Abstract—We present a decentralized path-planning algorithm for navigating multiple differential-drive robots in dense environments. In contrast to prior decentralized methods, we propose a novel congestion metric-based replanning that couples local and global planning techniques to efficiently navigate in scenarios with multiple corridors. To handle dense scenes with narrow passages, our approach computes the initial path for each agent to its assigned goal using a lattice planner. Based on neighbors’ information, each agent performs online replanning using a congestion metric that tends to reduce the collisions and improves the navigation performance. Furthermore, we use the Voronoi cells of each agent to plan the local motion as well as a corridor selection strategy to limit the congestion in narrow passages. We evaluate the performance of our approach in complex warehouse-like scenes with hundreds of agents and demonstrate improved performance and efficiency over prior methods. In addition, our approach results in a higher success rate in terms of collision-free navigation to the goals.

I. INTRODUCTION

Recent advances in AI and robotics have led to novel applications of these technologies in inventory management, delivery, and industrial automation. These applications can employ hundreds of robots [1] to transport between multiple source and destination locations. Recently, there has been considerable interest in using robots for warehouse automation. Many organizations use large-scale fulfillment centers that organize the inventory using several movable pods. The pods get transported by a team of robots to different picking stations, where each item gets picked by a human operator.

It is important to develop efficient navigation algorithms that can be deployed in such dense and complex scenarios. Multi-agent pathfinding (MAPF) is a key problem in multi-robot systems and involves computing collision-free paths for the agents to their respective goal locations. Moreover, warehouse-like scenes are regarded as complex environments with narrow passages and dynamic obstacles corresponding to other robots and possibly humans. A key issue in terms of navigation is avoiding robot-robot and robot-human collisions. Most current solutions tend to use centralized planners to compute multi-agent paths. These centralized planners [2], [3] use global knowledge of all agent states and compute a path for each agent using a composite state-space that considers all the agents simultaneously. For optimal MAPF solvers, the state-space can grow exponentially with the number of agents [4]. This global knowledge can be used to generate optimal paths and can also provide some guarantees in terms of completeness but results in increased computation cost [5]. Furthermore, these centralized methods may not work well when multiple robots get deployed in an unknown and unstructured environment, e.g., in scenarios with human co-workers or unknown dynamic obstacles. The state-of-the-art MAPF methods tend to generate piece-wise linear paths that involve “stop and turn” maneuvers and can be inefficient for non-holonomic agents [6]. Moreover, the centralized computation can create a single point of failure [7].

In decoupled approaches agents plan their individual paths to improve scalability, while still using some global information similar to centralized methods. Decoupled approaches [8], [9] use a block of memory containing the current paths of all agents, and the agents take turns replanning their paths. Sequentially planning agent paths affects completeness as one agent’s path could block another agent’s path to goal [9].

In contrast, decentralized planners [10], [11], [12] use local information to navigate robots towards their goals. They improve scalability by trading path optimality and completeness for faster computation. In these planners, agents make independent decisions to avoid the collision and reach their goals using neighbors’ state information. Since, decentralized methods[11], [12] direct the robots in the goal direction, an agent can get into local minima in the presence of static obstacles. Hence, global planners are often used to generate initial paths to the goal, and decentralized methods follow the paths while avoiding the collisions locally [13], [14]. Deadlocks are still possible as the agents could block each other’s global plans, causing over-crowding or some agents may not be able to reach the goals.

Main Results: We present a novel algorithm for multi-agent navigation in dense, congested environments with hundreds of agents. Our formulation is based on coupled global
and local planning approach for collision-free navigation. Our method is decentralized, where each agent replans its path independently based on local observation of its environment. We use lattice planning as the basis for generating smooth paths for differential drive agents by accounting for static obstacles. Our main contributions include:

- We present a novel formulation for decentralized multi-agent navigation that combines lattice-based replanning and Voronoi diagram computations.
- We propose a congestion metric to select between the alternate paths to the goal from the agent’s current location. A suitable path is chosen that is considered safe based on the congestion metric. Our metric considers path distance, potential collisions, and agent crowding while performing conflict resolution and reducing deadlocks.
- To alleviate crowding or congestion, we propose an approach to handle narrow passages by allowing each agent to reserve passageways using a shared database.

We evaluate our method in simulated environments with narrow passageways and observe significant improvement in performance over prior decentralized strategies. Our evaluation metrics include path length, success rate in reaching the goal, deadlocks, and collisions observed. In addition, our proposed method scales with the number of agents owing to their decentralized nature.

II. RELATED WORK

Multi-agent navigation is a well-studied topic in robotics involving the collision-free navigation of multiple agents towards their respective goals. Multiple methods have been proposed in this respect, and the problem is NP-hard in terms of computing optimal paths [15]. Please refer to [16] for a detailed survey. In general, multi-agent navigation methods are classified into centralized or decentralized planners.

A. Centralized Planners

Centralized planners consider a composite state-space including all the agents in the workspace [17]. The composite state-space provides global information about the workspace to compute collision-free paths to the goal. Centralized planners produce smooth, near-optimal paths that are guaranteed to be collision-free. On the other hand, they are not scalable to a large number of agents due to their large state space and centralized computations [18], [19]. Centralized methods like push and swap [20] are complete and computationally efficient but may not generate optimal paths. Further, some planners assign priority to agents to break conflicts and improve scalability, as in Cooperative A* [9] or Priority Based Search (PBS) [21]. Yu and LaValle [5] present an algorithm for generating optimal makespan solutions for multi-robot planning in challenging problems using a mapping to a multilow network.

B. Decentralized Planners

In decentralized planners, each agent independently computes a path that locally avoids collisions. Velocity obstacle(VO) [10] computes a set of collision-free velocities based on the position and velocity of the agent and its neighbors. The VO formulation was extended to consider reciprocity in RVO [22] and formulated as a linear programming problem in ORCA [11]. The formulation is extended to a variety of agent dynamics, including double integrator agents [23], differentiable drive [24], and non-holonomic agents [25]. Buffered Voronoi Cell (BVC) [12] is an efficient decentralized method that can compute collision-free trajectories for single integrator agents. In contrast to VO-based formulations, BVC only requires the agents’ position information to compute collision-free paths. RVO and BVC is used for multi-agent navigation in [26].

C. Multi-Agent Path Finding

Hybrid planners combine the advantages of centralized planners and decoupled computation. Sharon et al. [4] present Conflict Based Search (CBS), a two-level planner for optimal multi-agent pathfinding. CBS constructs a conflict tree (CT) in which each node contains a set of constraints. CBS checks for conflicts among its paths and resolves them by splitting CT nodes into child nodes. ECBS [27] enhances the scalability of CBS to a large number of agents by using a bounded sub-optimal planner. CBS-based planners are extensively used in warehouse planning. These methods consider a grid structure environment and compute a piecewise linear path. Since warehouse robots are primarily non-holonomic, they can result in inefficient maneuvers such as turning in place. Kinematic constraints are considered [3], [28] to compute smooth and efficient paths. Guo et al. [29] propose spatial and temporal splitting schemes to increase scalability of methods like ECBS on challenging maps, with negligible effects on path optimality.

Above MAPF methods plan using a centralized server, and the scalability is limited due to the centralized computation. Decoupled planners [8], [9] use token passing where a block of memory is communicated between the agents. This block of memory consists of the current planned path of each agent, and the agent currently holding the token is allowed to replan and update the memory.

Geraerts and Overmars propose a corridor map [30] method for generating smooth, short paths for multiple agents in real-time. He et al. [31] propose computing bridges with geometric properties to enable the computation of collision-free paths in environments with narrow passages. Park et al. [32] propose a cost based evaluation of agent trajectories similar to our proposed method.

Recently, game-theoretic planners have been gaining traction in the multi-agent navigation community. Spica et al. [33] present a planner for two-player autonomous drone racing that showcases competitiveness amongst agents. Mylavanam et al. [34] present a differential game approach to multi-agent collision avoidance.

In contrast to the above method, we present a decentralized conflict cost-based replanning method that uses a lattice structure and BVC to plan a smooth, collision-free path. Though our method plans on a lattice map, which is similar to grid maps used in prior MAPF methods, we resolve
TABLE I: Symbols and Notations

| Notation | Definition |
|----------|------------|
| $A_i$ | Refers to the $i^{th}$ agent |
| $t_i$ | Task $i$ which is an element of task set $T$ |
| $p_i$ | Current 2-D position of $A_i$ [$(x_i, y_i)$] |
| $s_i$ | Initial configuration of $A_i$ |
| $g_i$ | 2-D goal position of task $t_i$ |
| $v_i$ | 2-D velocity of $A_i$ [$(v_{i,x}, v_{i,y})$] |
| $R_i$ | Radius of $A_i$’s enclosing circle |
| $N_i$ | Neighbors of $A_i$ |
| $V_i, \bar{V}_i$ | Voronoi and Buffered Voronoi cell for $A_i$ |
| $\partial V_i$ | Boundary of the set $V_i$ |
| $E_{TV}$ | Environment model for lattice planner |
| $\pi(s, g)$ | Path between configurations $s$ and $g$ |

Fig. 2: Our multi-agent replanning method. Each agent received its goal position and neighbors’ positions as an input. A set of alternate paths to the goal are generated considering the possible trajectories from the current position. We compute congestion metrics for each path and each agent chooses path that minimizes the congestion metric. Each agent follows the chosen path such that its local trajectories are limited within its BVC.

Our overall approach is summarized in Algorithms 1 and 2. A representative scenario would be a narrow passageway of single-agent width. Agents entering the passageway from opposite direction could result in a deadlock. Thus, in addition to our planning method, we propose using a reservation table of the narrow passageways in the environment. A set of narrow corridors is identified and updated in a central database accessible by all agents. During execution, these passageways can be reserved for a single direction motion (Section V).

III. PROBLEM FORMULATION AND BACKGROUND

In this section, we provide an overview of our approach. Table I summarizes the symbols and notations used in the paper.

A. Problem Statement

We consider the problem of generating multi-agent paths for collision-free navigation. We consider a workspace $W \subset \mathbb{R}^2$ with $N$ homogeneous agents. In this work, we limit our approach to disk-shaped agents with differential drive dynamics. We consider a pair of agents $A_i$ and $A_j$, for $i, j \in \{1, 2, ..., N\}$ to be in collision if their bounding disks overlap, that is,

$$\|p_i - p_j\| \leq R_i + R_j.$$ 

We consider that an external task allocator assigns the goals (2-D coordinates) to the agents. In many dense scenarios, a decentralized planner can result in deadlocks as agents can block each other’s paths or the planner can get stuck in a local minimum. Moreover, we use the success rate of a multi-agent system in term of reaching the goal configuration as a suitable metric to evaluate the planner’s performance.

B. Assumptions

In our approach, we consider disk-shaped, homogeneous agents. We assume each agent $A_i$ knows its neighbors’ positions, where neighbors are agents located within a certain radius of $A_i$. This information can be obtained using perception or communication modules or using a Vicon setup. The agents’ operating environment is known, and the environment’s lattice discretization is precomputed.

Since we resolve collision using local information, agents moving in narrow passageways could result in congestion. A representative scenario would be a narrow passageway of single-agent width. Agents entering the passageway from opposite direction could result in a deadlock. Thus, in addition to our planning method, we propose using a reservation table of the narrow passageways in the environment. A set of narrow corridors is identified and updated in a central database accessible by all agents. During execution, these passageways can be reserved for a single direction motion (Section V).

C. Lattice Planning

Lattice-based motion planners sample the feasible state space of the robot in a regular fashion to construct a graph structure [35]. The graph has feasible states as its nodes and feasible motions primitives as the edges. In this paper, we consider non-holonomic constraints, and the graph $G = (V, E)$ embedded in Euclidean space presents the feasible, static, obstacle-free poses with feasible motion as the edges. We precompute a lattice graph that accounts for the static obstacles in the environment, and the agents use AD* [36]/ARA* [37] algorithm to compute paths to their respective goals.

D. Buffered Voronoi Cell (BVC)

BVC [12] is a contracted Voronoi region generated by retracting the Voronoi edges by a distance equal to the agent’s bounding radius. Disk-shaped agents remain collision-free provided their centers reside inside or on the boundary of BVC. Given $N$ agents on a 2-D plane, the buffered Voronoi cell corresponding to an agent $A_i$ is given as,

$$\bar{V}_i = \left\{ p \in \mathbb{R}^2 \left| \left( p - \frac{p_i + p_j}{2} \right)^T p_{ij} + R_i \| p_{ij} \| \leq 0, \forall j \neq i \right. \right\}.$$ 

Here, $p_{ij} = p_i - p_j$. The method plans a collision-free path for the agent $A_i$ that is constrained to lie within $\bar{V}_i$.

IV. MULTI-AGENT REPLANNING

In this section, we summarize our planning algorithm, the congestion cost computation, and our replanning approach. Our overall approach is summarized in Algorithms 1 and 2 and a high-level overview is illustrated in Figure 2 A three-agent scenario is illustrated in Figure 3.
A. Global Plan Generation

We consider a system with $N$ homogeneous agents with individual goals (for each agent) assigned through an external task allocator. We represent a task using a 2-tuple $(t_i, g_i)$. Here, $t_i$ represents the task id, and $g_i$ represents the goal configuration. Our planning algorithm utilizes a lattice structure as its basis to compute the initial global plan and the subsequent re-plan. The lattice graph for the environment is precomputed.

Each agent $A_i$ individually computes a smooth path between its source configuration $s_i$ and its goal configuration $g_i$ using the environment layout. The lattice graph structure is queried to identify a suitable path using the AD* [36] planner. Since the graph edges are precomputed motion primitives, the computed path is feasible for the agent. Moreover, the path is collision-free with the static obstacles in the environment. The computed initial global path is represented by $\pi_i(s_i, g_i)$, which represents a sequence of 3-tuples given by $(p_{i,x}, p_{i,y}, \theta_i)$ between the agent’s source $(s_i)$ and its goal $(g_i)$ configuration.

B. Collision Avoidance Using BVC and Congestion Metric Replanning

The computed global path $\pi_i(s_i, g_i)$ is not guaranteed to be collision-free as it is oblivious to inter-agent collisions or collisions with dynamic obstacles. As the agent traverses this path, its local environment changes, and collisions may be imminent due to neighboring agents. Considering the agent’s current position $p_i$, the subsequent portion of the path to goal is given by $\pi_{i}(p_i, g_i)$. We employ a cost function to intelligently choose between multiple possible paths between $p_i$ and $g_i$ to avoid collisions and congestion. We formulate this as an optimization problem to choose a desired global path, which minimizes the metric.

The main goal of the congestion metric is to perform the task of collision avoidance while moving the agent towards its goal. In this regard, the metric must account for the possibility of collision in the path, the path cost, and density/crowding in the path. A good path must have a low probability of collision, a small path cost, and should not move too close to its neighbors. Our metric accounts for these criteria using three costs, as mentioned below.

Let us assume that agent $A_i$ is at a location $p_i$. The agent at a lattice node can choose $n$ different motion primitives $(prim_k \forall k \in 1, 2, \ldots, n)$, which are the edges of the lattice graph. The agent following the primitives will reach subsequent lattice nodes represented by $p_{prim_k,i}$. Considering each $p_{prim_k,i}$ as the starting configuration, we compute $n$ paths to the goal. The $n$ paths are represented by $\pi_{prim_k,i}$ for $k \in 1, 2, \ldots, n$.

$$\pi_{prim_k,i}(p_i, g_i) \leftarrow \pi_i(p_i, p_{prim_k,i}) + \pi_i(p_{prim_k,i}, g_i)$$

The set of $n$ paths is represented by,$$
\Pi_i = \bigcup_{k=1}^{n} \pi_{prim_k,i}(p_i, g_i).
$$

We choose a suitable path from the set $\Pi_i$ such that the agent avoids collision, reduces congestion, and approaches its goal. We rank the $n$ paths based on their congestion metric, which is given by

$$C(\pi_{prim_k,i}) = k_c \ast C_{collision} + k_g \ast C_{goal} + k_n \ast C_{neighbor},$$

where $k_c$, $k_g$, and $k_n$ are positive weights for the individual costs.

- **Collision Cost ($C_{collision}$):** This captures the cost of colliding with an obstacle. Based on the neighbors’ positions, and velocity $(p_j, v_j \forall j \in N)$, we identify collisions in the current path using $\pi_i$. For each neighbor lying in the current path, we add a positive value to the cost.
- **Goal Cost ($C_{goal}$):** This computes the increase in the path length due to replanning and promotes choosing the shorter path.
- **Crowding Cost ($C_{neighbors}$):** To reduce agent crowding, we use a neighbor cost term. The neighbor cost is computed as,

$$C_{neighbors} = \begin{cases} \frac{1}{\pi_i(p_i, p_j)}, & \text{if } ||p_i - p_j|| \leq 5R_i \\ 0, & \text{otherwise} \end{cases}$$

Here, $t$ is the approximate time instant at which the agent would reach $p_i$ from its current position using $\pi_i$.

Since the agent has a finite set of motion primitives at each node, it is easy to compute the cost for each possible path. We choose a global path $\pi_i(p, g)$ that minimizes the congestion.

$$\pi_i(p_i, g_i) \leftarrow \arg \min_{\pi \in \Pi_i} C(\pi)$$

Using the congestion metric, the agent chooses a suitable path to maneuver towards the goal.

To reduce the possibility of collision, we use BVC to limit the progress of the agent along the computed paths.
Algorithm 1: Proposed Multi-Agent Navigation Method

Input: Robot State, Env;  
Output: Action;  
\( \pi_i \leftarrow \text{LatticePlanner}(\text{Env}, s_i, g_i) \);  
while \( s_i \neq g_i \) do  
\( p_{\text{new},i} = \text{nextPosition}(\pi_i) \);  
\( V_i \leftarrow \text{computeBVC}(\text{RobotPositions}) \);  
if \( p_{\text{new},i} \in V_i \) then  
\( p_i = p_{\text{new},i} \);  
end  
\( \pi_i(p_i, g_i) = \text{conflictResolution}() \);  
end

Algorithm 2: conflictResolution()

\( \Pi = \pi_i(p_i, g_i) \);  
for \( x \in \text{successor}(p_i) \) do  
\( \text{*/ successor is a node reachable from } \ 
\text{current node following a single } \ 
\text{motion primitive */} \ 
\pi_{\text{temp}} \leftarrow \text{LatticePlanner}(\text{Env}, x, g_i) \);  
\( \Pi \leftarrow \Pi \cup \pi_{\text{temp}} \);  
end  
\( \pi_i(p_i, g_i) \leftarrow \text{arg min}_{\pi \in \Pi} C(\pi) \);  
return \( \pi_i(p_i, g_i) \);

The agent constructs a buffered Voronoi cell (BVC) based on its position \( (p_i) \) and its neighbors’ positions. The BVC indicates the collision-free configurations for the agent, and if all agents continue to move within their BVCs, the agents will remain collision-free. Each agent follows its global path such that it lies within its BVC.

\[ p_i \in V_i \implies ||p_i - p_j|| > R_i + R_j, \forall j \in N_i \]

At each time step, the agent moves ahead along its path if the next position is with in the BVC. The BVC is update at the next time step based on the updated local information. The process is continued until the robot reaches its goal.

V. DEADLOCK REDUCTION AND HANDLING NARROW PASSAGES

Environments consisting of narrow passageways can result in deadlocks and longer paths, as agents may try to traverse the passage in opposite directions. To reduce such inefficiencies, we restrict any narrow passage to single direction traffic at any given time. As a preprocess, we create a database of all narrow passageways in the environment. During runtime, all agents can access this shared database to reserve passageways or to check a passageway’s reservation status.

An agent whose path traverses a passageway can reserve it for unidirectional travel on reaching a certain radius around the entry. Other agents can traverse the reserved corridor provided its traversal direction aligns with the reserved traversal direction for the passageway. That is, agents cannot enter the passageway in the opposite direction.

The initial paths for both agents pass through the top corridor. Since the agents arrive from opposite directions, we reserve the corridor dynamically in a particular direction.

Each passageway in the warehouse layout is identified and represented by a unique identifier. Let us say the layout consists of \( n \) passageways represented by \( P_i \), \( \forall i \in \{1, 2, ..., n\} \). For each passageway \( P_i \) we define a parameter \( \text{reserved}_{P_i} = (\text{status, direction, starttime, endtime}) \).

A shared database holds this block of data and is accessible by all agents.

Based on the reservation status, we have three cases.

- **Case 1: Passageway unreserved**
  Agents can select an unreserved passageway to move in a particular direction. Based on the agent’s estimated arrival time at the passageway’s entry, the passageway is reserved for a computed time duration. Since the agent knows the distance to the passageway’s entry from its global path and the average velocity, we have a lower bound estimate of the earliest arrival time. Similarly, the reservation end time is computed using a duration estimate based on the passageway length and the average agent speed.

- **Case 2: Passageway reserved in same direction**
  Other agents intending to use the passageway in the reserved direction can update the reservation end time based on an estimate of time duration required to traverse the passageway.

- **Case 3: Passageway reserved in opposite direction**
  In this case, we create temporary obstacles at the entry of the passageway and replan a global path. Thus, the agent will compute a path to its goal using a different passageway.

Following the above algorithm, we have a global path that is followed using our collision avoidance rule. The above algorithm is repeated until the agent reaches the goal.

VI. RESULTS

A. Experimental Setup

Our algorithm was implemented on an Intel Xeon octacore processor at 3.6 GHz with 32 GB memory and GeForce
Here, $DF$ is the fraction of agents stuck in a deadlock, and $CF$ is the fraction of agents that collided. Episode that have both collision and deadlocks are accounted in $CF$. Equation (1) summarizes the success rate of our proposed method in dense environments.

$$SR = 1 - DF - CF.$$ 

Here, $DF$ is the fraction of agents stuck in a deadlock, and $CF$ is the fraction of agents that collided. Episode that have both collision and deadlocks are accounted in $CF$. Table II summaries the success rate of our proposed method in dense environments.

3) Path Length: In Table II we summaries the average path length of the agents in the evaluation scenario. We tabulate the path length while the agent uses a AD* star planner alone with our collision avoidance. This provides a comparision of the increawe in path length from the individual agents path.

VII. CONCLUSION, LIMITATIONS, AND FUTURE WORK

We introduced a decentralized multi-agent path planner to navigate robots to their respective goals in dense environments. We evaluated our algorithm against the decentralized method in a set of comprehensive scenarios in terms of path length, success rate, etc. Our algorithm shows improved performance in a variety of scenarios.

Our method has a few limitations. We currently consider a homogeneous system, and all the agents use the same navigation strategy. Moreover, we require the positions and velocity of the neighboring robots to be known. The corridor selection requires a stable communication channel between the robot and the central server, and performance could decline due to communication lag. Our future work would be to generalize the algorithm for any dynamics, consider human obstacles, and test the algorithm on physical robotic systems.

REFERENCES

[1] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, “Lifelong multi-agent path finding in large-scale warehouses.”
[2] M. Liu, H. Ma, J. Li, and S. Koenig, “Task and path planning for multi-agent pickup and delivery,” in Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2019.
[3] W. Hönig, T. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayaniyan, and S. Koenig, “Multi-agent path finding with kinematic constraints,” in Twenty-Sixth International Conference on Automated Planning and Scheduling, 2016.
[4] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding,” Artificial Intelligence, vol. 219, pp. 40–66, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386
[5] J. Yu and S. M. LaValle, “Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics,” IEEE Transactions on Robotics, vol. 32, no. 5, pp. 1163–1177, 2016.
[6] L. Cohen, T. Uras, T. S. Kumar, and S. Koenig, “Optimal and bounded-suboptimal multi-agent motion planning,” in Twelfth Annual Symposium on Combinatorial Search, 2019.
[7] P. Velagapudi, K. Sycara, and P. Scerri, “Decentralized prioritized planning in large multirobot teams,” in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010, pp. 4603–4609.
[8] H. Ma, J. Li, T. Kumar, and S. Koenig, “Lifelong multi-agent path finding for online pickup and delivery tasks,” arXiv preprint arXiv:1705.10868, 2017.
[9] D. Silver, “Cooperative pathfinding,” Aide, vol. 1, pp. 117–122, 2005.
[10] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” The International Journal of Robotics Research, vol. 17, no. 7, pp. 760–772, 1998.
[11] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in Robotics research. Springer, 2011, pp. 3–19.
[12] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, “Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells,” IEEE Robotics and Automation Letters, vol. 2, no. 2, pp. 1047–1054, 2017.
[13] J. Van Den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin, “Interactive navigation of multiple agents in crowded environments,” in Proceedings of the 2008 symposium on Interactive 3D graphics and games, 2008, pp. 139–147.
[14] S. H. Arul, A. J. Sathyamoorthy, S. Patel, M. Otte, H. Xu, M. C. Lin, and D. Manocha, “Lswarm: Efficient collision avoidance for large swarms with coverage constraints in complex urban scenes,” IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3940–3947, 2019.
[15] H. Ma, S. Koenig, N. Ayaniyan, L. Cohen, W. Hönig, T. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon, “Overview: Generalizations of multi-agent path finding to real-world scenarios,” arXiv preprint arXiv:1702.05315, 2017.

1http://sbpl.net/Home
TABLE II: Evaluation Scenarios

| Environment                  | Agents | AD* Planner | Proposed Method |
|------------------------------|--------|-------------|-----------------|
|                              |        | Agents Collided | Path Length | Agents Deadlocked | Agents Collided | Success Rate | Path Length |
| Environment 1 (Vertical Agent Spacing 4m) | 5  | 4 | 37.77 | 0 | 0 | 1 | 42.50 |
|                              | 10 | 9 | 37.20 | 0 | 0 | 1 | 41.13 |
|                              | 20 | 20 | 36.95 | 0 | 0 | 1 | 41.96 |
|                              | 30 | 30 | 38.13 | 0 | 0 | 1 | 43.99 |
| Environment 2 (Vertical Agent Spacing 2m) | 20 | 19 | 36.99 | 0 | 0 | 1 | 41.03 |
|                              | 40 | 40 | 37.69 | 0 | 2 | 0.95 | 44.67 |
|                              | 60 | 60 | 37.36 | 0 | 4 | 0.93 | 45.36 |
| Environment 2 (Vertical Agent Spacing 4m) | 5  | 0 | 41.82 | 0 | 0 | 1 | 42.71 |
|                              | 10 | 6 | 40.04 | 0 | 0 | 1 | 41.97 |
|                              | 20 | 20 | 40.19 | 0 | 0 | 1 | 44.34 |
| Environment 2 (Vertical Agent Spacing 2m) | 20 | 19 | 40.07 | 0 | 0 | 1 | 43.13 |
|                              | 40 | 40 | 40.16 | 0 | 0 | 1 | 46.57 |

Fig. 7: Agent paths for 20 agents in a simulated warehouse environment.