyshev's inequality). Our objective can be formalized as follows:

$$\begin{array}{lll} \text{minimize} & TTime(A) & \\ \text{s.t.} & \|\mathbf{p}_i^t - \mathbf{p}_j^t\| > r_i + r_j, \underset{i \neq j}{\forall} i,j \in [1,n] & \\ & dist(\mathbf{p}_i^t, O_j) > r_i, \forall i \in [1,n], j \in [1,k] & \\ & \|\mathbf{v}_i^t\| \le \upsilon_i^{\max}, \qquad \forall i \in [1,n] & \end{array} \tag{3}$$

where $dist(\cdot)$ denotes the shortest distance between two positions. To simplify the notation, in the rest of the paper we omit the index of the specific agent being referred, unless it is needed for clarity.

Minimizing Eq. 3 for a large number of agents using a centralized planner with complete information is intractable (PSPACE-hard [24]), given the combinatorial nature of the optimization problem and the continuous space of movement for the agents. Since we require that the agents navigate *independently* and without explicit communication with each other, Eq. 3 has to be minimized in a decentralized manner. As the agents do not know in advance which trajectories are feasible, the problem becomes for each agent to decide how to move at each timestep, given its perception of the local environment. This is the question addressed by our online learning approach, ALAN, which is described next.

5 ALAN

ALAN is an action selection framework, which provides a set of preferred velocities an agent can choose from, and a reward function the agent uses to evaluate the velocities and select the velocity to be used next. ALAN keeps an updated reward value for each action using a moving time window of the recently obtained rewards. If information about the set of navigation environments is available, ALAN can take advantage of an *action learning* approach to compute, in an offline manner, an action set that is optimized for one or a set of scenarios (see Section 7).

In ALAN, each agent runs a continuous cycle of sensing and action until it reaches its destination. To guarantee real-time behavior, we impose a hard time constraint of 50 ms per cycle. We assume that the radii, positions and velocities of nearby agents and obstacles can be obtained by sensing. At each cycle the agent senses and computes its new collision-free velocity which is used until the next cycle. The velocity has to respect the agent's geometric and kinematics constraints while ensuring progress towards its goal.

To achieve this, ALAN follows a two-step process. First, the agent selects a preferred velocity $\mathbf{v}^{\text{pref}}$ (as described later in Section 5.3). Next, this $\mathbf{v}^{\text{pref}}$ is passed to ORCA which produces a collision-free velocity $\mathbf{v}^{\text{new}}$, which is the velocity the agent will use during the next timestep.

Algorithm 1 shows an overview of ALAN. This algorithm is executed at every cycle. If an action is to be selected in the current cycle (line 3, in average every 0.2 s), the Softmax action selection method (presented in Section 5.3) returns a $\mathbf{v}^{\text{pref}}$ (line 4), which is passed to ORCA. After computing potential collisions, ORCA returns a new collision-free velocity $\mathbf{v}^{\text{new}}$ (line 6), and the *getAction* method returns the action a that corresponds to the $\mathbf{v}^{\text{pref}}$ selected (line 7). This action a is executed (line 8), which moves the agent with the collision-free velocity $\mathbf{v}^{\text{new}}$ for the duration of the cycle, before updating the agent's position for the next simulation step (line 9). The agent determines the quality of the action a (lines 10-12) by computing its reward value (see Section 5.1). This value becomes available to the action selection mechanism, which will select a new $\mathbf{v}^{\text{pref}}$ in the next cycle. This cycle repeats until the agent reaches its goal.

Algorithm 1: The ALAN algorithm for an agent

```
 1: initialize simulation
 2: while not at the goal do
 3:     if $UpdateAction(t)$ then
 4:         $\mathbf{v}^{\text{pref}} \leftarrow Softmax(Act)$
 5:     end if
 6:     $\mathbf{v}^{\text{new}} \leftarrow ORCA(\mathbf{v}^{\text{pref}})$
 7:     $a \leftarrow getAction(\mathbf{v}^{\text{pref}})$
 8:     $Execute(a)$
 9:     $\mathbf{p}^{t} \leftarrow \mathbf{p}^{t\text{-}1} + \mathbf{v}^{\text{new}} \cdot \Delta t$
10:     $\mathcal{R}_a^{goal} \leftarrow GoalReward(a^{t-1})$ (cf. Eq. 5)
11:     $\mathcal{R}_a^{polite} \leftarrow PoliteReward(a^{t-1})$ (cf. Eq. 6)
12:     $\mathcal{R}_a \leftarrow (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}$
13: end while
```

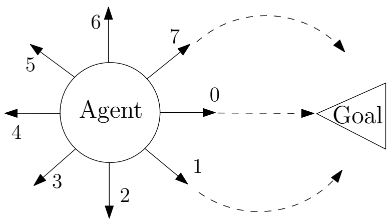


Fig. 3: Example set of actions with the corresponding action ID. The eight actions correspond to moving at 1.5 m/s with different angles with respect to the goal: 0°, 45°, 90°, 135°, −45°, −90°, −135° and 180°.

The main issue is how an agent should choose its preferred velocity. Typically, an agent would prefer a velocity that drives it closer to its goal, but different velocities may help the entire set of agents to reach their destinations faster (consider, for example, an agent moving backwards to alleviate congestion). Therefore, we allow the agents to use different *actions*, which correspond to different preferred velocities (throughout the rest of this paper, we will use the terms preferred velocities and actions interchangeably). In principle, finding the

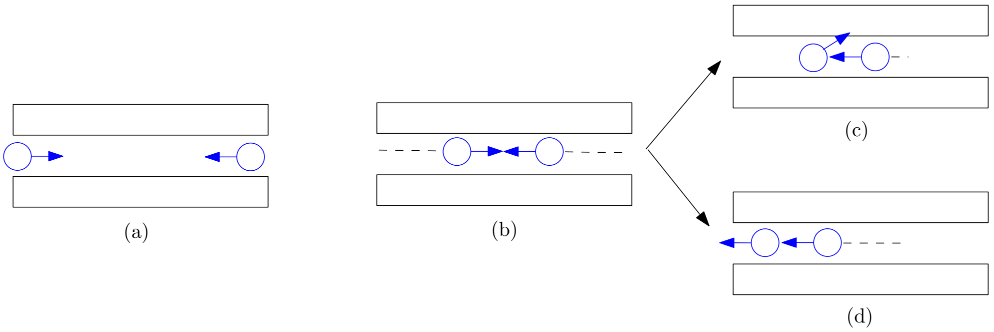


Fig. 4: Two agents moving to their goals in opposite sides of the corridor. Different behaviors are produced by optimizing different metrics. (b) When meeting in the middle of the corridor, agents cannot continue their goal oriented motions without colliding. (c) Considering only goal progress when choosing actions results in one agent slowly pushing the other out of the corridor. (d) Considering both goal progress and effect of action on other agents results in one agent moving backwards to help the other move to its goal, reducing the travel time for both.

best motion would require each agent to make a choice at every step in a continuous 2D space, the space of all possible speeds and directions. This is not practical in real-time domains. Instead, agents plan their motions over a discretized set of a small number of preferred velocities, the set *Act*. An example set of 8 actions uniformly distributed in the space of directions is shown in Fig. 3. We call this set *Sample* set.

Different action sets affect the performance of the agents. We analyze this in Section 7, where we present an offline learning method to find an optimal set of actions.

5.1 Reward Function

The quality of an agent's selected $\mathbf{v}^{\text{pref}}$ is evaluated based on two criteria: how much it moves the agent to its goal, and its effect on the motion of nearby agents. The first criterion allows agents to reach their goals, finding non-direct goal paths when facing congestion or static obstacles. The second criterion encourages actions that do not slow down the motion of other agents. To do this, agents take advantage of the reciprocity assumption of ORCA: when a collision is predicted, both potentially colliding agents will deviate to avoid each other. Hence, if a collision-free $\mathbf{v}^{\text{new}}$ computed by ORCA is different from the selected preferred velocity $\mathbf{v}^{\text{pref}}$, it also indicates a deviation for another agent. Therefore, to minimize the negative impact of its decisions on the nearby agents, i.e., to be *polite* towards them, each agent should choose actions whose $\mathbf{v}^{\text{new}}$ is similar to the $\mathbf{v}^{\text{pref}}$ that produced it. This duality of goal oriented and "socially aware" behaviors, in humans, has been recently studied in [45]. Here, we show that considering both criteria in the evaluation of each action reduces the travel time of the agents overall. See Fig. 4 for an example.

Specifically, we define the reward $\mathcal{R}_a$ for an agent performing action a to be a convex combination of a *goal-oriented* component and a *politeness* component:

$$\mathcal{R}_a = (1-\gamma)\cdot\mathcal{R}_a^{goal} + \gamma\cdot\mathcal{R}_a^{polite}, \tag{4}$$

where the parameter γ, called *coordination factor*, controls the influence of each component in the total reward ($0 \le \gamma < 1$).

The *goal-oriented* component $\mathcal{R}_a^{goal}$ computes the scalar product of the collision-free velocity $\mathbf{v}^{\text{new}}$ of the agent with the normalized vector pointing from the position $\mathbf{p}$ of the agent to its goal $\mathbf{g}$. This component promotes preferred velocities that lead the agent as quickly as possible to its goal. Formally:

$$\mathcal{R}_a^{goal} = \mathbf{v}^{\text{new}}\cdot\frac{\mathbf{g}-\mathbf{p}}{\|\mathbf{g}-\mathbf{p}\|} \tag{5}$$

The *politeness* component $\mathcal{R}_a^{polite}$ compares the executed preferred velocity with the resulting collision-free velocity. These two velocities will be similar when the preferred velocity does not conflict with other agents' motions, and will be different when it leads to potential collisions. Hence, the similarity between $\mathbf{v}^{\text{new}}$ and $\mathbf{v}^{\text{pref}}$ indicates how polite is the corresponding action, with respect to the motion of the other agents. Polite actions reduce the constraints on other agents' motions, allowing them to move and therefore advancing the global simulation state. Formally:

$$\mathcal{R}_a^{polite} = \mathbf{v}^{\text{new}}\cdot\mathbf{v}^{\text{pref}} \tag{6}$$

If an agent maximizes $\mathcal{R}_a^{goal}$, it would not consider the effects of its actions on the other agents. On the other hand, if the agent tries to maximize $\mathcal{R}_a^{polite}$, it has no incentive to move towards its goal, which means it might never reach it. Therefore, an agent should aim at maximizing a combination of both components. Different behaviors may be obtained with different values of γ. In Section 6.7, we analyze how sensitive the performance of ALAN is to different values of γ. Overall, we found that $\gamma = 0.4$ provides an appropriate balance between these two extremes.

Fig. 5 shows an example of conditions an agent may encounter. Here, there is congestion on one side of the agent, which results in low reward values for the left angled motion. The other actions are not constrained, and consequently their reward value is higher. In this case, the agent will choose the straight goal-oriented action, as it maximizes $\mathcal{R}_a$.

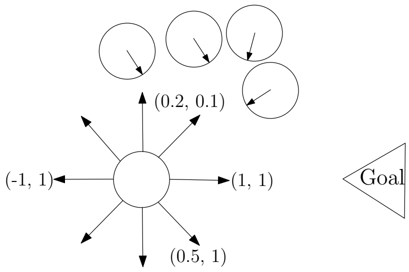


Fig. 5: Example of reward values for different actions under clear and congested local conditions. The reward $\mathcal{R}_a$ of each action a is shown as a pair of goal-oriented and a politeness components ($\mathcal{R}_a^{goal}$, $\mathcal{R}_a^{polite}$).

5.2 Multi-armed Bandit Formulation

As the number of states is very large, we adapt a stateless representation. Each agent can select one action at a time, hence the question is which one should the agent execute at a given time. In ALAN, agents learn the reward value of each action through its execution, in an online manner, and keep the recently obtained rewards (using a moving time window of the rewards) to decide how to act. We allow a chosen action to be executed for a number of cycles, and perform an a-posteriori evaluation to account for bad decisions. This way, the problem of deciding how to move becomes a resource allocation problem, where agents have a set of alternatives strategies and have to learn their estimated value via sampling, choosing one at each time in an online manner until they reach their goals.

Online learning problems with a discrete set of actions and stateless representation can be well formulated as multi-armed bandit problems. In a multi-armed bandit problem, an agent makes sequential decisions on a set of actions to maximize its expected reward. This formulation is well-suited for stationary problems, as existing algorithms guarantee a logarithmic bound on the regret [3]. Although our problem is non-stationary in a global sense, as the joint local conditions of the agents are highly dynamic, individual agents can undergo periods where the reward distribution changes very slowly. We refer to Fig. 6 for an example of a navigation task, where we can distinguish three periods with different reward distributions.

Therefore, by learning the action that maximizes a local reward function (Eq. 4) in each of these stationary periods, agents can adapt to the local conditions.

5.3 Action Selection

We now describe how ALAN selects, at each action decision step, one of the available actions based on their computed reward values and a probabilistic action-selection strategy, Softmax.

5.3.1 Softmax

Softmax is a general action selection method that balances exploration and exploitation in a probabilistic manner [48,57,53]. This method biases the action selection towards actions that have higher value (or reward, in our terminology), by making the probability of selecting an action dependent on its current estimated value. The most popular Softmax method uses the Boltzmann distribution to select among the actions. Assuming that $\mathcal{R}_a$ is the reward value of action a, the probability of choosing a is given by the following equation [48]:

$$Softmax(a) = \exp\left(\frac{\mathcal{R}_a}{\tau}\right) \Big/ \sum_{a=1}^{|Act|} \exp\left(\frac{\mathcal{R}_a}{\tau}\right) \tag{7}$$

The degree of exploration performed by a Boltzmann-based Softmax method is controlled by the parameter τ, also called the *temperature*. With values of τ close to zero the highest-valued actions are more likely to be chosen, while high values of τ make the probability of choosing each action similar. We use a value of τ=0.2, as we found that it shows enough differentiation between different action values without being too greedy.

Another critical design issue of our action selection method is the duration of the time window used. Keeping old samples with low values might make a good action look bad, but discarding them too quickly will ignore the past. Because of this, we use a moving time window of the most recently obtained rewards, and compute the estimated value of each action based only

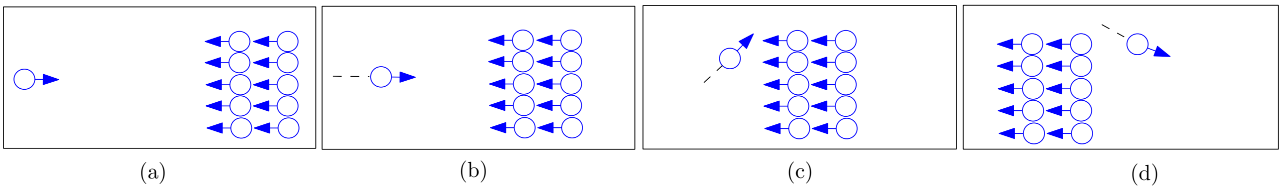


Fig. 6: Distinguishable periods of different reward distribution for the agent on the left. (a) The agent must reach its goal on the other side of a group of agents moving in the opposite direction. The optimal action in each period changes between (b) the goal oriented motion, (c) the sideways motion to avoid the incoming group, and (d) the goal oriented motion again, once the agent has avoided the group.

on the rewards in that time window, using the last sampled reward for each. If an action has not been sampled recently, it is assumed to have a neutral (zero) value, which represents the uncertainty of the agent with respect to the real value of the action. Actions with a neutral value have a low probability of being selected if the currently chosen action has a "good" value (>0), and have a high probability of being selected if the currently chosen action has a "bad" value (<0). When making an action decision, an agent retrieves the last sampled reward value for each action in the time window, or zero if the action has not been sampled recently. These values are then used by Softmax (Eq. 7) to determine the probability of each action being chosen.

In Section 6.6 we analyze the effect of different sizes of time window on the performance of ALAN.

5.3.2 Evolution of rewards during simulation

As agents move to their goals, their evaluation of the available actions affects the probability of choosing each action. Fig. 7 shows three simulation states of a navigation task while Table 1 shows, for each action of the black agent, the computed rewards and probability of being chosen as the next action. The goal of this evaluation is to empirically show how the estimated value of each action changes as the agent faces different conditions, and how these estimates affect the probability of the action being chosen.

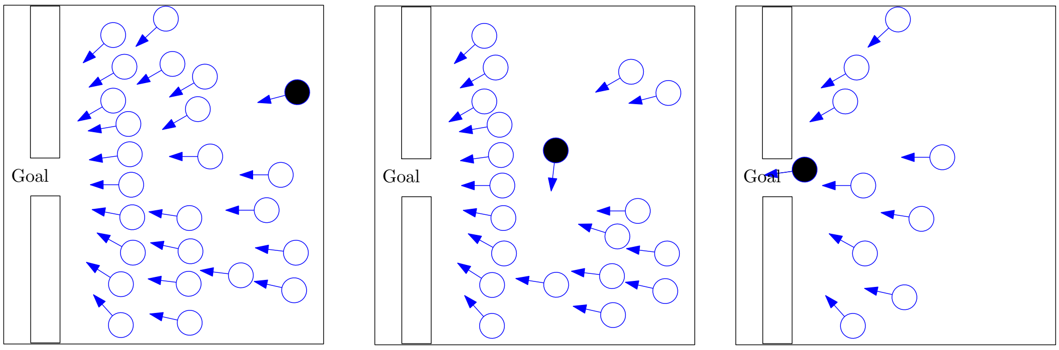
In the Initial state (Fig. 7(a)), the black agent can move unconstrained towards the goal, which is reflected in the high reward and corresponding probability of the goal oriented action (ID 0). In the Middle state (Fig. 7(b)), the black agent faces congestion that translates into a low reward for the goal oriented action. Instead, it determines that the action with the highest value is moving left (ID 6), which also has the highest probability of being chosen. Finally, in the End state (Fig. 7(c)), the goal path of the black agent is free. Through exploration, the black agent determines that the goal oriented motion (ID 0) is again the one with the best value, though with lower reward value than in the beginning, as the wall prevents the agent from moving at full speed. With a 56.7% probability, the agent selects the goal oriented motion and eventually reaches its goal. Note that the actions not sampled during the time window used in this experiment (2s) are assigned the neutral zero value.

6 Evaluation

We now present the experimental setup, performance metrics, and scenarios used to compare the performance of ALAN to other navigation approaches (Section 6.4). We also evaluate the design choices of ALAN, specifically the action selection method (Section 6.5), the time window length (Section 6.6), and the balance between goal progress and politeness, controlled by the coordination factor γ (Section 6.7) in the reward function. Additional results are presented later, after we extend the action selection method to include learning the action space.

6.1 Experimental Setup

We implemented ALAN in C++. Results were gathered on an Intel Core i7 at 3.5 GHz. Each experimental result is the average over 30 simulations. In all our runs, we updated the positions of the agents every $\Delta t = 50\,\mathrm{ms}$ and set the maximum speed $\upsilon^{\max}$ of each agent to 1.5 m/s and its radius to 0.5 m. Agents could sense other agents within a 15 m radius, and obstacles within 1 m. To avoid synchronization artifacts, agents are given a small random delay in how frequently they can update their $\mathbf{v}^{\mathrm{pref}}$ (with new $\mathbf{v}^{\mathrm{pref}}$ decisions computed every 0.2 s on average). This delay also gives ORCA a few timesteps to incorporate sudden velocity changes before the actions are evaluated. Small random perturbations were added to the preferred velocities of the agents to prevent symmetry problems.



(a) Initial (b) Middle (c) End

Fig. 7: Screen shots of three states of a navigation problem. (a) Initially, the black agent can move unconstrained towards the goal. (b) During its interaction with other agents, the black agent moves sideways since this increases its reward. (c) Finally, when its goal path is free, the black agent moves again towards the goal.

Simulation state		Action ID							
		0	1	2	3	4	5	6	7
Initial	reward	**0.997**	0	0	0.147	0	0.145	0	0
	prob	**94.1%**	0.64%	0.64%	1.34%	0.64%	1.33%	0.64%	0.64%
Middle	reward	-0.05	-0.42	-0.54	0	0.001	-0.192	**0.456**	0
	prob	5.4%	0.83%	0.46%	7.1%	7.1%	2.7%	**69.3%**	7.1%
End	reward	**0.63**	0.47	0	0.48	0	0	0.177	0
	prob	**56.7%**	25%	2.4%	3%	2.4%	2.4%	5.8%	2.4%

Table 1: Reward values and probability for each action to be chosen by the black agent using ALAN in the three different states shown in Fig. 7. See Fig. 3 for the corresponding set of actions.

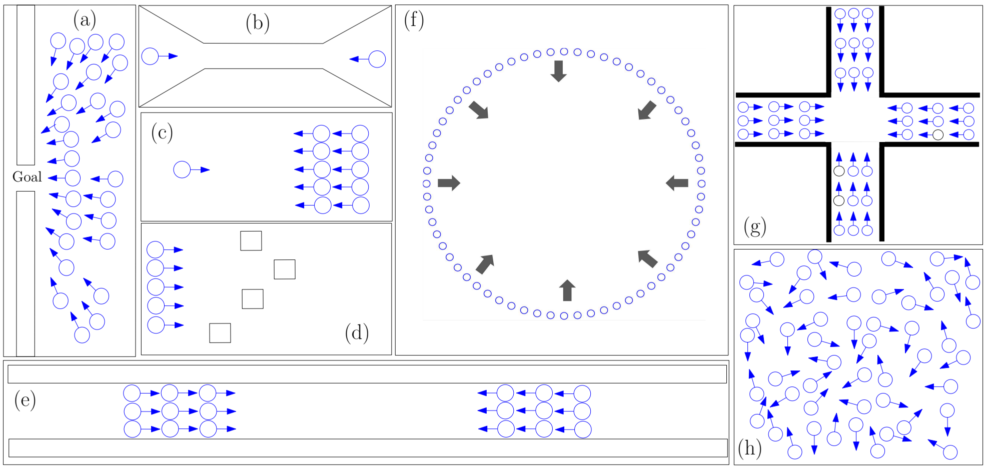


Fig. 8: Simulated scenarios:(a) Congested, (b) Deadlock, (c) Incoming, (d) Blocks, (e) Bidirectional, (f) Circle, (g) Intersection and (h) Crowd.

6.2 Performance Metric

To evaluate the performance of ALAN, we measure the time that the agents take to reach their goals compared to the upper bound of their theoretical minimum travel time. We call this metric *interaction overhead.*

Definition: Interaction Overhead. The interaction overhead is the difference between the travel time of the set of agents, as measured by Eq. 2, and the upper bound of their travel time if all the agents could follow their shortest paths to their goals at maximum speed without interacting with each other, i.e.:

$$Interaction\ Overhead = TTime(A) - MinTTime(A)$$

where $MinTTime(A)$ is the upper bound of the theoretical minimum travel time of the set of agents A, evaluated as follows:

$$MinTTime(A) = \mu\left(MinimumGoalTime(A)\right) + 3\sigma\left(MinimumGoalTime(A)\right) \quad (8)$$

where $MinimumGoalTime(A)$ is the set of travel times for all agents in A, if they could follow their shortest route to their goals, unconstrained, at maximum speed.

The interaction overhead metric allows us to evaluate the performance of ALAN from a theoretical standpoint in each of the navigation scenarios. An interaction overhead of zero represents a lower bound on the optimal travel time for the agents, and it is the best result that any optimal centralized approach could potentially achieve.

6.3 Scenarios

To evaluate ALAN we used a variety of scenarios, with different numbers of agents and, in some cases, with static obstacles. Figure 8 shows the different simulation scenarios. These include: (a) Congested: 32 agents are placed very close to the narrow exit of an open hallway and must escape the hallway through this exit (Fig. 8(a)); (b) Deadlock: Ten agents start at opposite sides of a long, narrow corridor. Only one agent can fit in the narrow space (Fig. 8(b)); (c) Incoming: A single agent interacts with a group of 15 agents moving in the opposite direction (Fig. 8(c)); (d) Blocks: Five agents must avoid a set of block-shaped obstacles to reach their goals (Fig. 8(d)); (e) Bidirectional: two groups of 9 agents each move in opposite directions inside a corridor (Fig. 8(e)); (f) Circle: 80 agents walk to their antipodal points on a circle (Fig 8(f)); (g) Intersection: 80 agents in four perpendicular streams meet in an intersection (Fig 8(g)); (h) Crowd: 400 randomly placed agents must reach their randomly assigned goal positions, while moving inside a squared room (Fig 8(h)).

6.4 Comparison of ALAN to Other Navigation Approaches

We compare the interaction overhead of ALAN with other navigation algorithms: ORCA, the Social Forces model proposed by Helbing et al. [19] (extensively used to simulate the navigation of pedestrians [18,25,17,20]), and the Predictive collision avoidance model proposed in [27]. Results can be observed in Figure 9. In most cases, ALAN outperforms the other approaches and gets agents to their goals even when the other three approaches fail to do so. In scenarios with obstacles, ALAN is able to move the agents to their goals while some (sometimes all) other evaluated approaches cannot. Here, the diversity of motions available and the behavior encouraged by the reward function in ALAN allows agents to find alternative paths to the goal while avoiding obstacles, and "get out of the way" of other agents when such paths do not exist, backtracking and allowing them to move to their goals.

In obstacle-free scenarios (Circle and Incoming), agents have more space to maneuver while moving to their goals. In the Circle scenario, the exploratory behavior of ALAN before and after congestion develops prevents it from outperforming ORCA and the Social Force models. In a smaller scenario like Incoming, the overhead of exploration does not affect ALAN as much as in Circle, allowing it to outperform both ORCA and the Social Force model. However, with the Predictive model, agents in the group make space for the single agent to move directly to its goal, reaching it faster than with ALAN.

From this evaluation, we can observe that ALAN works especially well when agents are highly constrained by both other agents and static obstacles, and its performance advantage is more moderate when agents go through long periods of unconstrained motion.

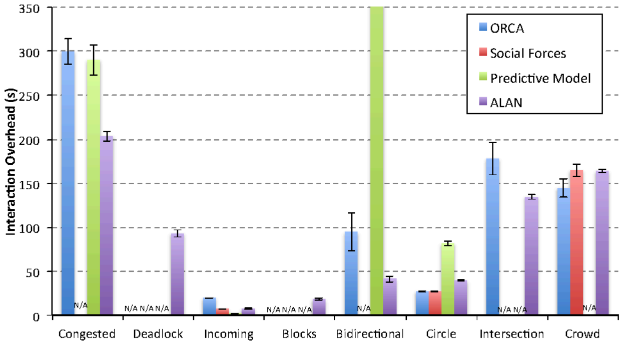


Fig. 9: Interaction overhead of ORCA, the Social Forces model, the Predictive model, and ALAN in all scenarios. N/A indicates cases where the corresponding method was unable to get agents to their goals.

6.5 Evaluation of Action Selection Method

A key component of ALAN is its Softmax inspired action selection method. Here, we validate this design choice by comparing the interaction overhead of different action selection methods, namely, ϵ-greedy [48] (with an ϵ value of 0.1) and UCB [3], within the context of ALAN. This evaluation is done using the Sample action set (Fig. 3).

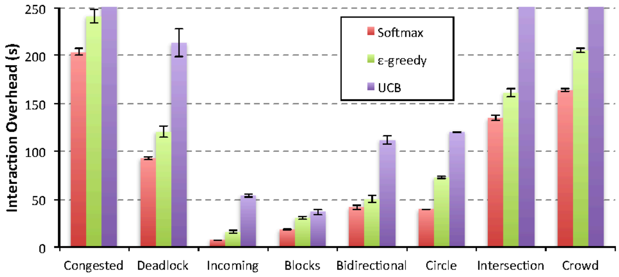


Fig. 10: Interaction overhead of ALAN, using different action selection methods (Softmax, ϵ-greedy, and UCB) with the Sample action set (Fig.3) in all scenarios.

Results (Fig. 10) indicate that the Softmax action selection helps ALAN achieve the best results. This can be explained by the combination of Softmax's probabilistic nature and its non-uniform randomized exploration. Unlike ϵ-greedy, Softmax exploration is inversely proportional to action values. Unlike UCB, the action choice is probabilistic, and it does not depend on the frequency with which each action has been chosen, which is important as that number is not necessarily related to the optimal action.

6.6 Effect of time window size

Fig. 11 shows a summary of the interaction overhead results obtained by varying the size of the time window (up to 20 secs). As the figure shows, in general, keeping the estimated values for too long or too little time hurts performance. Discarding action estimates too quickly (which turns their value into zero) makes the agent "forget" the previously chosen actions; agents do not have intuition of which actions can provide a better reward value, as all have the same probability of being chosen. On the other hand, keeping action estimates for too long perpetuates possibly outdated values, and reduces the probability of choosing an action that might have recently increased its quality. Results show that a time window of 1-5 seconds provides a good balance: it provides agents with some recent information, useful for biasing the exploration towards recently tried "good" actions and away from "bad" actions, while also preventing an outdated reward value from introducing noise in the action decision of the agent. Unless otherwise noted, we use a time window of 2 s throughout all our experiments. This corresponds to approximately 10 action decisions (as a decision is made on average every 0.2 s).

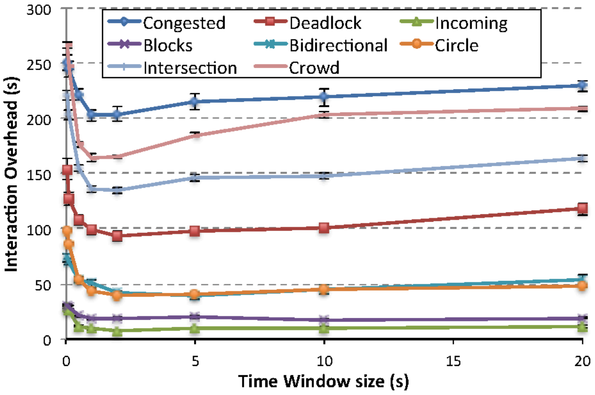


Fig. 11: Interaction overhead of ALAN in all scenarios, for different sizes of the time window used for computing the estimated value of each action.

6.7 Coordination Factor γ

The coordination factor γ controls how goal oriented or polite is the behavior of the agents in ALAN, based on the reward function (Eq. 4). Fig. 12 shows how the value of γ affects the performance of our approach. We varied the value of γ between 0 and 0.9, where γ=0 means that agents optimize their actions only based on their goal progress, while γ=0.9 implies that agents optimize their actions based mostly on their politeness, and barely take into account their goal progress. With γ=1 agents have no incentive to make progress towards their goals.

Fig. 12 shows that a high weight on the politeness component (a high value of γ) increases the interaction overhead in all scenarios. This is more noticeable with values of $\gamma > 0.6$. Here, the agents are too deferent towards each other, which ends up slowing down their progress. On the other hand, a high weight on the goal oriented component (low values of γ) seems to only have a significant negative effect on the DEADLOCK scenario, and a slight negative effect on the INTERSECTION. In the DEADLOCK scenario, maximizing the goal progress prevents agents (of one of the two groups) from quickly backtracking and clearing the way for agents in the opposite group. In this case, a balance between goal oriented and polite behavior (γ values between 0.3 and

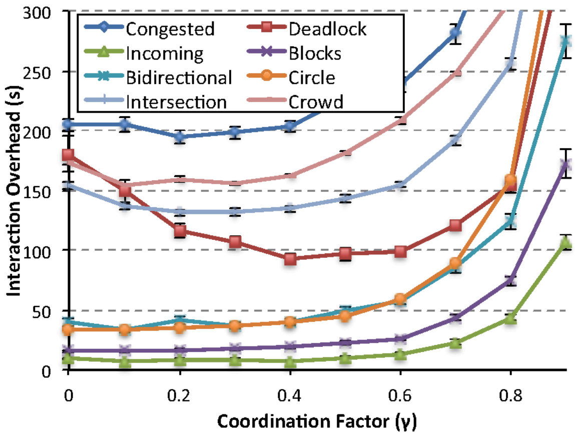


Fig. 12: Interaction overhead of ALAN in all eight scenarios, for a range of values of the coordination factor parameter γ.

0.6) allows agents to more quickly switch between both types of behavior. In other scenarios, ALAN is robust to a wide variety of γ values, minimizing the interaction overhead values when $\gamma < 0.5$. In these cases, optimizing the action selection based mostly on the goal progress allows agents to find alternative goal paths, using the open space to avoid congestion.

Overall, giving slightly more weight to the goal oriented component than the politeness component allows agents to alternate between goal oriented and polite behaviors based to their local conditions. For these reasons, we used a γ value of 0.4 in all ALAN experiments.

7 Action Space Learning

Up until now, we used the pre-defined sample set of actions shown in Fig. 3 to choose a new preferred velocity. However, depending on the environment, different sets of actions might improve the navigation. We propose an offline learning approach based on a Markov Chain Monte Carlo (MCMC) method [16,35] with simulated annealing [30] to determine, for a given environment (or a set of environments), the set of actions that minimize the travel time.

Although MCMC is typically used as a sampling method, we use it as an optimization method of sampling with a bias towards regions of better performance. We chose MCMC over other methods due the nature of our problem, as in our case the effectiveness of any subset of actions depends on the others. Consequently, greedy methods like gradient descent cannot succeed due to local minima issues. Furthermore, the bandit formulation for choosing actions within an action set does not apply because the optimization cannot be decomposed to each action. Finally, evolutionary methods only work well when better solutions to subproblems (subsets of the actions) are likely to provide a better solution to the whole problem, which is not the case in our domain.

Our method is summarized in Algorithm 2. It starts from a set composed of two actions, one action along the goal direction, the other action in a random direction. The MCMC process searches through the action space with biased exploration towards action sets that promote more time-efficient interactions. The explored action set with the highest performance is regarded as the result at the end of the process. Below we describe each step in more detail.

Algorithm 2: The MCMC action space learning

```
$Act \leftarrow \{GoalDir, RandomDir\}, Act_{opt} \leftarrow Act$
$F \leftarrow Evaluate(Act), F_{opt} \leftarrow F$
$\mathcal{T} \leftarrow \mathcal{T}_{init}, d\mathcal{T} \leftarrow (\mathcal{T}_{final} - \mathcal{T}_{init})/(N-1)$
for $i = 1$ to $N$ do
    $M \leftarrow SelectModification(Act, i)$
    $Act' \leftarrow ApplyModification(Act, M)$
    $F' \leftarrow Evaluate(Act', i)$
    if $F' < F_{opt}$ then
        $F_{opt} \leftarrow F', Act_{opt} \leftarrow Act'$
    end if
    if $Rand(0,1) < q(Act, Act')exp((F - F')/\mathcal{T})$ then
        $F \leftarrow F', Act \leftarrow Act'$
    end if
    $\mathcal{T} \leftarrow \mathcal{T} - d\mathcal{T}$
end for
return $Act_{opt}$
```

Action Set Modification. In each iteration, we perform one of the following types of modifications:

- Modify an action within an interval around its current direction, symmetric on both sides.
- Remove an action that is not the initial goal-directing one.
- Add an action within the modification interval of an existing action.

The first type of modification is explored with higher weight (i.e. performed more often), because we consider the quality of the actions to be more important than the number of actions. Following the simulated-annealing scheme, the modification range decreases over iterations as the simulation moves from global exploration to local refinement. The modification ranges are determined by short learning processes.

Action Set Evaluation. The performance of each new action set is evaluated via ALAN simulation runs. Eq. 2 is used to estimate the travel time of the set of agents. Here the set of agents is made implicit, while the action set is an explicit input to the simulation. We evaluate an action set Act with the function F, whose definition is equivalent to the definition of $TTime$ in Eq. 2 but with action set as the explicit argument rather than the set of agents. The simulation is repeated multiple times and the average evaluation from all repeated runs is used to evaluate the action set. Following the simulated annealing scheme, the number of simulation runs increases over iterations, as later local refinement has less uncertainty.

Action Set Update. We use a common version of MCMC, the Metropolis-Hasting Monte Carlo [16] scheme to reject some of the attempted modifications to efficiently explore better action sets. The probability of keeping a change is related to how it changes the evaluation F, which is the key to biasing towards action sets with lower evaluation values. The probability to accept a new action set Act' over a previous action set Act is

$$min\Big(1,\ q(Act, Act')\, exp\Big(\frac{F-F'}{\mathcal{T}}\Big)\Big), \tag{9}$$

where F and F' are the evaluation with action set Act and Act' respectively, $q(Act, Act')$ is a factor accounting for the asymmetric likelihood of attempted transitioning between Act and Act', and $\mathcal{T}$ is a parameter within the simulated-annealing scheme. The parameter $\mathcal{T}$ decreases over iterations, making the probability of accepting unfavorable changes decrease, which moves the optimization from global exploration towards local refinement.

After a predefined set of iterations of the MCMC process, the action set Act with the lowest travel time is returned. In our domain, agents have no previous knowledge of the environment, which means that they cannot determine which actions are available beforehand. However, this MCMC approach allows us to do a qualitative analysis of what behaviors are most effective in each type of environment, as we will see next.

7.1 Optimized Action Sets

Below we discuss the optimized set of actions that MCMC returned for each of the scenarios shown in Fig. 8, along with a learned action set that would work well across different scenarios, even ones not considered in the learning process.

7.1.1 Action Sets Optimized for Each Scenario

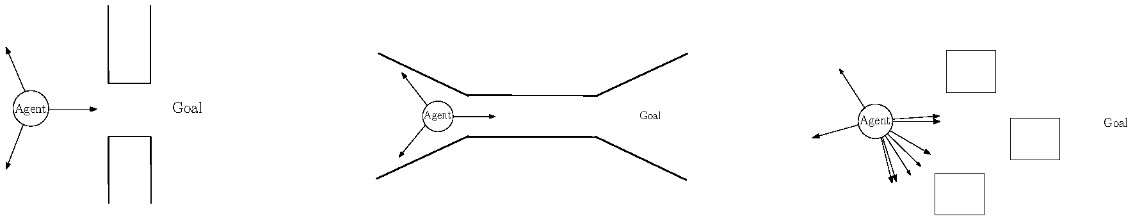
Fig. 13 and Fig. 14 show the set of actions computed by MCMC for different scenarios. As a general observation, the action sets learned for all these scenarios contain at least one action that moves the agent, to some degree, backwards from its goal. This backtracking helps in reducing congestion, allowing agents to quickly move to their goals. In the Congested, Deadlock, and Crowd scenarios, our MCMC approach found that a set of just three actions is enough to minimize the arrival time of the agents, while only two actions are needed for the Intersection. In contrast, the action set found in the Blocks scenario is larger and highly asymmetrical as compared to the previous cases. Most actions in this scenario move the agents closer to their goals, unlike the dominant backtracking motions of the previous scenarios. Similar to the Blocks scenario, in the Bidirectional scenario, a number of actions were computed by MCMC that mostly bias the motion of the agents to their right. This bias allows agents to create lanes in each side of the corridor, increasing the efficiency of their own motions and creating space for agents coming in the opposite direction.

Fig. 15 shows the optimized set of actions for the Incoming and Circle scenarios that are void of static obstacles. A common pattern found by MCMC for these environments is that the actions are heavily biased towards one of the sides of the agents. This bias, along with the absence of obstacles, allows agents to move around other agents using the available space. In the Circle scenario, for example, the optimized actions allow a vortex-shaped pattern to emerge when agents reach the center of the environment, which avoids congestion and helps the agents reach their goals faster. Note that, in both scenarios, the two sideways actions are very similar to each other. This gives agents a more fine grained control of their avoidance behavior, minimizing the detour from their goal oriented motion.

7.1.2 Multi-scenario Optimized Action Set

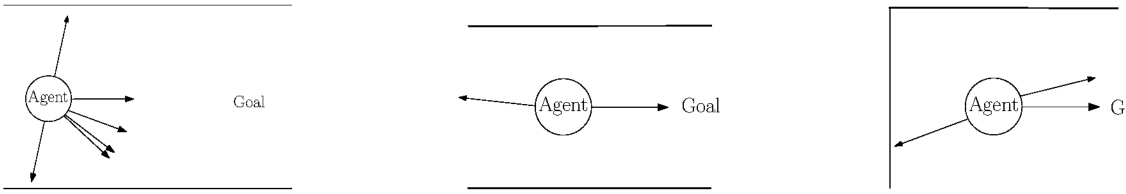
To learn a multi-scenario action set, first we trained MCMC on five scenarios, leaving out the Crowd, Bidirectional, and Intersection scenarios as test examples. We chose to leave out these scenarios because without being identical to other scenarios, they share some features with the training set: they have obstacles which constrain the motion of the agents, and also require agents to interact with each other. Then, we evaluated the resulting multi-scenario optimized action set in the entire set of eight scenarios.

The learned multi-scenario optimized action set can be seen in Fig. 16. We can observe two main features. First, there is asymmetry in the actions, which is helpful in obstacle-free environments to implicitly coordinate the motion of agents and avoid congestion. Sec-



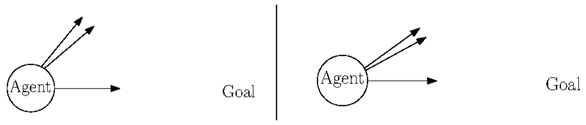
(a) Congested (b) Deadlock (c) Blocks

Fig. 13: Optimized set of actions found by the MCMC method for the (a) Congested, (b) Deadlock and (c) Blocks scenarios.



(a) Bidirectional (b) Intersection (c) Crowd

Fig. 14: Optimized set of actions found by the MCMC method for the (a) Bidirectional, (b) Intersection and (c) Crowd scenarios.



(a) Incoming (b) Circle

Fig. 15: Optimized set of actions (velocities) found by the MCMC method for the (a) Incoming and (b) Circle scenarios.

ond, half of the actions move the agents backwards from their goals, which is useful in very constrained scenarios. Again, the presence of redundant actions, both backwards as well as towards the goal, give agents better control of their behaviors.

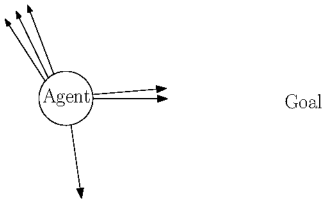


Fig. 16: Optimized set of actions (velocities) found by the MCMC method when trained on five of the eight scenarios in Fig. 8.

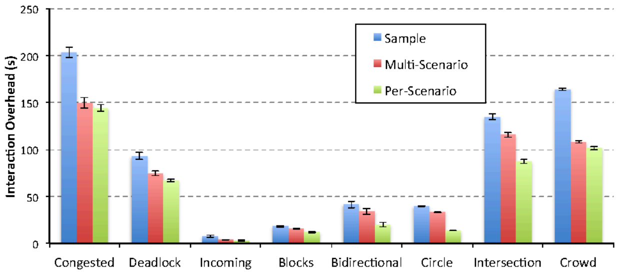


Fig. 17: Interaction overhead (s) of ALAN, using the sample action set, the multi-scenario optimized action set, and per scenario optimized action set.

7.2 Comparison of Performance between Action Sets

We compared the interaction overhead results of using ALAN with different action sets: the sample set shown in Fig. 3, and the per-scenario and multi-scenario optimized sets computed using our MCMC approach. Fig. 17 shows the corresponding results.

Overall, our MCMC approach learns optimized action sets that outperform the sample actions. When computed on a per-scenario basis, this optimized action set outperforms the sample action set in all scenarios (in all paired t-tests, $p < 0.05$). When computed using five of the eight evaluated scenarios, it still outperforms the sample set in most scenarios ($p < 0.05$) while performing similarly in the Bidirectional scenario. Note that while the Bidirectional, Intersection and Crowd scenarios were not included in the training process of the multi-scenario optimized action set, this set still outperforms the Sample action set in the Intersection and Crowd. This indicates that the multi-scenario action set generalizes well to previously unseen environments.

As expected, the interaction overhead results of the per-scenario optimized action set are better than the multi-scenario action set, with pairwise differences being statistically significant ($p < 0.05$) in most scenarios. We can observe that agents using the multi-scenario optimized action set display behaviors typically attributed to social conventions in human crowds, where pedestrians defer to others to improve the flow and avoid clogging situations. An example can be seen in the Deadlock scenario (agents backtrack to defer to incoming agents). These behaviors enable agents to reduce their travel time without the need for specific (and often unavailable) domain knowledge. Agents show human-like behaviors in the Bidirectional scenario (each group of agents avoids incoming agents moving to their right), and implicitly coordinated motion in the Circle scenario (agents form a vortex in the middle of the scenario to avoid congestion).

8 Analysis of ALAN

In this section, we analyze different aspects of ALAN, such as its runtime, how its performance scales with respect to the number of agents, as well as its robustness to failure in the actuators of the agents. We also compare the performance of ALAN with a strategy where the preferred velocity of each agent is randomized at different time intervals. Unless otherwise noted, results labeled with ALAN are obtained with the multi-scenario optimized set of actions (Fig. 16).

8.1 Runtime Complexity

During each simulation cycle, each agent performs two main operations: it first chooses a preferred velocity using its online action-selection algorithm and then maps this velocity to a collision-free one using ORCA. In practice, since the number of actions that need to be evaluated is small, selecting a new action has a negligible runtime, while ORCA dominates the overall runtime performance. Consequently, similar to ORCA, ALAN runs in $\mathcal{O}(n)$ time per agent, where n is the number of neighboring agents and obstacles used to compute the non-colliding velocity of the agent. In time units, ORCA takes approx. 1.5×10^{-5} seconds to compute a new collision-free velocity, while ALAN takes approx. 3×10^{-6} to select a new preferred velocity. In total, ALAN takes approx. 1.8×10^{-5} of processing time for each agent. This corresponds to less than a thousandth of a simulation timestep, which allows us to simulate hundreds of agents in real-time. The runtime performance reported above was obtained on an Intel i7 CPU at 3.5 GHz using a single core in the Crowd scenario.

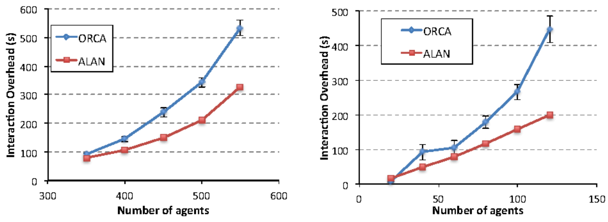
8.2 ALAN vs random velocity perturbation

Table 2 compares the interaction overhead performance of ALAN, ORCA, and a random action selection, where agents select a random action (from the Sample action set) every 1, 2 or 3 seconds. Results indicate that, in most scenarios, randomizing the selection of the preferred velocity does prevent (or solve) congestion, which results in lower travel times than ORCA in many cases, or even allows agents to reach their goals when ORCA alone cannot. However, as can be seen in the table, selecting random actions leads to higher interaction overhead values compared to ALAN, as it does not allow agents to adapt to the changes in the local navigation conditions.

With respect to the performance obtained by ALAN, we can observe that, in general, random perturbations to the preferred velocity of ORCA perform worse than ALAN, as this does not allow agents to adapt to the changes in the local navigation conditions.

8.3 Scalability

To analyze how the performance of ALAN scales with the number of agents, we varied the number of agents in the Intersection and Crowd scenarios (Fig. 8), and evaluated the interaction overhead time. Results, shown in Fig. 18, indicate that ALAN scales better than



(a) Crowd (b) Intersection

Fig. 18: Interaction overhead as a function of the number of agents in the Crowd and Intersection scenarios.

Method	CONGESTED	DEADLOCK	INCOMING	BLOCK	BIDIRECTIONAL	CIRCLE	INTERSECTION	CROWD
ORCA w/random action every 1s.	238.2	337.5	12.7	48.2	**37.7**	49.4	164	151.6
ORCA w/random action every 2s.	290.3	553.2	10.6	106.6	**34.4**	31.8	166.2	138.5
ORCA w/random action every 3s.	263.4	768.9	10.9	151.9	**36.4**	**28.6**	129.6	137.9
ORCA	299.7	N/A	19.8	N/A	94.9	**27.4**	178.2	144.6
ALAN	**149.5**	**74.4**	**3.9**	**15.7**	**33.9**	33.4	**115.6**	**107.9**

Table 2: Comparison of interaction overhead of ORCA, ALAN, and a sample action chosen randomly every few seconds. Bold numbers indicate best results, which may be more than one if there is no statistically significant difference between them.

ORCA in both scenarios. In the CROWD environment, the performance of ALAN and ORCA is similar with 350 agents, but as we increase the number of agents, the difference is more noticeable. In the INTERSECTION scenario, the difference in performance between ORCA and ALAN is noticeable starting at 40 agents, and increases as the number of agents increases.

8.4 Limitations of ALAN

Although ALAN successfully reduces the travel time of the agents compared to existing navigation approaches, it is not free of limitations. One such limitation relates to the probabilistic nature of its action selection. Specifically, there is no guarantee that agents will always choose the optimal action. Also, agents evaluate their actions based only on their past observations without considering their long-term consequences. This might prevent an agent from reaching its goal, for example, when large obstacles block its goal oriented paths.

In scenarios where the agent density is very high (approx. 1 agent per square meter), ALAN has difficulties in moving all agents to their goal locations, when they have conflicting goal paths. Note that in such high density scenarios, similar to real pedestrians, ALAN agents focus mainly on not colliding rather than on progressing to their goals. Under these settings, the maximum number of agents that ALAN can simulate, in real time, is approximately 1850. This makes ALAN usable for other multi-robot domains such as swarm robotics.

8.4.1 Applicability of ALAN to multi-robot systems

To use ALAN in multi-robot systems, some assumptions would need to be changed. Since ALAN depends on ORCA for computing collision-free velocities, it makes the same assumptions of holonomic disc-shaped robots as ORCA. Hence, ORCA would need to be adapted to account for other robot shapes. Currently, we do not assume bounds on the acceleration of the agents and do not consider rotations in the time to take an action. Robots with non-holonomic constraints would need to account for rotations and other kinematic constraints, which could be done, for example, using recent extensions to ORCA [6,11,1]. Even without bounds on the acceleration, the motions produced by ALAN look realistic in many of the scenarios.

ORCA assumes that agents can sense perfectly the positions and velocities of other agents, which is not necessarily true in robot systems. Fortunately, this problem has been tackled previously by other researchers (for example, [21], where authors deal with the problem of uncertainty in the localization of agents). Hence, we can use existing solutions to reduce the gap between simulation and real world execution of ALAN.

Imperfect Actuators. We always include a small amount of noise in the computed preferred velocities to avoid symmetry issues. This noise can reflect some level of inaccuracy of the actuators. Since in the real world actuators can fail, we evaluated the performance of ALAN when each action chosen has some probability of not being executed, because the actuators failed. Results shown in Fig. 19, indicate that ALAN is robust to failure in the actuators.

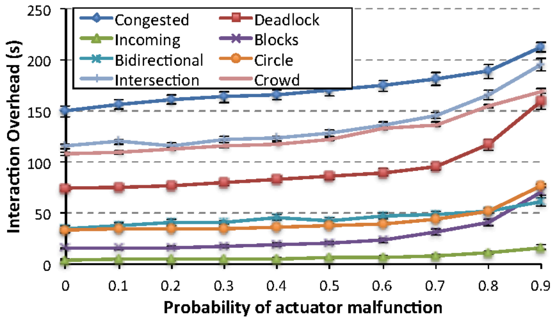


Fig. 19: Interaction overhead of ALAN in the eight scenarios shown in Fig. 8, when there is a probability actions will not be executed.

Performance degrades gracefully as the probability of actions not being executed increases. Specifically, the rate at which the interaction overhead values increase depends on the frequency of change of the locally optimal action. In the INCOMING scenario, for example, the locally optimal action for the single agent only changes a couple of times (to avoid the group and to resume goal oriented motion), hence the performance degradation is not noticeable until the probability of actuator failure is over 70%. On the other hand, in the CONGESTED scenario the performance degradation is visible at around 20% of probability of actuator failure. Overall, ALAN still performs well under these conditions.

9 Conclusions and Future Work

In this paper, we addressed the problem of computing time-efficient motions in multi-agent navigation tasks, where there is no communication or prior coordination between the agents. We proposed ALAN, an adaptive learning approach for multi-agent navigation. We formulated the multi-agent navigation problem as an action selection problem in a multi-armed bandit setting, and proposed an action selection algorithm to reduce the travel time of the agents.

ALAN uses principles of the Softmax action selection strategy and a limited time window of rewards to dynamically adapt the motion of the agents to their local conditions. We also introduced an offline Markov Chain Monte Carlo method that allows agents to learn an optimized action space in each individual environment, and in a larger set of scenarios. This enables agents to reach their goals faster than using a predefined set of actions.

Experimental results in a variety of scenarios and with different numbers of agents show that, in general, agents using ALAN make more time-efficient motions than using ORCA, the Social Forces model, and a predictive model for pedestrian navigation. ALAN's low computational complexity and completely distributed nature make it an ideal choice for multi-robot systems that have to operate in real-time, often with limited processing resources.

There are many avenues for future research. We plan to investigate the applicability of ALAN to heterogeneous environments, for example, by letting ALAN agents learn the types of the other agents present in the environment and their intended goals. This would allow an agent to more accurately account for the behavior of nearby agents during action selection. Finally, we would also like to port our approach to real robots and test it in real-world environments, such as for search and rescue operations or evacuation planning.

Acknowledgements

This work was partially funded by the University of Minnesota Informatics Institute, the CONICYT PFCHA/DOCTORADO BECAS CHILE/2009 - 72100243 and the NSF through grants #CHS-1526693, #CNS-1544887, #IIS-1748541 and #IIP1439728.

References

1. Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P., Siegwart, R.: Optimal reciprocal collision avoidance for multiple non-holonomic robots. In: Distributed Autonomous Robotic Systems, pp. 203–216. Springer (2013)
2. Audibert, J.Y., Munos, R., Szepesvári, C.: Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. Theoretical Computer Science **410**(19), 1876–1902 (2009)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3), 235–256 (2002)
4. Bayazit, O., Lien, J.M., Amato, N.: Better group behaviors in complex environments using global roadmaps. In: 8th Int'l Conf. on Artificial Life, pp. 362–370 (2003)
5. van den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. In: Proc. International Symposium of Robotics Research, pp. 3–19. Springer (2011)
6. van den Berg, J., Snape, J., Guy, S.J., Manocha, D.: Reciprocal collision avoidance with acceleration-velocity obstacles. In: IEEE International Conference on Robotics and Automation, pp. 3475–3482 (2011)
7. Buşoniu, L., Babuška, R., De Schutter, B.: A comprehensive survey of multi-agent reinforcement learning. IEEE Trans. Syst., Man, Cybern. C, Appl. Rev **38**(2), 156–172 (2008)
8. Cunningham, B., Cao, Y.: Levels of realism for cooperative multi-agent reinforcement learning. In: Advances in Swarm Intelligence, pp. 573–582. Springer (2012)
9. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using Velocity Obstacles. The Int. J. of Robotics Research **17**, 760–772 (1998)
10. Funge, J., Tu, X., Terzopoulos, D.: Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In: 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 29–38 (1999)
11. Giese, A., Latypov, D., Amato, N.M.: Reciprocally-rotating velocity obstacles. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 3234–3241 (2014)
12. Godoy, J., Karamouzas, I., Guy, S.J., Gini, M.: Adaptive learning for multi-agent navigation. In: Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1577–1585 (2015)
13. Guy, S., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., Dubey, P.: Clearpath: highly parallel collision avoidance for multi-agent simulation. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 177–187 (2009)
14. Guy, S., Kim, S., Lin, M., Manocha, D.: Simulating heterogeneous crowd behaviors using personality trait theory. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 43–52 (2011)

15. Guy, S.J., Chhugani, J., Curtis, S., Pradeep, D., Lin, M., Manocha, D.: PLEdestrians: A least-effort approach to crowd simulation. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 119–128 (2010)
16. Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**(1), 97–109 (1970)
17. Helbing, D., Buzna, L., Werner, T.: Self-organized pedestrian crowd dynamics and design solutions. Traffic Forum 12 (2003)
18. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature **407**(6803), 487–490 (2000)
19. Helbing, D., Molnar, P.: Social force model for pedestrian dynamics. Physical review E **51**(5), 4282 (1995)
20. Helbing, D., Molnar, P., Farkas, I.J., Bolay, K.: Self-organizing pedestrian movement. Environment and Planning B: Planning and Design **28**(3), 361–384 (2001)
21. Hennes, D., Claes, D., Meeussen, W., Tuyls, K.: Multi-robot collision avoidance with localization uncertainty. In: Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems, pp. 147–154 (2012)
22. Henry, P., Vollmer, C., Ferris, B., Fox, D.: Learning to navigate through crowded environments. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 981–986 (2010)
23. Hettiarachchi, S.: An evolutionary approach to swarm adaptation in dense environments. In: IEEE Int'l Conf. on Control Automation and Systems, pp. 962–966 (2010)
24. Hopcroft, J.E., Schwartz, J.T., Sharir, M.: On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". The Int. J. of Robotics Research **3**(4), 76–88 (1984)
25. Johansson, A., Helbing, D., Shukla, P.K.: Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. Advances in Complex Systems **10**, 271–288 (2007)
26. Karamouzas, I., Geraerts, R., van der Stappen, A.F.: Space-time group motion planning. In: Algorithmic Foundations of Robotics X, pp. 227–243. Springer (2013)
27. Karamouzas, I., Heil, P., van Beek, P., Overmars, M.: A predictive collision avoidance model for pedestrian simulation. In: Motion in Games, *LNCS*, vol. 5884, pp. 41–52. Springer (2009)
28. Karamouzas, I., Overmars, M.: Simulating and evaluating the local behavior of small pedestrian groups. IEEE Trans. Vis. Comput. Graphics **18**(3), 394–406 (2012)
29. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. Int. J. Robotics Research **5**(1), 90–98 (1986)
30. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., et al.: Optimization by simmulated annealing. Science **220**(4598), 671–680 (1983)
31. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
32. Kornhauser, D.M., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science (1984)
33. Macready, W.G., Wolpert, D.H.: Bandit problems and the exploration/exploitation tradeoff. IEEE Trans. Evol. Comput. **2**(1), 2–22 (1998)
34. Martinez-Gil, F., Lozano, M., Fernández, F.: Multi-agent reinforcement learning for simulating pedestrian navigation. In: Adaptive and Learning Agents, pp. 54–69. Springer (2012)
35. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. The Journal of Chemical Physics **21**(6), 1087–1092 (1953)
36. Ondřej, J., Pettré, J., Olivier, A.H., Donikian, S.: A synthetic-vision based steering approach for crowd simulation. ACM Trans. Graphics **29**(4), 123 (2010)
37. Pelechano, N., Allbeck, J., Badler, N.: Controlling individual agents in high-density crowd simulation. In: Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 99–108 (2007)
38. Pelechano, N., Allbeck, J.M., Badler, N.I.: Virtual crowds: Methods, simulation, and control. Synthesis Lectures on Computer Graphics and Animation **3**(1), 1–176 (2008)
39. Pettré, J., Ondrej, J., Olivier, A.H., Crétual, A., Donikian, S.: Experiment-based modeling, simulation and validation of interactions between virtual walkers. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 189–198 (2009)
40. Popelová, M., Bída, M., Brom, C., Gemrot, J., Tomek, J.: When a couple goes together: walk along steering. In: Motion in Games, *LNCS*, vol. 7060, pp. 278–289. Springer (2011)
41. Ratering, S., Gini, M.: Robot navigation in a known environment with unknown moving obstacles. Autonomous Robots **1**(2), 149–165 (1995)
42. Reynolds, C.: Steering behaviors for autonomous characters. In: Game Developers Conference, pp. 763–782 (1999)
43. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics **21**(4), 24–34 (1987)
44. Shao, W., Terzopoulos, D.: Autonomous pedestrians. Graphical Models **69**(5-6), 246–274 (2007)
45. Sieben, A., Schumann, J., Seyfried, A.: Collective phenomena in crowdswhere pedestrian dynamics need social psychology. PLoS one **12**(6) (2017)
46. Solovey, K., Yu, J., Zamir, O., Halperin, D.: Motion planning for unlabeled discs with optimality guarantees. In: Robotics: Science and Systems (2015)
47. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning **3**(1), 9–44 (1988)
48. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
49. Torrey, L.: Crowd simulation via multi-agent reinforcement learning. In: Proc. Artificial Intelligence and Interactive Digital Entertainment, pp. 89–94 (2010)
50. Tsai, J., Bowring, E., Marsella, S., Tambe, M.: Empirical evaluation of computational fear contagion models in crowd dispersions. Autonomous Agents and Multi-Agent Systems pp. 1–18 (2013)
51. Uther, W., Veloso, M.: Adversarial reinforcement learning. Tech. rep., Carnegie Mellon University (1997)
52. van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 1928–1935 (2008)
53. Whiteson, S., Taylor, M.E., Stone, P.: Empirical studies in action selection with reinforcement learning. Adaptive Behavior **15**(1), 33–50 (2007)
54. Yu, J., LaValle, S.M.: Planning optimal paths for multiple robots on graphs. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 3612–3617. IEEE (2013)
55. Zhang, C., Lesser, V.: Coordinated multi-agent learning for decentralized POMDPs. In: 7th Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM) at AAMAS, pp. 72–78 (2012)

56. Zhang, C., Lesser, V.: Coordinating multi-agent reinforcement learning with limited communication. In: Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1101–1108 (2013)
57. Ziebart, B.D., Ratliff, N., Gallagher, G., Mertz, C., Peterson, K., Bagnell, J.A., Hebert, M., Dey, A.K., Srinivasa, S.: Planning-based prediction for pedestrians. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 3931–3936 (2009)