Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Enhancing Lifelong Multi-Agent Path-finding by Using Artificial Potential Fields

Arseni Pertzovskiy, Roni Stern, Ariel Felner, Roie Zivan

Ben-Gurion University of the Negev

Abstract

We investigate the use of Artificial Potential Fields (APFs) for improving online multi-agent pathfinding in lifelong settings, where agents are continuously assigned new goals upon completing previous tasks. We present APF-enhanced variants of several online coordination methods, including prioritized planning, large-neighborhood search, and priority-inheritance–based schemes. Our results show that integrating APFs into these algorithms substantially increases overall system throughput, achieving up to a 7-fold improvement in task completion rate in densely populated environments.

Introduction

Artificial Potential Fields (APFs) (Khatib 1986) are a physics-inspired method for guiding mobile agents in environments with obstacles. Agents are modeled as particles influenced by attractive and repulsive forces: goals exert attraction, while obstacles and nearby agents exert repulsion (Koren and Borenstein 1991). APFs have been used in motion planning and collision avoidance (see below).

We study how APFs can improve coordination efficiency in lifelong multi-agent pathfinding (LMAPF), where agents operate continuously and receive a new goal immediately after finishing the previous one (Švancara et al. 2019; Li et al. 2021a). In LMAPF, agents repeatedly traverse shared regions of the environment, and congestion accumulates over time. Our objective is to mitigate congestion by making agents proactively avoid high-traffic regions through APF-based repulsion.

APF can be applied directly, where each agent greedily follows local forces. However, this is insufficient for long-horizon planning in dense environments. Therefore, to exploit APFs effectively in an online setting, we embed them within existing real-time coordination schemes such as prioritized single-agent planning, large-neighborhood search, and priority-inheritance–based approaches. We modify the cost model of each planner so that future congestion contributes to action cost, encouraging agents to select paths that reduce long-term interference with others. These APF-augmented variants retain the structure and computational efficiency of their underlying planners while gaining an online mechanism for congestion avoidance. Experiments on standard lifelong benchmarks show that APFs provide substantial benefit in the online setting, achieving up to a 7-fold increase in completed tasks.

Definition and Background

A *multi-agent pathfinding* problem (MAPF) is defined by a tuple $\langle k, G, s, g\rangle$ where k is the number of agents, $G = (V, E)$ is an undirected graph, $s : [1, ..., k] \rightarrow V$ and $g : [1, ..., k] \rightarrow V$ map an agent to a start and to a goal vertex, respectively. Time is discretized. At each time step, an agent should perform either a *move* action from its current location to an adjacent location or *wait*, i.e., stay idle in its current location. A *single-agent path* for agent a_i, denoted π_i, is a sequence of actions that start from s_i and end in g_i. A solution to a MAPF is a set of single-agent paths $\pi = \{\pi_1, \ldots, \pi_k\}$, one per agent, that do not *conflict*, i.e, do not occupy the same vertex or cross the same edge in opposite directions at the same time.

Lifelong MAPF (LMAPF) (Li et al. 2021a) is a version of *online MAPF* (Švancara et al. 2019). When an agent reaches its current goal, it receives a new goal to travel to from a task assigned module (outside the path-planning system). Since solving LMAPF involves solving multiple MAPF problems, the efficiency of LMAPF algorithms is commonly measured by the overall system *throughput* achieved, which is measured by the number of tasks completed in a given period of time (Li et al. 2021b; Morag, Stern, and Felner 2023).

The Rolling-Horizon Collision Resolution framework (RHCR) (Li et al. 2021b), solves LMAPF problems by repeatedly planning the next k steps (k is the *horizon* parameter) to execute based on the current locations and goals of the agents. The agents then move $w \leq k$ steps, where w is a *window* parameter, and the process repeats.

Prioritized Planning (PrP) and LNS2 In PrP (Bennewitz, Burgard, and Thrun 2001; Leet, Li, and Koenig 2022; Varambally, Li, and Koenig 2022; Zhang et al. 2022; Chan et al. 2023). The agents plan sequentially according to some predefined order. When the i^{th} agent plans, it must not conflict with the paths chosen for all $i-1$ agents that planned before it. Such single-agent paths are found by a low-level search algorithm such as Temporal A* (TA*) or SIPPS (see below). PrP is simple and fast, but may not be very effective in very dense environments due to possible deadlocks. Large

Neighborhood Search for MAPF (LNS2) (Li et al. 2022) is an incomplete MAPF algorithm that overcomes some of the pitfalls of PrP. LNS2 starts by assigning paths to the agents even if they conflict. LNS2 then applies a *repair* procedure, where PrP is used to replan for a subset of agents, aiming to minimize conflicts with other agents. LNS2 repeats this repair procedure until the resulting solution is conflict-free.

Temporal A*, SIPP, and SIPPS The Temporal A*, SIPP, and SIPPS algorithms plan a path for a single agent under constraints. They are used in many MAPF algorithms, such as CBS (Sharon et al. 2015), PBS (Ma et al. 2019), PrP, and LNS2. Temporal A* (TA*) uses A* on the *spatio-temporal state-space* where a state is a vertex-time pair (v, t). TA* receives the start and goal locations of an agent, along with a list of vertex and edge constraints. A vertex (edge) constraint is a pair (v, t) (e, t) specifying that the agent must not plan to be at v (cross e) at time-step t. TA* returns the shortest path that avoids the given constraints. Safe Interval Path-planning (SIPP) (Phillips and Likhachev 2011) improves TA* by performing an A* search on a state space where each state is defined by a vertex and a safe (time) interval, representing that the agent occupies that vertex in this time interval and that this does not violate any given constraint. Safe Interval Path-planning with Soft Constraints (SIPPS) (Li et al. 2022) generalizes SIPP to accept both *hard* and *soft* constraints. SIPPS returns a path that does not violate any hard constraints and minimizes the number of soft constraints violated. SIPPS is designed to run within LNS2.

PIBT and LaCAM PIBT (Okumura et al. 2022) is a MAPF algorithm that searches in the *configuration space*. A configuration is an assignment of the agents to locations at a time-step. PIBT iteratively generates a configuration for the next time-step until reaching a goal configuration. LaCAM (Okumura 2023) is a two-level algorithm. At its high level, it searches the configuration space in-depth-first search manner. At its low level, it chooses the next successor, usually by applying PIBT on the current node. LaCAM* (Okumura 2023) continues the search in an anytime fashion and eventually converges to the optimal solution.

Usage of APFs APFs were used for robot motion planning in continuous spaces (Zhang, Lin, and Chen 2018; Shin and Kim 2021; Vadakkepat, Tan, and Ming-Liang 2000). Hagelbäck et al. (Hagelbäck and Johansson 2008; Hagelback and Johansson 2009) used APFs in real-time games to avoid obstacles. Liu et al. (Liu, Ge, and Goh 2017) used APFs in multi-agent formation; Dinh et al. (Dinh, van Lon, and Holvoet 2016) introduced Delegate MAS that simulated food foraging behavior in ant colonies; and many works (Semnani et al. 2020; Fan et al. 2020; Agrawal et al. 2022; Dergachev and Yakovlev 2021) introduced reinforcement learning algorithms (RL) that incorporated AFPs together with Optimal Reciprocal Collision Avoidance (ORCA) (Van den Berg et al. 2011) or Force-based motion planning (FMP) (Semnani et al. 2020). To the best of our knowledge, we are the first to apply APFs to solve LMAPF.

Integrating APFs in TA* and SIPPS

We propose to use APFs within the TA* and SIPPS single-agent path-finding algorithms, such that the resulting path avoids collisions with the paths of other agents but also attempts to keep a distance from them by taking into consideration the repulsion of their APFs. We refer to our TA* and SIPPS variants as *Temporal A* with APFs* (TA*+APF) and *SIPPS with APFs* (SIPPS+APF), respectively.

Temporal A* with APFs

TA*+APF a tuple $\langle G(V, E), s, g, \{\pi_1, \ldots, \pi_{k'}\}\rangle$ where $G(V, E)$ is the underlying graph s and g are the start and goal locations of the path planning agent; and π_i is a path for agent a_i for every $i = 1, \ldots, k'$. The output of TA*+APF is a path from s to g that does not conflict with any of the paths $\pi_1, \ldots, \pi_{k'}$. In PrP TA*+APF will be given the paths planned for the higher-priority agents. In LNS2, the given set of paths includes the paths already planned for the other agents within the neighborhood of the planning agent.

To keep distance from these paths, TA*+APF creates for every path $\pi_i \in \{\pi_1, \ldots, \pi_{k'}\}$ a repulsion APF function APF_i that maps every location-time pair (v, t) to a real number representing the added penalty incurred by planning to occupy v at time t. TA*+APF considers these penalties when computing the cost of every move. The APF induced by agent a_i on location v and time t is computed as follows:

$$APF_i(v,t) = \begin{cases} 0 & \text{if } d(v, \pi_i[t]) \geq d_{max} \\ w \cdot \gamma^{-d(v,\pi_i[t])} & \text{otherwise} \end{cases} \quad (1)$$

where d_{max}, γ, and w are predefined parameters and $d(v, t, \pi_i[t])$ is the minimal distance between v and $\pi_i[t]$ (we used Manhattan distance). In each time-step t', an agent considers only APFs that are calculated at t'. In the special case of $d_{max} = 0$, $APF_i(v, t)$ is always 0. This corresponds to plain TA*, which only considers conflicts where agents are exactly at the same location (distance 0 from each other). To aggregate all the APFs, we used a simple sum. That is, the APF cost of moving into location v at time is

$$APF(v,t) = \sum_{i \in \{1,\ldots,k'\}} APF_i(v,t) \quad (2)$$

$APF(v, t)$ is a function of a vertex and a time step. We considered incorporating $APF(v, t)$ within TA* in two ways.

(1) h-inspired APF: similarly to h-values, we calculate $APF(v, t)$ only for the last node on the path (i.e., vertex v, the current location of the agent) and add it as a penalty to the f-value of a node n and use:

$$f(n) = g(n) + h(n) + APF(v,t) \quad (3)$$

where v is the last vertex of the path associated with n.

(2) g-inspired APF: We calculate $APF(v', t)$ for every vertex v' on the path and aggregate these costs along the path (similarly to g-values) until we reach the current vertex v. Therefore, in TA*+APF we aggregate $APF(v', t)$ when a node is generated, as follows:

$$agg_{APF}(n) = agg_{APF}(p) + APF(v,t) \quad (4)$$

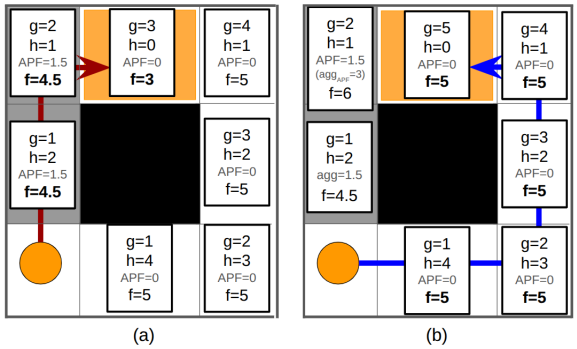

Figure 1: The orange circle and orange square are the agent's start and goal. The gray cells are APFs of cost 1.5.

where p is the parent of n, and (v, t) is the vertex and time-step of node n. Initially, for the start node $agg_{APF}(start) = APF(start, t_0)$ (it has no parent). TA*+APF runs TA* and expands nodes according to

$$f(n) = g(n) + h(n) + agg_{APF}(n) \tag{5}$$

The h-inspired APF was ineffective. $h(n)$ is designed to quickly guide the search towards the goal. Incorporating APFs in a similar way doesn't affect what path TA* selected but rather how quickly it is found. By contrast, $g(n)$ represents the cost of the best path found so far to n. Thus, aggregating APFs along the path, similarly to g-values, directly results in TA* returning paths that avoid other agents' paths.

In Figure 1 an agent needs to go from the orange circle to the orange square. The gray zone is the APFs of another agent of cost 1.5, which affects the two cells at the middle- and top-left. Fig. 1(a) shows the h-inspired APF while Fig. 1(b) shows the the g-inspired APF. The blue path always has $f = 5$. When the search expands the start cell, it will generate the middle-left cell with $f = g + h + APF = 4.5$. So, it will be expanded, and the top-left cell is generated. That cell too has an APF of 1.5. Now, if the h-inspired APF is used, then its cost will also be $f = g + h + APF = 4.5$, it will be expanded, and the goal will be reached with $g(goal) = 3$. However, if the g-inspired APF is used, then the APFs are aggregated, and the cost of the top-left cell will be $f = g + h + agg_{APF} = 2 + 1 + 3 = 6$. In that case, the search will proceed with the blue path and reach the goal with $g(goal) = 5$, avoiding the congested gray cells.

Using APFs incurs some computational overhead, compared to vanilla TA*. This overhead is due to the need to compute APFs for every newly generated path. The computational complexity of generating an APF for the path of a single agent is $O((d_{max})^2 \cdot l)$, where $(d_{max})^2$ is the maximal number of nodes affected by the APF induced by an agent occupying a single vertex at a specific time step; and l is the length of the longest path. This computation is done for each of the k agents, adding a total overhead of $O(k \cdot (d_{max})^2 \cdot l)$.

SIPPS with APFs

SIPPS+APF is based on SIPPS. Thus, a search node n in SIPPS+APF represents a vertex $n.v$ and a time interval $[t^n_{start}, t^n_{end})$, and the input to SIPPS+APF includes a set of *soft constraints* it aims to minimize violating. We define an APF cost for a SIPPS+APF node n as follows:

$$APF(n) = \max_{t \in [t^n_{start}, t^n_{end})} \sum_{i \in \{1,\dots,k'\}} APF_i(v, t) \tag{6}$$

where $APF_i(v, t)$ is defined in TA*+APF. In other words, $APF(n)$ represents the highest APFs an agent can encounter during its time interval. In SIPPS, the open list is sorted first according to the number of soft collisions violated and then according to $f(n)$. SIPPS+APF follows the same sorting rule, but adds the APF costs in each of these components. In more detail, SIPPS+APF maintains for each node n two additional values, $c_{APF}(n)$ and $g_{APF}(n)$, which are computed as follows when n is generated:

$$c_{APF}(n) = c_{APF}(p) + c(p, n) + APF(n) \tag{7}$$

$$g_{APF}(n) = t^n_{start} + APF(n) \tag{8}$$

where p is the parent of n, $c(p, n)$ is the number of soft constraints violated when moving from p to n. SIPPS+APF sorts the open list first based on c_{APF} and then based on $g_{APF} + h(n)$. The analysis of the runtime complexity overhead incurred by APFs in SIPPS+APF is similar to the analysis described for TA*+APF. An additional computational cost is incurred due to the maximization of the time steps in the relevant safe interval (Eq. 6). So, the overhead equals $O(k \cdot d^2_{max} \cdot l^2)$, where k, d_{max}, and l are defined as earlier.

Completeness of TA*+APF and SIPPS+APF

We note that our new f-function (with the APF) is only considered when computing the priority of nodes in the open list. However, every node in TA*+APF and in SIPPS+APF still maintains the original spatio-temporal information as in regular TA* and SIPPS. In particular, the original g-value, which adds up the physical steps is still maintained and is used to determine the exact time step for identifying conflicts as well as to prune duplicates. Thus, the search spaces are the same with and without APFs, and the only difference is the order in which nodes are expanded. Consequently, completeness is preserved for both algorithms.

PIBT and LaCAM with APFs

Both PIBT (Okumura et al. 2022) and LaCAM (Okumura 2023) use a heuristic to prioritize the actions of agents when generating configurations. In these algorithms, each agent chooses its next action by sorting the vertices adjacent to it based on a heuristic estimate of their distance from the goal. We propose to add APFs to these heuristic estimates and refer to our PIBT variant as *PIBT with APFs* (PIBT+APF). Specifically, when agent i in PIBT+APF picks the next location, it continues to calculate several steps forward, defined by the t_{max} parameter. Those steps follow agent i's optimal path to its goal and ignore other agents. Then, PIBT+APF builds APFs around these steps with Equation 1 and sums up all the values in the time dimension.

$$PIBT\_APF_i(v) = \sum_{j \in \{t_{curr},\dots,t_{curr}+t_{max}\}} APF_i(v, j) \tag{9}$$

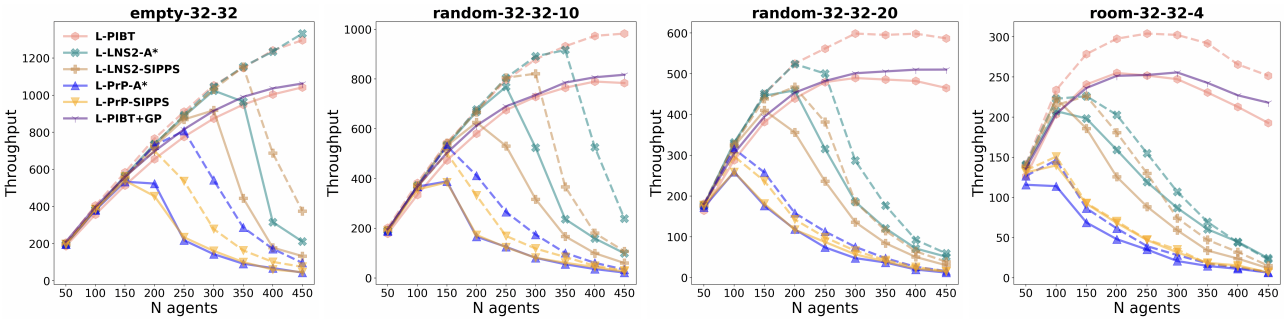

Figure 2: LMAPF: Average Throughput. Dashed lines - APF-enhanced; Solid lines - no APFs

where, t_{curr} defines the current time step. As a result, an APF is created for an agent i. Finally, when every agent sorts the vertices adjacent to its current location, it considers the APFs created by the previously planned agents and sums all the APFs for every neighboring vertex as follows:

$$APF(v) = \sum_{i \in \{1,\ldots,k'-1\}} PIBT\_APF_i(v) \qquad (10)$$

where $PIBT\_APF_i(v)$ is defined above (Eq. 9). Finally, we sort the vertices according to $h(v)+APF(v)$. The overhead incurred by PIBT_APFs in terms of run-time complexity is $O(k \cdot d_{max}^2 \cdot t_{max})$ per time-step, corresponding to computing APF for all k agents and nodes in radius d_{max}. The total overhead is $O(k \cdot d_{max}^2 \cdot t_{max} \cdot l)$, where l is the length of the longest path. Our APF-enhanced version of the LaCAM algorithm uses PIBT+APF for its low-level search. This approach of adapting APFs to PIBT, in part, is inspired by the work of Chen et al. (Chen et al. 2024) in which the authors use congestion-avoiding paths to navigate agents. Our work can be viewed as the generalization of this approach, where we consider not only the future steps of agents, but also the surrounding locations of these steps. We use their method as a baseline in our experiments.

Experimental Study

We empirically compared the use of APFs within PrP (Silver 2005), LNS2 (Li et al. 2022), PIBT (Okumura et al. 2022), LaCAM (Okumura 2023), and LaCAM* (Okumura 2023), where PrP and LNS2 are implemented once with TA* and once with SIPPS. Versions that use APFs are denoted in our plots by dashed lines. As an additional baseline, we implemented *PIBT with Guide Path* (L-PIBT+GP) from Chen et al. (Chen et al. 2024). All experiments were carried out on four different maps from theLMAPF benchmark (Stern et al. 2019): *empty-32-32*, *random-32-32-10*, *random-32-32-20* and *room-32-32-4*. The number of agents ranged from 50 to 450. We executed 15 random instances for every number of agents, map, and algorithm. The APF parameters used were $w = 1$, $d_{max} = 4$, and $\gamma = 2$ for TA*+APF, $w = 0.1$, $d_{max} = 3$, and $\gamma = 3$ for SIPPS+APF, and $w = 0.1$, $d_{max} = 2$, $\gamma = 3$, and $t_{max} = 2$ for PIBT+APF, which were observed to work best in general across all grids. All algorithms were implemented in Python and ran on a MacBook Air with an Apple M1 chip and 8GB of RAM.

We solved LMAPF problems using the RHCR framework with the parameters *window* and planning *horizon* set to 5. The planning phase for each algorithm was limited to 10 seconds. In planning-failure events, we followed Morag et al.'s (Morag, Stern, and Felner 2023) robustLMAPF framework using the *AllAgents* + *iStay* + *Persist* configuration. This corresponds to having all agents plan in every planning period (*AllAgents*); failing agents stay in their place (*iStay*); and all agents that do have a path and can follow it without conflicts, do so (*Persist*). In the planning phase, our experiment was halted after each agent performed 100 steps. The main metric in the LMAPF experiments is the average *throughput* of each algorithm, which is the common metric for measuring their quality (Li et al. 2021b; Song, Na, and Yu 2023; Morag, Stern, and Felner 2023).

Figure 2 plots the average throughput of the different algorithms as a function of the # of agents. Solid lines represent the original algorithms, and dashed lines represent our APF-enhanced versions. The results for LaCAM and LACAM* with and without APFs were virtually the same as for PIBT due to the fact that they both use PIBT as their low-level search algorithm. Therefore, we only plot the results of PIBT. The results demonstrate that using APFs significantly increases the throughput of all other algorithms on all maps. The significant advantage observed in using TA*+APF within the RHCR framework may suggest that it indeed biases agents towards paths that avoid the paths of other agents, reducing future collisions and congestion. In most cases, PIBT+APF outperforms other approaches, including L-PIBT+GP, and improves the original PIBT by a significant margin. As an interesting side phenomenon, we observed the superior performance of LNS2 algorithms using A*, compared to LNS2 with SIPPS. This could be due to the fact that A* prioritizes short paths and SIPPS prioritizes paths with minimum soft constraints.

Conclusion and Future Work

We proposed to use APFs in TA* and in SIPPS, which are key components of many LMAPF algorithms. The resulting algorithms add bias to the search to avoid passing near the paths of other agents. Experimentally, we showed that the introduction of APFs in the context of lifelong MAPF (when using APFs in TA*, SIPPS, and PIBT) is highly beneficial, resulting in significantly higher overall system throughput. Future work will study how to effectively set the parameters d_{max}, γ, t_{max}, and w for APF in TA*, SIPPS, and PIBT. Additionally, APFs can be integrated for other algorithms.

Acknowledgments

This research was partly supported by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Initiative and by the Marcus Endowment Fund, both at Ben-Gurion University of the Negev; by the Binational Science Foundation (BSF) fund #2022189 awarded to Roie Zivan; the Israel Science Foundation (ISF) grant #1238/23 awarded to Roni Stern; #909/23 to Ariel Felner; and by the Ministry of Science grant No. 0008612/1001948158 to Ariel Felner and Roni Stern.

References

Agrawal, A.; Hariharan, S.; Bedi, A. S.; and Manocha, D. 2022. DC-MRTA: Decentralized Multi-Robot Task Allocation and Navigation in Complex Environments. In *IROS*, 11711–11718. IEEE.

Bennewitz, M.; Burgard, W.; and Thrun, S. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In *ICRA*, volume 1, 271–276.

Chan, S.-H.; Stern, R.; Felner, A.; and Koenig, S. 2023. Greedy Priority-Based Search for Suboptimal Multi-Agent Path Finding. In *SoCS*, 11–19.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic flow optimisation for lifelong multi-agent path finding. In *AAAI*, volume 38, 20674–20682.

Dergachev, S.; and Yakovlev, K. 2021. Distributed multi-agent navigation based on reciprocal collision avoidance and locally confined multi-agent path finding. In *CASE*.

Dinh, H. T.; van Lon, R.; and Holvoet, T. 2016. Multi-agent route planning using delegate MAS. In *Workshop on Distributed and Multi-Agent Planning*, 24–32. London, UK.

Fan, T.; Long, P.; Liu, W.; and Pan, J. 2020. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *IJRR*, 39(7): 856–892.

Hagelback, J.; and Johansson, S. 2009. A multi-agent potential field-based bot for a full RTS game scenario. In *AAAI*, volume 5, 28–33.

Hagelbäck, J.; and Johansson, S. J. 2008. Using multi-agent potential fields in real-time strategy games. In *AAMAS*, 631–638.

Khatib, O. 1986. The potential field approach and operational space formulation in robot control. In *Adaptive and Learning Systems: Theory and Applications*, 367–377. Springer.

Koren, Y.; and Borenstein, J. 1991. Potential field methods and their inherent limitations for mobile robot navigation. In *ICRA*, 1398–1404.

Leet, C.; Li, J.; and Koenig, S. 2022. Shard systems: Scalable, robust and persistent multi-agent path finding with performance guarantees. In *AAAI*, 9386–9395.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *IJCAI*.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *AAAI*.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. *AAAI*.

Liu, X.; Ge, S. S.; and Goh, C.-H. 2017. Formation potential field for trajectory tracking control of multi-agents in constrained space. *International Journal of Control*, 90(10): 2137–2151.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with consistent prioritization for multi-agent path finding. In *AAAI*, 7643–7650.

Morag, J.; Stern, R.; and Felner, A. 2023. Adapting to Planning Failures in Lifelong Multi-Agent Path Finding. In *SoCS*.

Okumura, K. 2023. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *AAAI*, volume 37, 11655–11662.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *AI*, 310: 103752.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *ICRA*, 5628–5635. IEEE.

Semnani, S. H.; Liu, H.; Everett, M.; De Ruiter, A.; and How, J. P. 2020. Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning. *IEEE RA-L*, 5(2): 3221–3226.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *AI*.

Shin, Y.; and Kim, E. 2021. Hybrid path planning using positioning risk and artificial potential fields. *Aerospace Science and Technology*, 112.

Silver, D. 2005. Cooperative Pathfinding. In *AIIDE*.

Song, S.; Na, K.-I.; and Yu, W. 2023. Anytime Lifelong Multi-Agent Pathfinding in Topological Maps. *IEEE Access*, 11: 20365–20380.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–158.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI*.

Vadakkepat, P.; Tan, K. C.; and Ming-Liang, W. 2000. Evolutionary artificial potential fields and their application in real time robot path planning. In *Congress on Evolutionary Computation*.

Van den Berg, J.; Guy, S. J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-Body Collision Avoidance. In *Robotics Research*, 3–19.

Varambally, S.; Li, J.; and Koenig, S. 2022. Which MAPF Model Works Best for Automated Warehousing? In *SoCS*.

Zhang, H.-y.; Lin, W.-m.; and Chen, A.-x. 2018. Path planning for the mobile robot: A review. *Symmetry*, 10(10): 450.

Zhang, S.; Li, J.; Huang, T.; Koenig, S.; and Dilkina, B. 2022. Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding. In *SoCS*.