Adaptive Neighborhood Selection for MAPF via Non-Stationary Bandits

AMATH SOW*, DANIEL DE LENG, MARIUSZ WZOREK, and FREDRIK HEINTZ, Linköping University, Sweden

Adaptive Large Neighborhood Search (ALNS) is a meta-heuristic framework widely used to solve large-scale MAPF problems. It uses selection mechanisms such as the Roulette Wheel or Multi-Armed Bandit (MAB) policies, which adaptively learn to identify the most effective neighborhood (heuristic) at each iteration. A key assumption of these policies is that each heuristic's reward distribution (effectiveness) is stationary. However, the best heuristic can change during the search process. In this work, we address the inherent non-stationarity of neighborhood selection in ALNS-based MAPF solvers, where the reward distribution of the heuristic changes over time. We introduce DyMAB (Dynamic multi-armed bandit for neighborhood selection), a novel anytime algorithm that integrates a non-stationary multi-armed bandit (MAB) framework into ALNS. DyMAB leverages a windowed queue to track recent rewards, allowing for dynamic adaptation of neighborhood selection at each iteration. This ensures that the most effective heuristic is prioritized, improving overall search efficiency. We present a theoretical analysis showing that DyMAB achieves sub-linear regret under standard non-stationary bandit assumptions, offering formal performance guarantees. We benchmark DyMAB against a range of state-of-the-art anytime solvers across diverse MAPF scenarios, demonstrating that our method significantly enhances solution quality and scalability. In most cases, DyMAB achieves up to a 50% reduction in AUC and up to a 30% decrease in the Sum of Delays, highlighting its superiority in optimizing large-scale MAPF instances.

JAIR Track: Multi-Agent Path Finding

JAIR Associate Editor: Daniel Harabor

JAIR Reference Format:
Amath Sow, Daniel de Leng, Mariusz Wzorek, and Fredrik Heintz. 2026. Adaptive Neighborhood Selection for MAPF via Non-Stationary Bandits. *Journal of Artificial Intelligence Research* 86, Article 39 (July 2026), 27 pages. DOI: 10.1613/jair.1.19324

1 Introduction

Multi-Agent Path Finding (MAPF) (Stern et al. 2021) is a problem where multiple agents navigate in a shared environment from their start locations to goal locations without collisions. Different solvers have been proposed with varying levels of success, broadly classified as *optimal* and *near-optimal* approaches (Felner et al. 2017). Optimal solvers, such as CBS (Sharon et al. 2015) and Improved CBS (ICBS) (Boyarski et al. 2015), guarantee that a least-cost solution will be found using a two-level search strategy. Still, their scalability is limited to small instances of problems due to exponential time complexity. To address scalability, near-optimal solvers trade-off solution quality for faster computations, offering practical alternatives for large and complex environments (Barer et al. 2014; Li, Ruml, et al. 2021). Among near-optimal solvers, anytime algorithms have become very popular for their focus on finding near-optimal solutions within a given time budget. MAPF-LNS (Li, Chen, et al. 2021),

*Corresponding Author.

Authors' Contact Information: Amath Sow, ORCID: 0009-0006-5681-9797, amath.sow@liu.se; Daniel de Leng, ORCID: 0000-0001-6356-045X, daniel.de.leng@liu.se; Mariusz Wzorek, ORCID: 0000-0003-2147-2114, mariusz.wzorek@liu.se; Fredrik Heintz, ORCID: 0000-0002-9595-2471, fredrik.heintz@liu.se, Linköping University, Linköping, Östergötland, Sweden.

DOI: 10.1613/jair.1.19324

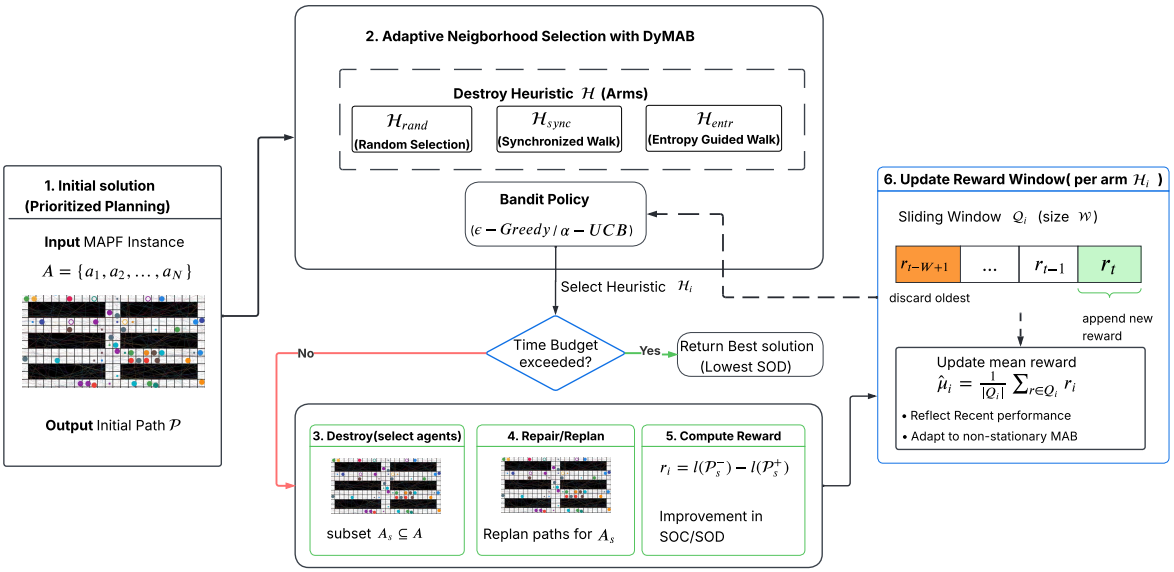

Fig. 1. Overview of the DyMAB algorithm architecture. An initial sub-optimal solution is generated using Prioritized Planning. The adaptive selection mechanism operates over a set of distinct destroy heuristics (Arms): $\mathcal{H}_{rand}$, $\mathcal{H}_{sync}$, and $\mathcal{H}_{entr}$. At each iteration, a non-stationary bandit policy (ϵ-Greedy or α-UCB) selects a heuristic to destroy a subset of agents, followed by a repair operator to replan their paths. The resulting improvement in the solution cost yields a reward $r_i = l(\mathcal{P}_i^-) - l(\mathcal{P}_i^+)$. This reward is appended to a sliding window Q_i of size W, discarding the oldest entries to update the mean reward estimate $\hat{\mu}_i$. This dynamic feedback loop allows the policy to continuously adapt to non-stationary heuristic performance by prioritizing recent successes. The search loop continues until the time budget is exhausted, returning the solution with the lowest Sum of Delays (SOD).

a state-of-the-art anytime solver, uses the Large Neighborhood Search (LNS) meta-heuristic technique from combinatorial optimization (Shaw 1998; Windras Mara et al. 2022). This method iteratively refines an initial sub-optimal solution by employing destroy heuristics to remove paths of selected agents and repair operators to replan these paths, aiming to minimize the overall cost. ALNS extends LNS by maintaining a set of heuristics. Instead of using a fixed strategy, ALNS aims to adaptively select the most promising heuristic at each iteration based on their observed performance. Techniques like Roulette Wheel (Li, Chen, et al. 2021) or Multi-Armed Bandit (MAB) (Phan, Huang, et al. 2024) have been employed for this adaptive selection mechanism.

However, a fundamental assumption underlying these adaptive selection mechanisms is the stationarity of the reward distribution associated with each heuristic. This assumption means that the effectiveness of a particular heuristic remains constant during the search. In the context of MAPF, this may not accurately reflect the dynamic nature of the solution space. More specifically, different heuristics target different aspects of the solution space. For example, some focus on the most delayed agents, others on collision graphs, and some target dense environmental bottlenecks. As the solution improves and conflicts are resolved, the performance of these heuristics can shift significantly. Consequently, a heuristic that was effective early in the search may become less performant later on. Therefore, relying on historical performance under a stationarity assumption may lead to suboptimal heuristic selection and hinder the overall search efficiency.

To address this limitation, we introduce DyMAB (Dynamic Multi-Armed Bandit for Neighborhood Selection), a novel anytime MAPF solver specifically designed to address the inherent non-stationarity of heuristic performance within the ALNS framework. The DyMAB architecture is illustrated in Fig 1. The key contributions of our work are 1) two novel destroy heuristics specifically designed for ALNS-based solvers; 2) two novel MAB policies, DyMAB(α-UCB) and DyMAB(ϵ-Greedy), explicitly designed for non-stationary MAB; and 3) a theoretical analysis of the DyMAB framework, establishing dynamic regret bounds and optimal parameter conditions under piecewise-stationary reward assumptions.

The remainder of this paper is organized as follows. An overview of related work is presented in Section 2. Section 3 introduces the MAPF problem and its relation to non-stationary multi-armed bandits. Section 4 introduces our DyMAB approach. Section 5 presents theoretical analysis of DyMAB. An experimental evaluation showing the performance of DyMAB relative to the state-of-the-art is presented in Section 6. The paper ends with conclusions and a discussion of future work in Section 7.

2 Related Work

Anytime MAPF solvers are a subset of sub-optimal solvers that focus on finding near-optimal solutions within a time budget, enabling a trade-off between time and solution quality. One of the leading approaches in this category is MAPF-LNS (Li, Chen, et al. 2021), which has been applied effectively to large-scale MAPF problems. It uses a sub-optimal solver to generate an initial solution (Erdmann and Lozano-Perez 1986; Sajid et al. 2012) and then iteratively improves the solution. This is done by removing paths for a subset of agents (destroy heuristic), replanning these paths (repair operator), and selecting the solution with the lowest cost. The process continues until the time budget is exhausted. Adaptive LNS (ALNS) further enhances this approach by using a set of competing heuristics based on their historical performance. These destroy heuristics are a random uniform selection of N paths, an agent-based heuristic (H_{agent}), and a map-based heuristic (H_{grid}) (Li, Chen, et al. 2021). The agent-based heuristic H_{agent} focuses on the path of the agent with the maximum delay and other paths that affect it. The map-based heuristic H_{grid} randomly chooses a vertex with a high degree in the graph, which includes paths through the selected vertex. Then, a selection policy π such as a Roulette Wheel is used to choose between the set of heuristics H by maintaining updatable weights w (Shaw 1998). Recent advancements include Machine Learning-Guided LNS (ML-LNS) (Huang et al. 2022), which uses imitation learning to guide the selection of agent paths for replanning, and MAPF-LNS2 (Li, Chen, et al. 2022), which integrates SIPPS algorithm which is an improved version of SIPP (Phillips and Likhachev 2011) for faster single-agent path-finding, significantly speeding up the solution process and allowing for efficient solving of large scenarios with thousands of agents.

A notable recent algorithm is BALANCE (Bandit-based Adaptive Large Neighborhood Search Combined with Exploration) (Phan, Huang, et al. 2024). BALANCE uses a bi-level MAB scheme to adapt the selection of destroy heuristics and neighborhood sizes. It uses policies based on Upper Confidence Bound (UCB1), and Thompson Sampling that outperform previous MAPF-LNS solvers. However, these policies assume that the performance of the heuristics is stationary. In this paper, we demonstrate that removing this assumption enables more effective neighborhood selection, significantly improving solution costs and search efficiency even in large-scale scenarios with more than thousand agents. ADDRESS (Adaptive Destroy and Repair with Dynamic Exploration Strategy) (Phan, Zhang, et al. 2025) is another recent anytime LNS-based solver. It applies Thompson Sampling to the top-K most delayed agents to select a seed agent for neighborhood generation, focusing the search on the agents that contribute most to the overall solution cost. Another approach is LaCAM (Okumura 2023b), a scalable and complete MAPF solver that searches over agent configurations while lazily adding constraints to resolve conflicts. LaCAM* (Okumura 2023a) extends this into an anytime solver by iteratively refining an initial solution, although the first solutions can be of low quality and convergence to optimality may be slow in large-scale scenarios.

3 Background

3.1 Problem Formalization

We consider a MAPF problem (Stern et al. 2021), which is modeled as an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. The problem involves N agents, denoted as $A = \{a_1, a_2, \ldots, a_N\}$. Each agent $a_i \in A$ starts at vertex $s_i \in V$ and aims to reach a goal vertex $g_i \in V$ by executing a sequence of actions over discrete time-steps. At each time-step, an agent can move to an adjacent vertex or wait at its current vertex. A path $\mathcal{P}_i$ for agent a_i is a sequence of vertices: $\mathcal{P}_i = [p_{i_0}, p_{i_1}, \ldots, p_{i_{l(\mathcal{P}_i)}}]$, where $p_{i_0} = s_i$, $p_{i_{l(\mathcal{P}_i)}} = g_i$, and $l(\mathcal{P}_i)$ is the length or travel distance of the path (Li, Chen, et al. 2021). We consider two types of conflicts. A Vertex conflict $\langle a_i, a_j, v, t \rangle$ occurs when two agents a_i and a_j attempt to occupy the same vertex v at the same time t and an Edge conflict $\langle a_i, a_j, u, v, t \rangle$ occurs when the agents attempt to traverse the same edge $(u, v) \in E$ in opposite directions during the same time t (Stern 2019).

A valid solution is a set of collision-free paths $\mathcal{P} = \{\mathcal{P}_i \mid a_i \in A\}$. The objective is to find a solution $\mathcal{P}$ that minimizes the Sum of Costs (SOC), i.e., the total travel distance of all agents, defined as:

$$\arg\min_{\mathcal{P}} \sum_{\mathcal{P}_i \in \mathcal{P}} l(\mathcal{P}_i). \tag{1}$$

In the Anytime MAPF framework, the goal is equivalently expressed as minimizing the Sum of Delays (Li, Chen, et al. 2021)(SOD):

$$\arg\min_{\mathcal{P}} \sum_{\mathcal{P}_i \in \mathcal{P}} delays(\mathcal{P}_i), \tag{2}$$

where the delay of an agent a_i's path is computed as:

$$delays(\mathcal{P}_i) = l(\mathcal{P}_i) - d(s_i, g_i), \tag{3}$$

with $d(s_i, g_i)$ being the shortest path distance (in terms of graph edges) between the start vertex s_i and the goal vertex g_i.

Another important metric for evaluating the performance of anytime solvers is the Area Under the Curve (AUC), which represents the cumulative Sum of Delays. It is defined as:

$$AUC(t_{\text{limit}}) = \int_{t_{\text{init}}}^{t_{\text{limit}}} SOD(t)\, dt, \tag{4}$$

where a lower AUC indicates faster improvement in solution quality (Huang et al. 2022).

Another standard metric in MAPF is the Makespan, defined as $\max_i\ l(\mathcal{P}_i)$, i.e., the time at which the last agent reaches its goal. In this work, we focus on SOD/AUC minimization, which is the standard objective for LNS-based solvers (Li, Chen, et al. 2021, 2022).

3.2 Non-stationary Multi-Armed Bandits

The Multi-Armed Bandit (MAB) problem is a classic reinforcement learning framework that models the fundamental trade-off between exploration and exploitation. Inspired by a gambler facing multiple slot machines (historically known as "one-armed bandits"), the agent must decide which machine to play to maximize their total payout while learning which machine has the highest expected return. Formally, a K-armed bandit is a sequential decision problem in which, at each round $t \in \{1, \ldots, T\}$, an agent selects an arm $k_t \in \{1, \ldots, K\}$ and receives a reward $r_{k_t}(t)$ drawn from a distribution with an unknown mean μ_{k_t}. The goal is to maximize the cumulative reward $\sum_{t=1}^{T} r_{k_t}(t)$, or equivalently to minimize the cumulative regret $R_T = \sum_{t=1}^{T}(\mu^* - \mu_{k_t})$, where $\mu^* = \max_k \mu_k$.

In ALNS-based MAPF solvers, the algorithm's performance critically depends on selecting appropriate destroy heuristics at each iteration. Given multiple competing heuristics (each with unknown and evolving performance),

how can we decide which one to apply at each step to maximize overall solution improvement? This naturally aligns with the MAB framework, where each heuristic corresponds to an arm and pulling an arm yields a reward representing the improvement in solution cost. While classical MAB methods have been applied to this setting, they assume stationary reward distributions, which fail to capture the evolving nature of heuristic performance during the search.

The non-stationary MAB problem extends the classic MAB formulation by allowing reward distributions to change over time. At each time step t, the agent selects an arm k from a set of K arms and receives a reward $r_k(t)$, drawn from a time-varying distribution with a mean of $\mu_k(t)$ (Besbes et al. 2014; Liu et al. 2013). The objective is to maximize the cumulative reward, $\sum_{t=1}^{T} r_k(t)$, over a time horizon T. This setup introduces a unique challenge: balancing exploration (trying new or less certain arms) and exploitation (choosing arms known to yield high rewards) while adapting to temporal variability in reward distributions. Several approaches have been proposed to address non-stationarity. Sliding-Window UCB (SW-UCB) (Garivier and Moulines 2008) adapts traditional UCB methods by considering only the rewards obtained within a fixed sliding window. This allows the algorithm to prioritize recent rewards, making it responsive to changes in reward distributions. In contrast, Discounted UCB (D-UCB) (Allesiardo et al. 2017) uses a discount factor $\gamma \in (0, 1]$ to reduce the weight of older rewards exponentially. This method retains a memory of past observations but gradually diminishes their influence, increasing sensitivity to recent changes. Similar strategies have been applied to Thompson Sampling (Qi et al. 2023; Trovo et al. 2020).

4 DyMAB for MAPF

To enable adaptive heuristic selection within the ALNS framework for MAPF, we propose DyMAB, a MAB approach that accounts for the non-stationary performance of heuristics during search. Specifically, we introduce two non-stationary MAB policies tailored for this setting: DyMAB(α-UCB) and DyMAB(ϵ-Greedy). These policies maintain a sliding reward window queue Q of size W for each heuristic to prioritize recent performance and apply a *logarithmic decay* to the exploration parameters α and ϵ, encouraging early exploration followed by gradual exploitation of the most effective heuristics. Alternative formulations could be considered; for instance, modeling the full interaction as an MDP, or using a contextual bandit that conditions selection on features of the current solution state. We leave that as future work. In this section, we detail the neighborhood selection strategies and the DyMAB policies used to guide their application.

4.1 Neighborhood Selection

Efficient exploration of the solution space in ALNS relies on well-defined heuristics. This section introduces three distinct heuristics, each tailored to explore the solution space.

4.1.1 Random Neighborhood. For the *random neighborhood heuristic* H_{rand}, we randomly select uniformly a subset of agents A_s given a neighborhood size M. It is a naive approach and has shown interesting results, particularly for congested environments (Demir et al. 2012; Li, Chen, et al. 2021, 2022; Song et al. 2020).

4.1.2 Synchronized Random Walk Neighborhood. The *Synchronized random walk neighborhood* H_{sync} is inspired by the agent-based neighborhood approach presented in the MAPF-LNS framework (Li, Chen, et al. 2021). However, unlike the original approach, which selects only the most delayed agent, H_{sync} generalizes this by defining a set of seed agents S based on a delay metric D and then simultaneously performing random walk. This method balances between the most delayed agents and those blocking their progress, which can lead to more effective path repair and better overall solutions.

The corresponding pseudo-code is presented in Algorithm 1. The algorithm begins by selecting a set of seed agents S (line 2), sampled from the top-k most delayed agents that are not currently in the tabu list $\mathcal{T}$, where k is

randomly drawn from the interval $[2, \lfloor M/2 \rfloor]$ (line 1). These agents form the initial neighborhood $\mathcal{N}$ (line 3) and are added to the tabu list to avoid repeated selection.

Next, a synchronized random walk is performed. Each seed agent $a_i \in S$ is initialized at the start of its path, with x_i representing its current location and $t_i = 0$ as the starting timestep (line 5). The upper bound ub_i is set to the length of its current path minus one, restricting how far in timesteps the agent is allowed to walk (line 6). Variables `steps` (tracking the number of walk iterations) and `no_progress` (counting steps taken without expanding the neighborhood $\mathcal{N}$) are initialized alongside the maximum step limit (line 7).

At the beginning of the main loop, the algorithm stores the current neighborhood size (line 9) to detect subsequent progress. Then, for each seed agent a_i (line 10), it identifies a set of feasible next locations N_{x_i} (line 11). This set consists of all neighboring vertices (including the current location) that allow the agent to still reach its goal within its originally planned path length. This check ensures that the random walk does not unnecessarily extend the agent's path beyond its upper bound, enforced through the constraint $t_i + 1 + h(x) < l(\mathcal{P}_i)$, where $h(x)$ is a heuristic estimate of the remaining cost to the goal.

While feasible locations remain, a random location $y \in N_{x_i}$ is selected for agent a_i (line 13). Any agents conflicting with a_i at location y and time $t_i + 1$ are added to the neighborhood $\mathcal{N}$ via set union (line 14), automatically discarding duplicates. If $|\mathcal{N}|$ exceeds M after this addition, it is immediately truncated to retain exactly M agents (line 15–16). The agent's state (x_i, t_i) is then updated to the new position y and the next timestep $t_i + 1$ (line 17).

After processing all seed agents, the algorithm checks whether the neighborhood size has increased compared to the previous iteration. If $|\mathcal{N}| >$ `prev`, the counter `no_progress` is reset to zero, indicating that progress has been made. Otherwise, `no_progress` is incremented to track stagnation (lines 19–22). If no progress has been made for $\lfloor$`max`$/4\rfloor$ consecutive steps, each seed agent is randomly reset to a different location along its current path (lines 23–26). This re-randomization serves to reinitialize the synchronized walk and helps the algorithm escape potential local stagnation. The global step counter `steps` is incremented (line 27). Once the main loop terminates—either because the neighborhood $\mathcal{N}$ has reached the target size M or the step limit has been exceeded—the constructed neighborhood $\mathcal{N}$ is returned (line 28).

4.1.3 Entropy-based Neighborhood. The *Entropy-based neighborhood heuristic* H_{entr} leverages Shannon information theory, commonly applied in Vehicle Routing Problems (VRP) (Baranwal et al. 2022; Ma 2011), to identify agents with high path entropy. In our case, path entropy quantifies the variability of an agent's movements, considering both the frequency of location visits in its precomputed path and the likelihood of an agent visiting a particular location at each timestep. The key intuition is to prioritize agents with highly complex paths, as these are often found in congested areas where conflicts are more likely. Resolving such conflicts can significantly enhance the overall solution in MAPF.

Definition 1 (Path entropy). *Let $\mathcal{P}_i = [l_0, l_1, \ldots, l_k]$ represent the sequence of locations (vertices) in agent a_i's path, and $freq(l)$ denote the frequency of location $l \in V$ in the path, $l_0 = p_{i_0}$, $l_1 = p_{i_1}$ and $l_k = p_{i_{l(\mathcal{P}_i)}}$. The path entropy of agent a_i, denoted as $H(\mathcal{P}_i)$, is calculated using Shannon entropy:*

$$H(\mathcal{P}_i) = -\sum_{l \in \mathcal{P}_i} p(l) \log_2 p(l), \tag{5}$$

where $p(l) = freq(l)/|\mathcal{P}_i|$ is the probability of visiting location l. We define an entropy score $E(a_i)$ for each agent as:

$$E(a_i) = H(\mathcal{P}_i) \cdot (1 + D(a_i)), \tag{6}$$

where $D(a_i) = l(\mathcal{P}_i) - d(s_i, g_i)$ is the delay factor for agent a_i, as defined in Section 3.

Algorithm 1: Synchronized Random Walk Neighborhood

Input: Graph $G = (V, E)$, agents $A = \{a_1, \ldots, a_m\}$, paths $P = \{p_1, \ldots, p_m\}$, neighborhood size M, tabu list $\mathcal{T}$

Output: Neighborhood $\mathcal{N} \subseteq A$

/* Step 1: Seed agent selection */

$k \leftarrow \textsc{RandInt}(2, \lfloor M/2 \rfloor)$;

$S \leftarrow$ Top-k delayed agents from $A \setminus \mathcal{T}$;

$\mathcal{N} \leftarrow S$; $\mathcal{T} \leftarrow \mathcal{T} \cup S$;

/* Step 2: Synchronized random walk */

foreach $a_i \in S$ **do**

  $(x_i, t_i) \leftarrow (p_{i_0}.\text{loc}, 0)$;

  $ub_i \leftarrow l(\mathcal{P}_i) - 1$;

$steps \leftarrow 0$; $max \leftarrow |A|$; $no\_progress \leftarrow 0$;

while $steps < max \wedge |\mathcal{N}| < M$ **do**

  $prev \leftarrow |\mathcal{N}|$;

  foreach $a_i \in S$ **do**

    $N_{x_i} \leftarrow \{x \in \text{neighbors}(x_i) \cup \{x_i\} \mid t_i + 1 + h(x) < ub_i\}$;

    while $N_{x_i} \neq \emptyset \wedge |\mathcal{N}| < M$ **do**

      $y \leftarrow$ random $x \in N_{x_i}$;

      $\mathcal{N} \leftarrow \mathcal{N} \cup \textsc{ConflictingAgents}(a_i, x_i, y, t_i + 1)$;

      if $|\mathcal{N}| > M$ **then**

        $\mathcal{N} \leftarrow \mathcal{N}[:M]$;

      $(x_i, t_i) \leftarrow (y, t_i + 1)$;

      break;

  if $|\mathcal{N}| > prev$ **then**

    $no\_progress \leftarrow 0$;

  else

    $no\_progress \leftarrow no\_progress + 1$;

    if $no\_progress \geq \lfloor max/4 \rfloor$ **then**

      foreach $a_i \in S$ ***with*** $ub_i > 1$ **do**

        $t_i \leftarrow \textsc{RandInt}(0, ub_i)$;

        $x_i \leftarrow p_{i_{t_i}}.\text{loc}$;

  $steps \leftarrow steps + 1$;

return $\mathcal{N}$;

To select the seed agent, we use Boltzmann Sampling, a probabilistic technique that ensures diverse selection while favoring agents with higher entropy scores. The probability of selecting agent a_i is computed as:

$$p_b(a_i) = \frac{\exp\left(\frac{E(a_i)}{T}\right)}{\sum_{a \in A} \exp\left(\frac{E(a)}{T}\right)}, \tag{7}$$

where T is the temperature parameter controlling the randomness of selection. Higher T values promote exploration by increasing the likelihood of selecting agents with lower scores. The seed agent a_i is chosen such that:

$$r \leq \sum_{j=1}^{i} p_b(a_j), \tag{8}$$

where $r \sim U[0,1]$ is a uniform random value.

After selecting the seed agent, an entropy-guided walk is performed to expand the neighborhood $\mathcal{N}$. At each timestep t, for every potential next location l, a score $E(l)$ is computed as:

$$E(l) = \frac{H(l,t)}{1 + \mathrm{h}(l)}, \tag{9}$$

where $H(l,t)$ is the location entropy at (l,t).

Definition 2 (Location entropy). *The location entropy is calculated as*

$$H(l,t) = -\sum_{a_i \in A} p(a_i) \log_2 p(a_i), \tag{10}$$

where

$$p(a_i) = \frac{freq(a_i,l,t)}{\sum_{a_j \in A} freq(a_j,l,t)} \tag{11}$$

represents the likelihood of agent a_i visiting location l at timestep t, and $freq(a_i,l,t) = \mathbf{1}[p_{i_t} = l]$ is the indicator function over all *agents $a_i \in A$ using their current solution paths. Thus $H(l,t)$ captures the congestion at location l at time t across the entire current solution; vertices visited by no agent have zero entropy.*

The corresponding pseudo-code is presented in Algorithm 2. The process begins by computing the path entropy for each agent $a_i \in A \setminus \mathcal{T}$, where $\mathcal{T}$ is the tabu list and their entropy score $E(a_i)$ (lines 1–3). Then, a Boltzmann distribution is computed over the entropy scores and normalized. Each agent's probability of being selected is proportional to the exponential of its entropy score, scaled by a temperature parameter T that controls the degree of exploration(lines 4–11). A single seed agent is then sampled according to this distribution. This is done by generating a uniform random number $r \in [0,1]$ and selecting the first agent a_i such that the cumulative probability up to a_i exceeds r (lines 12–18). The selected seed agent is then added to the neighborhood $\mathcal{N}$ (line 19), and an entropy-guided walk is performed to expand the neighborhood (line 20), after which the resulting neighborhood $\mathcal{N}$ is returned (line 21).

The entropy-guided walk is detailed in Algorithm 3. The process begins by initializing the timestep $t = 0$ and setting the agent's current location *loc* to the start of its planned path (lines 1–2). The algorithm then enters a main loop that continues as long as the agent has not reached the end of its path ($t < |\mathcal{P}_i| - 1$) and the neighborhood has not yet reached the target size M (line 3).

At each iteration, the agent identifies all valid next locations *next_locs*, which includes its adjacent vertices and its current location to account for wait actions (line 4). To evaluate these candidates, an empty list *location_scores* is initialized (line 5). For each candidate location $l \in$ *next_locs*, the algorithm computes a probability distribution $p(a_j)$ representing the likelihood of any agent visiting l at timestep $t+1$ based on the current solution paths (line 7). This is used to calculate the location entropy $H(l, t+1)$, which quantifies the congestion or collision risk at that specific vertex and time (line 8). A final score $E(l)$ is then computed by scaling the location entropy inversely with the heuristic cost to the goal $h(l)$, ensuring that the walk is drawn toward congested areas without straying too far from a goal-oriented trajectory (line 9). This score is appended to *location_scores* (line 10).

Once all candidate locations are scored, a *moved* flag is initialized to false (line 11). The candidates are sorted in descending order based on their entropy scores $E(l)$, and the agent iterates through them to find the best valid

Algorithm 2: Entropy-Based Neighborhood

Input: Agents $A = \{a_1, \ldots, a_m\}$, Paths $P = \{p_1, \ldots, p_m\}$, Neighborhood size M, Tabu list $\mathcal{T}$, Temperature T
Output: Neighborhood $\mathcal{N}$

/* Step 1: Compute agent path entropies */
foreach $a_i \in A \setminus \mathcal{T}$ **do**
  $H(\mathcal{P}_i) \leftarrow -\sum_{l \in \mathcal{P}_i} \frac{\text{freq}(l)}{|\mathcal{P}_i|} \log_2 \left(\frac{\text{freq}(l)}{|\mathcal{P}_i|} \right)$;
  $E(a_i) \leftarrow H(\mathcal{P}_i) \cdot (1 + D(a_i))$;
/* Step 2: Boltzmann probabilities */
probabilities $\leftarrow [\,]$;
$Z \leftarrow 0$;
foreach $E(a_i)$ **do**
  $s_i \leftarrow \exp(E(a_i)/T)$;
  Append s_i to *probabilities*;
  $Z \leftarrow Z + s_i$;
for $i \leftarrow 1$ **to** $|probabilities|$ **do**
  $probabilities[i] \leftarrow probabilities[i]/Z$;
/* Step 3: Select seed agent */
$r \leftarrow \text{Uniform}(0, 1)$;
cumulative_prob $\leftarrow 0$;
for $i \leftarrow 1$ **to** $|probabilities|$ **do**
  $cumulative\_prob \leftarrow cumulative\_prob + probabilities[i]$;
  if $r \leq cumulative\_prob$ **then**
    seed_agent $\leftarrow i$;
    break;
/* Step 4: Entropy-guided walk */
$\mathcal{N} \leftarrow \{\text{seed\_agent}\}$;
$\mathcal{N} \leftarrow \text{LocEntropyRandomWalk}(\text{seed\_agent}, P, \mathcal{N}, M)$;
return $\mathcal{N}$;

move (line 12). For the highest-scoring candidate l, a temporal feasibility constraint $t + 1 + h(l) < |\mathcal{P}_i|$ is checked (line 13). This ensures that moving to l still leaves enough time for the agent to theoretically reach its goal within its original path length. If the constraint is satisfied, any agents conflicting with this move are identified and added to the neighborhood $\mathcal{N}$ (line 14).

Following a successful move selection, the agent updates its state (loc, t) to the new location and timestep (line 15) and sets the *moved* flag to true (line 16). If the neighborhood $\mathcal{N}$ reaches the target size M during this step, the algorithm immediately returns the constructed neighborhood (line 17). Otherwise, it breaks out of the inner evaluation loop to proceed to the next walk iteration (line 18).

If the agent evaluates all candidate locations and cannot find a single feasible move (i.e., *moved* remains false), the random walk terminates early to avoid an infinite loop or deadlock (lines 20–21). The process repeats until the walking agent finishes its path or the neighborhood is fully populated, at which point the final neighborhood $\mathcal{N}$ is returned (line 22).

Algorithm 3: Entropy-Guided Walk

Input: Agent a_i, Path $\mathcal{P}_i$, Neighborhood $\mathcal{N}$, Neighborhood size M
Output: Updated Neighborhood $\mathcal{N}$

$t \leftarrow 0$;
$loc \leftarrow p_{i_0}$.loc;
while $t < |\mathcal{P}_i| - 1 \wedge |\mathcal{N}| < M$ **do**
  $next\_locs \leftarrow \text{neighbors}(loc) \cup \{loc\}$;
  $location\_scores \leftarrow [\,]$;
  foreach $l \in next\_locs$ **do**
    $p(a_j) \leftarrow \frac{\text{freq}(a_j, l, t+1)}{\sum_{a_k \in A} \text{freq}(a_k, l, t+1)}$;
    $H(l, t+1) \leftarrow -\sum_{a_j \in A} p(a_j) \log_2 p(a_j)$;
    $E(l) \leftarrow H(l, t+1) \cdot \frac{1}{1+h(l)}$;
    Append $(l, E(l))$ to $location\_scores$;
  $moved \leftarrow$ false;
  foreach $(l, E(l)) \in Sorted(location\_scores)$ **do**
    if $t + 1 + h(l) < |\mathcal{P}_i|$ **then**
      $\mathcal{N} \leftarrow \mathcal{N} \cup \text{ConflictingAgents}(loc, l, t+1)$;
      $(loc, t) \leftarrow (l, t+1)$;
      $moved \leftarrow$ true;
      if $|\mathcal{N}| \geq M$ **then**
        return $\mathcal{N}$;
      break;
  if *Not moved* **then**
    break;
return $\mathcal{N}$;

4.2 DyMAB Problem Definition

We formally define *DyMAB* as a non-stationary (i.e., dynamic) MAB problem designed to optimize neighborhood selection within ALNS-based MAPF solvers. The core challenge is to effectively manage a set of K arms representing heuristics, $H = \{H_{rand}, H_{sync}, H_{entr}\}$, where each heuristic $H_i \in H$ corresponds to a specific neighborhood selection strategy (i.e., random selection, synchronized random walk and entropy-based).

At each time step t, each heuristic $H_i \in H$ is associated with a reward distribution $D_i(t)$, characterized by a time-dependent mean reward:

$$\mu_i(t) = \mathbb{E}[r_{i,t}], \quad r_{i,t} \sim D_i(t), \tag{12}$$

where $r_{i,t}$ represents the reward obtained by applying the heuristic H_i at time t. The time dependency of $\mu_i(t)$ reflects the non-stationary nature of the reward distributions in DyMAB. We first use a sub-optimal algorithm to compute an initial solution P. Specifically, we employ *Prioritized Planning* (PP) (Erdmann and Lozano-Perez 1986), which is an effective sub-optimal solver as proven in the MAPF-LNS framework (Li, Chen, et al. 2021). From the initial solution P, a subset of agents $A_s \subseteq A$ is selected based on the defined neighborhood. Their paths, denoted as P_s^-, are removed and re-planned to yield new paths P_s^+. The *reward* for a heuristic H_i is the improvement in

the global SOC objective directly attributable to that heuristic at iteration t, expressed as:

$$r_{i,t} = l(\mathcal{P}_s^-) - l(\mathcal{P}_s^+), \tag{13}$$

where $l(\mathcal{P}_s) = \sum_{\mathcal{P}_j \in \mathcal{P}_s} l(\mathcal{P}_j)$ denotes the total path length of the selected agents before ($\mathcal{P}_s^-$) and after ($\mathcal{P}_s^+$) replanning. Since replanning affects only the selected subset and the global SOC objective is $\min \sum_i l(\mathcal{P}_i)$, $r_{i,t}$ is precisely the change in the global objective: a positive value means the solution improved, while zero or negative means it did not.

To address the non-stationary nature of heuristic performance, DyMAB maintains a *windowed reward memory* Q_i of fixed size W for each heuristic H_i, inspired by non-stationary bandit frameworks (Allesiardo et al. 2017; Garivier and Moulines 2008; Liu et al. 2013). After each call of a heuristic, the most recent reward $r_{i,t}$ is appended to Q_i; when the queue reaches its capacity, the oldest reward is discarded. This sliding window mechanism ensures that the estimate of the expected reward $\hat{\mu}_i(t)$ for each heuristic is based exclusively on the most recent W samples:

$$\hat{\mu}_i(t) = \frac{1}{|Q_i|} \sum_{r \in Q_i} r_{i,t}. \tag{14}$$

Consequently, this mechanism ensures that the algorithm remains responsive to changes in reward distributions of the heuristics over time.

4.3 DyMAB Solvers

To select heuristics dynamically during search, DyMAB employs two non-stationary MAB policies: DyMAB(ϵ-Greedy) and DyMAB(α-UCB). Both policies incorporate the windowed reward estimates and a logarithmically decaying exploration parameter (ϵ and α, respectively). This decay schedule gradually shifts the algorithm from exploration to exploitation over time.

4.3.1 DyMAB(ϵ-Greedy) Solver. DyMAB(ϵ-Greedy) is presented in Algorithm 4. Each heuristic $H_i \in H$ is initialized with an estimated mean reward $\hat{\mu}_{H_i} = 0$ and an empty sliding window queue Q_{H_i} (line 1). An initial solution P is computed using Prioritized Planning, and the iteration counter N and exploration rate ϵ_t are initialized (line 2). At each iteration, a heuristic is selected by ϵ-Greedy: with probability ϵ_t a random heuristic is chosen (exploration), otherwise the heuristic with the highest $\hat{\mu}_{H_i}$ is chosen (exploitation) (lines 5–8). The selected heuristic identifies a subset of agents A_s; their paths P^- are replanned to produce P^+ (lines 9–10), and P is updated if the cost improves (line 12). The reward $r_{H_i} = l(P^-) - l(P^+)$ is computed (line 13) and passed to UpdateQueue, which appends it to Q_{H_i} and discards the oldest entry if $|Q_{H_i}| > W$; $\hat{\mu}_{H_i}$ is recomputed as the mean of Q_{H_i} (line 14). The parameter ϵ_t decays logarithmically to progressively shift the policy toward exploitation (line 15). However, to mitigate local optimality and encourage continued exploration of the search space, ϵ_t is periodically increased (lines 16–17). This process continues until the predefined time budget is reached, at which point the best solution found is returned (lines 18–19).

4.3.2 DyMAB(α-UCB) Solver. DyMAB(α-UCB) is presented in Algorithm 5. Each heuristic $H_i \in H$ is initialized with an estimated mean reward $\hat{\mu}_{H_i} = 0$, a pull count $n_{H_i} = 0$, and an empty sliding window queue Q_{H_i} (line 1). An initial solution P is computed using Prioritized Planning (PP), while the iteration counter N and the initial exploration coefficient α are initialized (line 2). At each iteration of the search loop, the iteration counter is incremented (line 4), and the algorithm selects the heuristic maximizing the UCB index $\hat{\mu}_{H_i} + \sqrt{2\alpha/(n_{H_i}+1)}$ (line 5). The chosen heuristic identifies a subset of agents A_s to destroy, extracting their current paths P^- (line 6), before a repair operator replans new paths P^+ (line 7). If the new paths improve the solution cost, the global solution P is updated (lines 8–9). The reward $r_{H_i} = l(P^-) - l(P^+)$ is then computed (line 10) and processed by the UpdateQueue helper function (shared with Algorithm 4). This ensures the queue does not exceed the

Algorithm 4: DyMAB(ϵ-Greedy)

Input: Graph $G = (V, E)$, agents A, heuristic set H, ϵ, λ, W, T
Output: Collision-free paths P

1 $\hat{\mu}_{H_i} \leftarrow 0, Q_{H_i} \leftarrow [\,]$ **for each** $H_i \in H$;
2 $P \leftarrow \text{PP}(A, G)$; $N \leftarrow 0$; $\epsilon_t \leftarrow \epsilon$;
3 **repeat**
4 $N \leftarrow N + 1$;
5 **if** rand() $< \epsilon_t$ **then**
6 $H_i \leftarrow \text{RandomHeuristic}(H)$;
7 **else**
8 $H_i \leftarrow \arg\max_{H_j \in H} \hat{\mu}_{H_j}$;
9 $A_s \leftarrow H_i(A)$; $P^- \leftarrow \{P_j \mid a_j \in A_s\}$;
10 $P^+ \leftarrow \text{Replan}(G, A_s, P \setminus P^-)$;
11 **if** $l(P^+) < l(P^-)$ **then**
12 $P \leftarrow (P \setminus P^-) \cup P^+$;
13 $r_{H_i} \leftarrow l(P^-) - l(P^+)$;
14 $\text{UpdateQueue}(Q_{H_i}, r_{H_i}, W)$; $\hat{\mu}_{H_i} \leftarrow \text{mean}(Q_{H_i})$;
15 $\epsilon_t \leftarrow \epsilon / \log(1 + \lambda \cdot N)$;
16 **if** $N \bmod T = 0$ **then**
17 $\epsilon_t \leftarrow \max(\epsilon, \epsilon_t \cdot \log(N))$;
18 **until** *time budget reached*;
19 **return** P;

window size W, allowing the algorithm to recompute the mean reward $\hat{\mu}_{H_i}$ and update the active pull count n_{H_i} based exclusively on recent observations (line 11). Finally, the exploration coefficient α undergoes a logarithmic decay, progressively reducing the exploration bonus to smoothly shift the search policy from exploration toward exploitation over time (line 12). The algorithm continues until the time budget is exhausted, after which the best solution found so far is returned (lines 13–21).

5 Theoretical Analysis of DyMAB

We now provide a general formal theoretical justification for DyMAB for a set of K arms or heuristics, showing that under standard assumptions of non-stationary bandits, DyMAB achieves sub-linear regret, provided that the exploration parameters (α in DyMAB(α-UCB) and ϵ in DyMAB(ϵ-Greedy)) decay logarithmically.

Definition 3 (Non-Stationary MAB for Heuristic Selection). *We consider $H = \{H_1, H_2, \ldots, H_K\}$ a set of heuristics (arms). At each time t, the reward from heuristic H_i is drawn from distribution $D_i(t)$ with unknown and a time-varying mean $\mu_i(t)$. We assume a bounded reward ($r_t \in [0, 1]$) for simplicity, and piecewise stationary, meaning there is at most B breakpoints dividing the time horizon T into $B + 1$ stationary segments. DyMAB use a sliding window of size W.*

Algorithm 5: DyMAB(α-UCB)

Input: Graph $G = (V, E)$, agents A, heuristic set H, α_0, λ, W
Output: Collision-free paths P

$\hat{\mu}_{H_i} \leftarrow 0$, $n_{H_i} \leftarrow 0$, $Q_{H_i} \leftarrow [\,]$ **for each** $H_i \in H$;
$P \leftarrow \mathrm{PP}(A, G)$; $N \leftarrow 0$; $\alpha \leftarrow \alpha_0$;
repeat
  $N \leftarrow N + 1$;
  $H_i \leftarrow \arg\max_{H_j \in H} \left(\hat{\mu}_{H_j} + \sqrt{\frac{2\alpha}{n_{H_j} + 1}} \right)$;
  $A_s \leftarrow H_i(A)$; $P^- \leftarrow \{P_j \mid a_j \in A_s\}$;
  $P^+ \leftarrow \textsc{Replan}(G, A_s, P \setminus P^-)$;
  if $l(P^+) < l(P^-)$ **then**
    $P \leftarrow (P \setminus P^-) \cup P^+$;
  $r_{H_i} \leftarrow l(P^-) - l(P^+)$;
  $\textsc{UpdateQueue}(Q_{H_i}, r_{H_i}, W)$; $\hat{\mu}_{H_i} \leftarrow \mathrm{mean}(Q_{H_i})$; $n_{H_i} \leftarrow |Q_{H_i}|$;
  $\alpha \leftarrow \alpha_0 / (1 + \lambda \cdot \log(1 + N))$;
until *time budget reached*;
return P;

Definition 4 (Dynamic Pseudo-Regret). *The dynamic pseudo-regret incurred by a policy π over T rounds (iterations) is defined as:*

$$R_T^\pi = \sum_{t=1}^{T} \mu^*(t) - \mathbb{E}[r_t^\pi], \tag{15}$$

where $\mu^*(t) = \max_i \mu_i(t)$ is the reward of the best heuristic at time t, and r_t^π is the reward obtained by policy π.

Theorem 1 (Dynamic Regret Bound of DyMAB(α-UCB)). *Assume that DyMAB(α-UCB) operates in a piecewise stationary environment with at most B breakpoints and uses a sliding window of size W and an exploration parameter α decaying logarithmically as*

$$\alpha = \frac{\alpha_0}{(1 + \lambda \log(1 + N_{iteration}))} \tag{16}$$

for constants $\alpha_0, \lambda > 0$. Then, the dynamic pseudo-regret of DyMAB(α-UCB) is bounded as:

$$R_T = O\left(K \frac{T}{W} \log W + KBW \right). \tag{17}$$

Proof. We base our proof directly on Garivier and Moulines' theorem (Garivier and Moulines 2008, Theorem 7), which states:

For any integer τ and any arm $i \in \{1, \ldots, K\}$:

$$\mathbb{E}_\tau \left[\tilde{N}_T(i) \right] \leq C(\tau) \frac{T \log \tau}{\tau} + \tau \Upsilon_T + \log^2 \tau, \tag{18}$$

where $C(\tau)$ is a constant depending on τ and the reward gaps, and Υ_T is the number of breakpoints. $\tilde{N}_T(i)$ denotes the number of times a sub-optimal arm i has been chosen during the first T round.

In DyMAB, we use a sliding window of size W, corresponding to τ in the theorem, and we operate in a piecewise stationary environment with at most B breakpoints, thus $\Upsilon_T = B$.
Step 1: Regret within stationary segments.

Applying Garivier and Moulines' theorem (Garivier and Moulines 2008, Theorem 7) directly, for any suboptimal heuristic i in any stationary segment:

$$\mathbb{E}\left[\tilde{N}_T(i)\right] \leq O\left(\frac{T \log W}{W} + BW\right). \tag{19}$$

Step 2: Total regret over all arms.

Summing over all K arms:

$$R_T = \sum_{i=1}^{K} \mathbb{E}\left[\tilde{N}_T(i)\right] \leq O\left(K\frac{T \log W}{W} + KBW\right). \tag{20}$$

Step 3: Conclusion.

Thus, DyMAB(α-UCB), using a sliding window of size W achieves:

$$R_T = O\left(K\frac{T}{W}\log W + KBW\right). \tag{21}$$

□

In the MAB literature, $K\frac{T}{W}\log W$ correspond to the regret in a stationary segment and KBW to the regret incurred during the breakpoint.

Theorem 2 (Dynamic Regret Bound of DyMAB(ϵ-Greedy)). *Assume that DyMAB(ϵ-Greedy) operates in a piecewise stationary environment with at most B breakpoints, using a sliding window of size W. Assume an exploration probability ϵ_t decaying logarithmically as $\epsilon_t = \frac{\epsilon_0}{\log(1+\lambda t)}$ for constants $\epsilon_0, \lambda > 0$. Then, the dynamic pseudo-regret of DyMAB(ϵ-Greedy) is bounded as:*

$$R_T = O\left(\frac{T}{\log T} + KT \cdot \sqrt{\frac{\log W}{W}} + KBW\right). \tag{22}$$

Proof. We decompose the total regret into:

$$R_T = R_T^{\text{stationary}} + R_T^{\text{breakpoints}}. \tag{23}$$

Step 1: Regret within Stationary Segments ($R_T^{\text{stationary}}$). Consider a stationary segment of length τ_s, during which the reward distributions are fixed.

Exploration regret:

At each round t, with probability ϵ_t, DyMAB selects a heuristic uniformly at random. In the worst case, it incurs unit regret per round (i.e., $\Delta_i \leq 1$). Hence, the total exploration regret over the entire horizon T is the sum of regrets incurred at each exploration step:

$$R_T^{\text{explore}} = \sum_{t=1}^{T} \epsilon_t \leq \sum_{t=1}^{T} \frac{\epsilon_0}{\log(1+\lambda t)}. \tag{24}$$

For large T, this sum can be approximated by an integral, and it is known that $\sum_{t=1}^{T} \frac{1}{\log t} = O\left(\frac{T}{\log T}\right)$. Therefore, the total exploration regret is:

$$R_T^{\text{explore}} = O\left(\frac{T}{\log T}\right). \tag{25}$$

Exploitation regret:
With probability $1 - \epsilon_t$, DyMAB selects the heuristic with the highest empirical mean over the last W samples. Regret occurs if a suboptimal arm is selected.

Let i^* be the optimal heuristic with mean reward μ_{i^*}, and let $i \neq i^*$ be any suboptimal heuristic with μ_i and gap $\Delta_i = \mu_{i^*} - \mu_i > 0$. Let $\hat{\mu}_j(W)$ be the empirical mean over the last W samples for heuristic j.

We define:

$$\delta_W := \sqrt{\frac{\log W}{2W}}. \tag{26}$$

Assume $\Delta_i > 2\delta_W$. If both empirical estimates are close to their means:

$$|\hat{\mu}_i(W) - \mu_i| < \delta_W \quad \text{and} \quad |\hat{\mu}_{i^*}(W) - \mu_{i^*}| < \delta_W, \tag{27}$$

then:

$$\hat{\mu}_i(W) < \mu_i + \delta_W = \mu_{i^*} - \Delta_i + \delta_W < \mu_{i^*} - \delta_W < \hat{\mu}_{i^*}(W), \tag{28}$$

which contradicts $\hat{\mu}_i(W) \geq \hat{\mu}_{i^*}(W)$, because during exploitation, if the algorithm chooses the suboptimal arm i, it means that the empirical mean of arm i looked better than or equal to the empirical mean of the optimal arm i^*.

Thus, the event $\hat{\mu}_i(W) \geq \hat{\mu}_{i^*}(W)$ implies that at least one of the two deviations exceeds δ_W:

$$\mathbb{P}\left(\hat{\mu}_i(W) \geq \hat{\mu}_{i^*}(W)\right) \leq \mathbb{P}\left(|\hat{\mu}_i(W) - \mu_i| \geq \delta_W\right) + \mathbb{P}\left(|\hat{\mu}_{i^*}(W) - \mu_{i^*}| \geq \delta_W\right). \tag{29}$$

Applying Hoeffding's inequality (for bounded rewards in $[0, 1]$):

$$\mathbb{P}\left(|\hat{\mu}_j(W) - \mu_j| \geq \delta_W\right) \leq 2\exp(-2W\delta_W^2) = 2\exp(-\log W) = \frac{2}{W}. \tag{30}$$

Therefore:

$$\mathbb{P}\left(\hat{\mu}_i(W) \geq \hat{\mu}_{i^*}(W)\right) \leq \frac{4}{W}. \tag{31}$$

Expected exploitation regret per round:
We partition suboptimal arms into two categories:

- Large-gap arms ($\Delta_i > 2\delta_W$):
  For these arms, the probability of choosing a sub-optimal one is at most $\frac{4}{W}$ (by Hoeffding's inequality and union bound). Since $\Delta_i \leq 1$, their total contribution to regret per round is:

$$\sum_{i:\Delta_i > 2\delta_W} \Delta_i \cdot \frac{4}{W} \leq \frac{4K}{W}. \tag{32}$$

- Small-gap arms ($\Delta_i \leq 2\delta_W$):
  These arms are hard to distinguish from the optimal arm, but even if selected, they incur small regret. The total regret per round is bounded by:

$$\sum_{i:\Delta_i \leq 2\delta_W} \Delta_i \leq 2K\delta_W = O\left(K \cdot \sqrt{\frac{\log W}{W}}\right). \tag{33}$$

Combining both contributions, the total expected exploitation regret per round is:

$$O\left(\frac{K}{W} + K \cdot \sqrt{\frac{\log W}{W}}\right). \tag{34}$$

Since typically $\sqrt{\frac{\log W}{W}} > \frac{1}{W}$ for practical W, we can simplify:

$$O\left(\frac{K}{W} + K \cdot \sqrt{\frac{\log W}{W}}\right) = O\left(K \cdot \sqrt{\frac{\log W}{W}}\right). \tag{35}$$

Total exploitation regret over the entire horizon T:
The expectation of this per-round exploitation regret is taken over T rounds. Since the exploitation probability is $1 - \epsilon_t$, and ϵ_t decays, $1 - \epsilon_t \approx 1$ for large t. Summing over T rounds, the total exploitation regret is:

$$R_T^{\text{exploit}} = \sum_{t=1}^{T} (1 - \epsilon_t) \cdot O\left(K \cdot \sqrt{\frac{\log W}{W}}\right) = O\left(KT \cdot \sqrt{\frac{\log W}{W}}\right). \tag{36}$$

Total regret in stationary segments ($R_T^{stationary}$):
Combining the total exploration and total exploitation regrets:

$$R_T^{\text{stationary}} = R_T^{\text{explore}} + R_T^{\text{exploit}} = O\left(\frac{T}{\log T} + KT \cdot \sqrt{\frac{\log W}{W}}\right). \tag{37}$$

Step 2: Regret Due to Breakpoints ($R_T^{\text{breakpoints}}$).

After each of the B breakpoints, each heuristic may continue to rely on outdated samples for up to W rounds. In the worst case, this induces regret of up to 1 per round per heuristic. Summing over all B breakpoints and K arms:

$$R_T^{\text{breakpoints}} = O(KBW). \tag{38}$$

Step 3: Total Regret.

Combining both the stationary and breakpoint components:

$$R_T = R_T^{\text{stationary}} + R_T^{\text{breakpoints}} = O\left(\frac{T}{\log T} + KT \cdot \sqrt{\frac{\log W}{W}} + KBW\right). \tag{39}$$

□

Proposition 3 (Optimal Window for DyMAB(α-UCB)). *If the time horizon T and the number of breakpoints B in a piecewise-stationary environment, are known in advance, the window size W can be chosen so as to minimize regret bound for DyMAB(α-UCB).*

$$W^\star = \Theta\left(\sqrt{\frac{T \log T}{B}}\right). \tag{40}$$

Substituting this value yields:

$$\mathcal{R}_T^{\text{UCB}} = O\left(K\sqrt{TB \log T}\right). \tag{41}$$

Proposition 4 (Optimal Window for DyMAB(ϵ-Greedy)). *If the time horizon T and the number of breakpoints B in a piecewise-stationary environment, are known in advance, the window size W can be chosen so as to minimize regret bound for DyMAB(ϵ-Greedy).*

$$W^\star = \Theta\left(\left(\frac{T^2 \log T}{B^2}\right)^{1/3}\right). \tag{42}$$

Table 1. Comparison of optimal regret and window size for DyMAB policies

Policy	Regret Bound	Optimal Window Size $W^\star$
DyMAB(α-UCB)	$O\left(K\sqrt{TB\log T}\right)$	$\Theta\left(\sqrt{\frac{T\log T}{B}}\right)$
DyMAB(ϵ-Greedy)	$O\left(\frac{T}{\log T} + KT^{2/3}(\log T)^{1/3}B^{1/3}\right)$	$\Theta\left(\left(\frac{T^2\log T}{B^2}\right)^{1/3}\right)$

This yields the regret:

$$\mathcal{R}_T^{\epsilon\text{-}Greedy} = O\left(\frac{T}{\log T} + KT^{2/3}(\log T)^{1/3}B^{1/3}\right). \tag{43}$$

As presented in Table 1, the regret bounds for the two DyMAB policies differ in their asymptotic behavior. Given an optimal window size W^*, the regret for DyMAB(α-UCB) can be minimized up to $O(K\sqrt{TB\log T})$, which is a well known bound for sliding window UCB (Garivier and Moulines 2008; Jones et al. 2021). For DyMAB(ϵ-Greedy), the regret bound consists of two terms: $O\left(\frac{T}{\log T} + KT^{2/3}(\log T)^{1/3}B^{1/3}\right)$, where the first term accounts for the cumulative regret due to uniform random exploration, and the second term reflects the exploitation regret under non-stationarity.

Both policies achieve sublinear regret in T, but differ in their sensitivity to the number of breakpoints B and in the structure of their exploration-exploitation trade-offs. DyMAB(α-UCB) exhibits a tighter dependence on both T and B—scaling as $\sqrt{TB}$, which makes it theoretically more favorable in environments with infrequent distributional shifts. In contrast, DyMAB(ϵ-Greedy)'s regret grows more rapidly with T, but its structure is analytically simpler and does not require confidence-based computations. Therefore, from a purely theoretical standpoint, DyMAB(α-UCB) offers a stronger regret guarantee when the number of breakpoints B and the horizon T are known.

6 Experimental Evaluation

We evaluate DyMAB algorithms on several maps from the Moving AI Lab benchmarks (Stern et al. 2021), as well as our newly introduced city map, `Linköping(|V|=572000)`. Compared to the standard benchmark maps, the Linköping Campus Valla map is significantly larger, increasing the robustness of our evaluation. In all experiments, DyMAB uses exactly three heuristics: H_{rand}, H_{sync}, and H_{entr}. The original MAPF-LNS (Li, Chen, et al. 2021) heuristics H_{agent} and H_{grid} are deliberately excluded. Following the ALNS orthogonality principle (Li, Chen, et al. 2022), the pool should consist of operators covering distinct search dimensions: H_{sync} subsumes H_{agent} (both target delayed agents), and H_{entr} subsumes H_{grid} (both target spatially congested regions). Including all five would introduce redundancy without additional complementary coverage. To implement α-UCB(α, λ_1, W_1) and ϵ-Greedy$(\epsilon, \lambda_2, W_2, T)$, we performed a grid search to tune all hyperparameters. Unless stated otherwise, we use the following default values: exploration parameters $\alpha = 10000$ and $\epsilon = 0.5$, decay factors $\lambda_1 = 10$ and $\lambda_2 = 0.01$, reward queue sizes $W_1 = W_2 = 100$, and an exploration period $T = 1000$. To evaluate DyMAB, we conduct experiments across 25 random scenarios for each map, and report the average Sum of Delays(SOD) or Average Sum of Cost(SOC) and AUCs. All experiments were run on an Intel Core i9-10900X CPU with 128GB RAM. Implementations of the proposed algorithms are available as an open-source project[1].

[1]GitHub repository: https://github.com/amathsow/dymab

6.1 Experiment 0: Reward Non-stationarity

To empirically validate the piecewise-stationarity assumption underlying Theorems 1 and 2, we analyze how the reward of each neighbourhood heuristic evolves during the LNS search. We run the LNS framework with a *uniform random* policy, so that at every iteration each of the three heuristics (H_{rand}, H_{sync}, H_{entr}) is sampled with equal probability. This avoids selection bias and provides an unbiased estimate of each heuristic's reward at every stage of the search. We evaluate on three maps: `Dense520d` (400 agents), `random-64-64-10` (200 agents), and `Berlin_1_256` (800 agents), with a time budget of 100 seconds, averaged over 10 random scenarios. The reward at each iteration is defined as the SOC improvement normalized by neighbourhood size, i.e., $r_t = (\text{SOC}_{\text{old}} - \text{SOC}_{\text{new}})/|N_t|$. Fig. 2 presents two complementary views. The top row plots the mean reward per heuristic. All three heuristics exhibit high rewards in early iterations, when the solution is far from optimal and large improvements are easy to find, followed by a monotonic decrease as the solution quality improves. This global decay reflects the diminishing-returns nature of LNS and already indicates non-stationarity: the reward distribution at iteration 50 differs substantially from that at iteration 500. The bottom row factors out this global trend by displaying the *normalized reward share* of each heuristic, i.e., its reward as a fraction of the total reward across all heuristics at each iteration. This reveals clear *dominance shifts*: for example, on the Dense and Berlin maps, H_{sync} captures the largest reward share in the early phase, while H_{entr} gradually becomes dominant as the search progresses and congestion patterns change. These transitions in relative performance, visible across all three maps, are similar to a piecewise-stationary model in which the best arm changes over time, providing empirical support for the sliding-window approach used in DyMAB.

6.2 Experiment 1: DyMAB Convergence

This experiment assesses the convergence properties of DyMAB as the time budget increases and compares it against the anytime BALANCE baselines (Phan, Huang, et al. 2024). We plot the Sum of Delays (SOD) and the Area Under the Curve (AUC) with respect to the time budget on `Dense520d` ($|V| = 28178$, 400 agents) and `Berlin_1_256` ($|V| = 47540$, 600 agents). Results are averaged over 25 random scenarios; the Empirically Best Policy corresponds to the minimum SOD achieved across all configurations. As shown in Fig. 3, all DyMAB variants converge towards the near-optimal empirical best policy as the runtime increases, consistently achieving lower SOD and lower AUC than BALANCE-UCB1 and BALANCE-Thompson on both maps. Both DyMAB(α-UCB) and DyMAB(ϵ-Greedy) demonstrate comparable convergence speed, confirming that the non-stationary bandit formulation is robust to the choice of exploration policy.

6.3 Experiment 2: Neighborhood Selection

Here, we compare the performance of the three heuristics discussed in Section 4, such as H_{sync}, H_{entr} and H_{rand} against Agent-Based Neighborhood (H_{agent}), and Grid-Based Neighborhood (H_{grid}) presented in MAPF-LNS framework (Li, Chen, et al. 2021). The experiments are conducted on `random-64-64-20 (|V| = 3270)` map and the two maps used in the previous experiment. The results, detailed in Table 2, indicate that the performance of the heuristics varies depending on the specific map and the number of agents m. Among the heuristics, H_{sync} demonstrates consistent performance, achieving lower AOD and AUC values compared to H_{rand}, H_{agent}, and H_{grid}, particularly in dense and large-scale scenarios. H_{entr} frequently yielded the lowest SOD and AUC, leveraging information-theoretic measures to identify critical paths and agents. The Adaptive version, corresponding to DyMAB(α-UCB), demonstrates the best overall performance, maintaining consistently low SOD and AUC. This is expected, as DyMAB learns to select the most effective heuristic over time. If we choose DyMAB(ϵ-Greedy), similar performance will be observed.

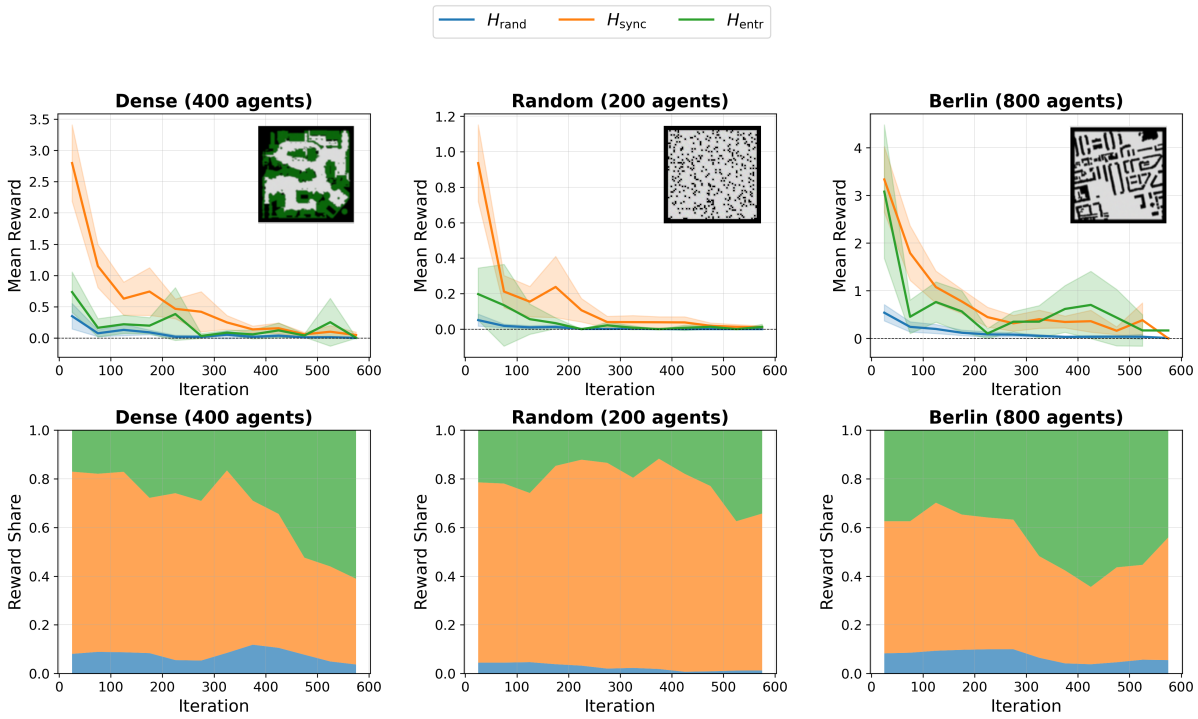

Fig. 2. Top: Mean reward per heuristic (H_{rand}, H_{sync}, H_{entr}) vs. iteration, with 95% confidence intervals. Bottom: Normalized reward share (fraction of total reward at each iteration). Results are averaged over 25 scenario.

6.4 Experiment 3: Effect of Sliding Window Size W

This experiment evaluates the impact of the sliding window size W on the performance of DyMAB(α-UCB) and DyMAB(ϵ-Greedy). The windowed reward queue Q plays a central role in enabling DyMAB to adapt to non-stationary environments by retaining recent reward observations for each heuristic. A smaller window size enables faster adaptation to changes in heuristic performance by emphasizing recent data, while a larger window provides more stable estimates, at the cost of slower responsiveness to changes.

All experiments are conducted with a time budget of 120 seconds on the Dense and Berlin maps, using varying numbers of agents. The average Sum of Delays over 25 random scenarios is plotted in Fig. 4.

DyMAB(α-UCB) benefits from larger window sizes, particularly when paired with higher exploration parameters (e.g., $\alpha = 50000$ or $\alpha = 100000$). Larger windows yield more reliable empirical mean estimates and tighter confidence bounds, which improve decision-making in environments where heuristic effectiveness changes frequently. This is consistent with the theoretical regret bound $O(K\sqrt{TB\log T})$, which is minimized when $W = \Theta(\sqrt{T\log T/B})$. In such cases, higher α values promote sufficient exploration early on, allowing the algorithm to identify the optimal heuristics with greater confidence. In contrast, DyMAB(ϵ-Greedy) exhibits stable performance across different values of W and ϵ, indicating greater robustness to parameter tuning. The random exploration ensures all heuristics are periodically sampled, making the method less sensitive to the precision of reward estimates. These results suggest that while DyMAB(α-UCB) is more sensitive to the proper tuning of W

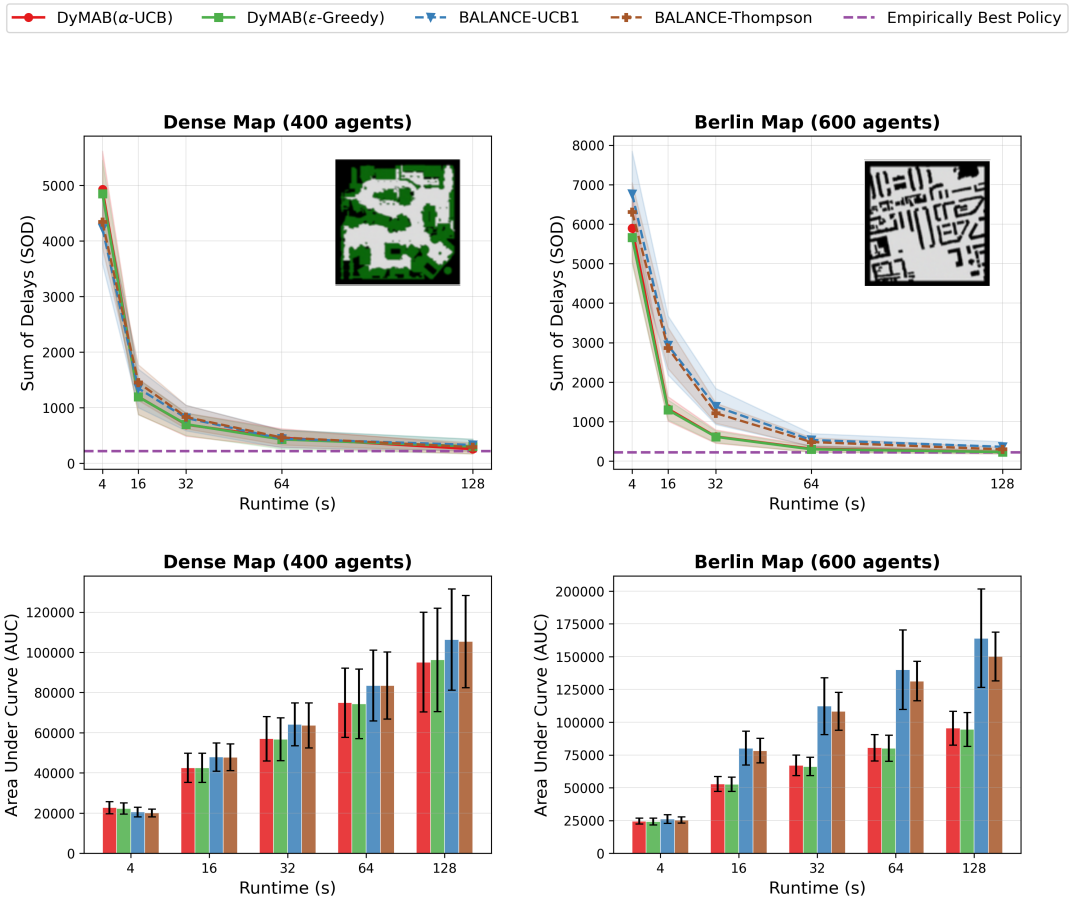

Fig. 3. Top:Average Sum of Delays for DyMAB(α-UCB) and DyMAB(ϵ-Greedy) with different time budgets compared with the empirical best Policy. Bottom: Corresponding AUC for each algorithm at each runtime budget. The error bar shows the 95% confidence interval. The runtime represents the time budget in seconds.

and α, it may achieve better performance when such tuning is feasible. Conversely, DyMAB(ϵ-Greedy) offers a more robust alternative under less favorable tuning conditions.

6.5 Experiment 4: Neighborhood Size M

We evaluate the performance of DyMAB and three heuristic methods under varying neighborhood sizes M across different maps and number of agents m. Figure 5 illustrates the average Sum of Delays across 25 scenarios for each map. As shown, the performance of the heuristics improves with increasing neighborhood size M. However, DyMAB demonstrates consistent performance regardless of M, achieving the smallest Sum of Delays across all scenarios. This stability is attributed to DyMAB's ability to focus on the most effective heuristic dynamically.

Table 2. Average SODs and AUCs for different heuristics: Random Neighbourhood (H_{rand}), Grid-Based Neighbourhood (H_{grid}), Agent-Based Neighbourhood (H_{agent}), Synchronised RandomWalk Neighbourhood (H_{sync}), Entropy-Based Neighbourhood (H_{entr}), and Adaptive using DyMAB(α-UCB) policy. The time budget is fixed at 100 seconds. Bold indicates the best value for SOD and AUC, and italic indicates the second-best.

Map	**m**	H_{rand}		H_{grid}		H_{agent}		H_{sync}		H_{entr}		**DyMAB(α-UCB)**	
		SOD	AUC	SOD	AUC	SOD	AUC	SOD	AUC	SOD	AUC	SOD	AUC
Random	100	4339.08	*931*	4330.76	933	4331.52	957	4331.20	953	**4313.24**	2412	*4330.48*	**846**
	200	8674.68	6685	8553.64	5806	8554.84	4226	*8552.88*	*4011*	8615.04	9154	**8550.60**	**3834**
	300	13003.29	37664	13016.71	22922	12899.04	12762	*12892.75*	*12454*	13002.41	24832	**12891.71**	**12431**
Dense	200	34705.56	46379	34875.72	51775	*34616.24*	**7444**	34640.72	15115	34765.16	28792	**34604.36**	*9082*
	400	72303.76	451169	71374.24	329440	*69491.40*	**93417**	**69490.56**	114784	70493.92	261053	69597.08	*102949*
	600	114151.08	1239469	110460.48	995797	**105949.52**	501808	106262.80	552941	110219.32	981363	*105992.76*	**455550**
Berlin	500	94269.08	540660	92313.88	339880	90289.52	*75805*	*90287.96*	80153	91478.76	295241	**90250.52**	**52460**
	800	158039.40	1544684	153406.36	1186468	147368.88	660376	*147360.20*	*660087*	153806.24	1259103	**146125.52**	**442094**
	1000	202849.91	2390499	201648.13	2249260	190882.13	1453086	*190456.25*	*1499752*	199148.04	2158026	**186354.17**	**1082926**

6.6 Experiment 5: State-of-the-art Comparison

This experiment compares DyMAB against the original MAPF-LNS, MAPF-LNS2, known to be very efficient, and two stationary MAB policies from BALANCE: Thompson Sampling and UCB1. We run all algorithms on five different maps such as `random-64-64-20(|V|=3270)`, `ostd003d(|V|=13214)`, `Dense520d(|V|=28178)`, `Berlin_1_256(|V|=47540)` and `Linkoping(|V|=572000)` with different numbers of agents m and a time budget of 120 seconds. The results presented in Fig. 6 show that all DyMAB policies (DyMAB(α-UCB) and DyMAB(ϵ-Greedy)) outperformed all other algorithms, especially in scenarios with a large number of agents. MAPF-LNS2 performs reasonably well in all maps compared to BALANCE and the original MAPF-LNS. DyMAB demonstrates adaptability, dynamically adjusting its heuristic selection through the sliding window rewards mechanism to respond effectively to changes in heuristic performance. Furthermore, DyMAB's convergence relies on its capability to learn to choose the best heuristic and a logarithmic decay that reduces the exploration over time. Additionally, in complex scenarios with larger maps and increasing agent counts, DyMAB achieves significant reductions in the Sum of Delays, outperforming recent state-of-the-art solvers in both computational efficiency and solution quality.

In Table 3, we report the number of iterations and AUCs for each algorithm. DyMAB consistently delivers the smallest AUC values, highlighting its rapid convergence across all tested maps and agent configurations. DyMAB(ϵ-Greedy) performs comparably to DyMAB(α-UCB), often within $1-2\%$ of AUC differences. DyMAB demonstrates adaptability, dynamically adjusting its heuristic selection through the sliding window rewards mechanism to respond effectively to changes in heuristic performance. Furthermore, DyMAB's convergence relies on his capability to learn to choose the best heuristic and a logarithmic decay that reduces the exploration over time. Additionally, in complex scenarios with larger maps and increasing agent counts, DyMAB achieves significant reductions in the Sum of Delays and AUC, outperforming recent state-of-the-art solvers in both computational efficiency and solution quality.

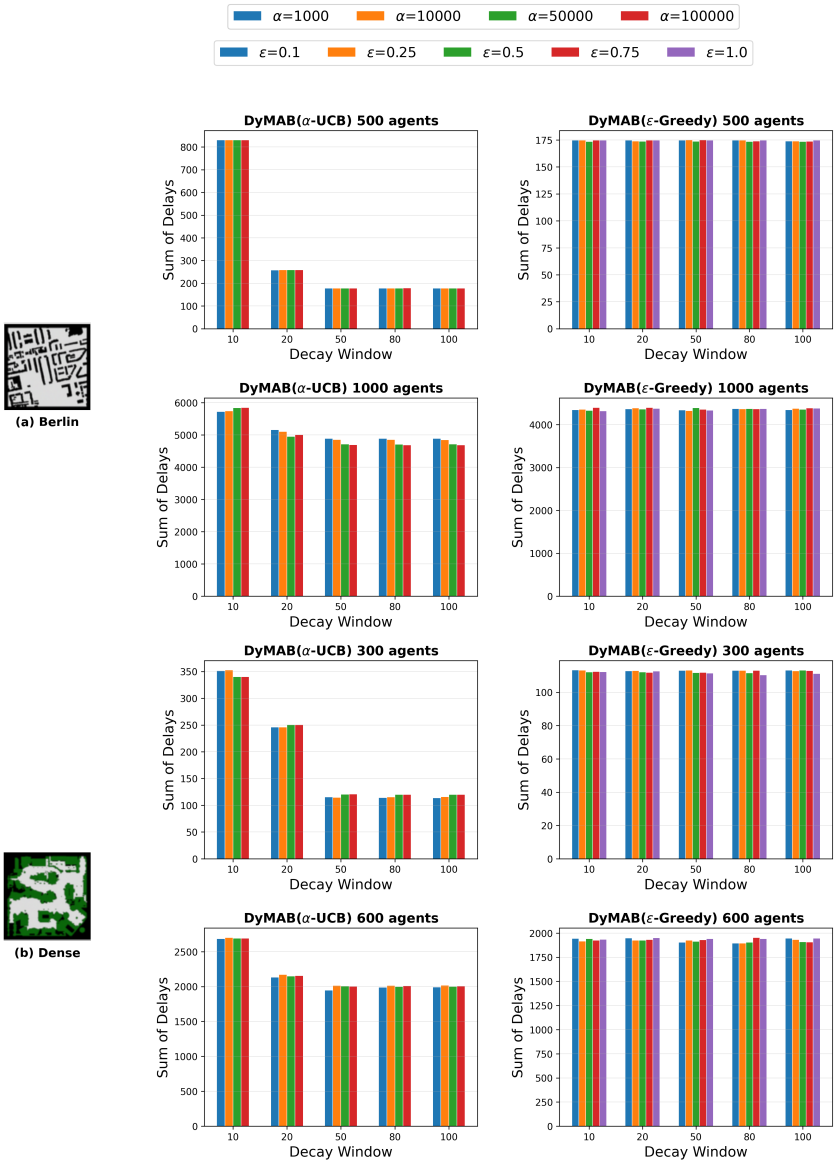

Fig. 4. Average Sum of Delays for DyMAB(α-UCB) and DyMAB(ϵ-Greedy) with different sliding window sizes W, and respective exploration parameters α and ϵ.

6.7 Experiment 6: Large-scale Test

In this experiment, we evaluate DyMAB on complex city maps and a large warehouse map to assess its performance in large-scale scenarios. The tested maps include `Boston_0_1024 (|V| = 796896)`, `Moscow_0_1024 (|V| = 795559)`, `NewYork_0_1024 (|V| = 792676)`, `Paris_1_1024 (|V| = 800729)`, `Shanghai_0_1024 (|V| =`

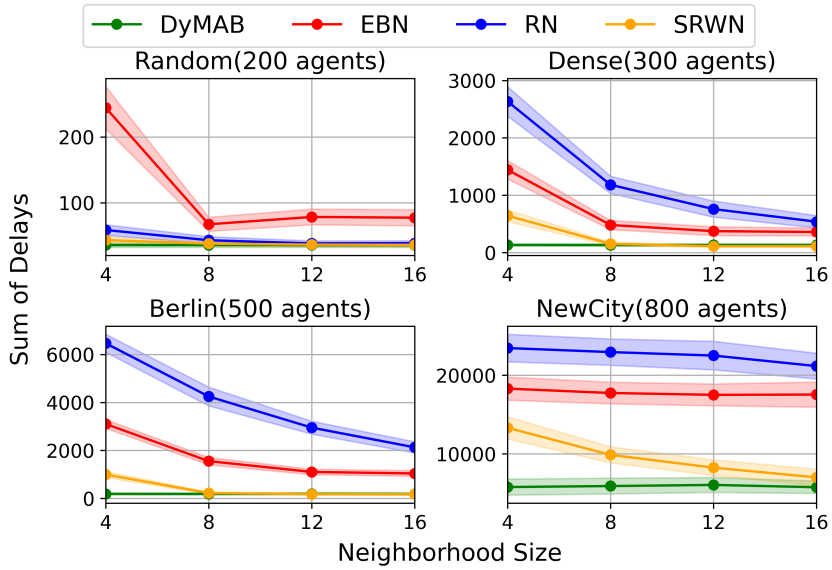

Fig. 5. Sum of delays for Random Neighborhood (RN), Synchronized RandomWalk Neighborhood (SRWN), Entropy-Based Neighborhood (EBN), and Adaptive using DyMAB(α-UCB) for different neighborhood sizes. Shaded areas show the 95% confidence interval. The legend applies to all plots.

`789333)`, `Sydney_0_1024 (|V| = 793424)`, and `warehouse-20-40-10-2-2 (|V| = 38756)` maps. The number of agents is fixed at $m = 3500$. Table 4 presents the results, comparing DyMAB with MAPF-LNS2, UCB1, Thompson Sampling, LaCAM (Okumura 2023b), LaCAM* (Okumura 2023a), and ADDRESS (Phan, Zhang, et al. 2025). DyMAB(α-UCB) consistently achieves competitive solution costs across all maps, and demonstrates strong anytime performance reflected by its AUC values. ADDRESS, which uses Thompson Sampling with a diverse heuristic pool, achieves faster early convergence on some maps (lower AUC on Boston) and a slightly lower final cost on Shanghai, while DyMAB's non-stationary adaptive selection yields better solutions overall. LaCAM and LaCAM* serve as reference points for complete search-based approaches; their higher costs confirm the advantage of anytime LNS methods at this scale. LaCAM* produces results similar to LaCAM at this scale because the initial solution search exhausts the majority of the time budget, leaving little room for refinement.

7 Conclusion

We have presented DyMAB as an anytime MAPF framework that addresses the non-stationary nature of heuristic performance for the ALNS-based solver. We proposed two heuristics, H_{sync} and H_{entr}, that explore the solution space efficiently and two policies, DyMAB(α-UCB) and DyMAB(ϵ-Greedy), that dynamically learn to choose the best heuristic while taking account of the heuristic shift in performance over time. This adaptability, coupled with their exploration-exploitation strategies(UCB with α decaying and ϵ-Greedy with ϵ decaying), ensures that DyMAB consistently produces near-optimal solutions within constrained time budgets. Our theoretical analysis demonstrates that DyMAB achieves sublinear regret under piecewise-stationary conditions. Although this standard non-stationarity model serves as an abstraction of the full MAPF-LNS setting, this formalization provides the necessary foundation to theoretically guarantee the framework's overall effectiveness.

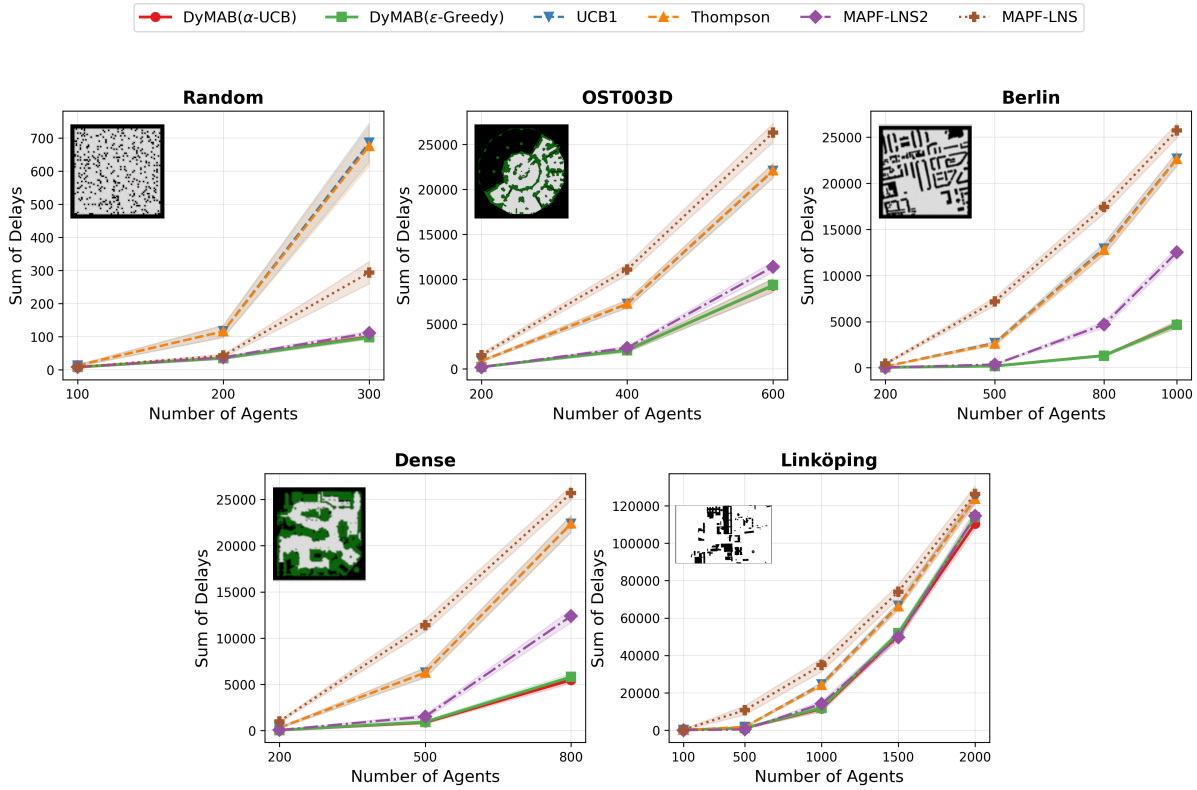

Fig. 6. Sum of delays for DyMAB(α-UCB) and DyMAB(ϵ-Greedy) compared with BALANCE (Thompson), BALANCE (UCB1), MAPF-LNS and MAPF-LNS2 for different numbers of agents. All experiments are run on the same hardware configuration with a time budget of 120 seconds. Shaded areas show the 95% confidence interval. The legend applies to all plots.

Our empirical evaluation demonstrates that DyMAB consistently outperforms recent state-of-the-art MAPF solvers, including MAPF-LNS, MAPF-LNS2, LaCAM, LaCAM*, and BALANCE (using both Thompson Sampling and UCB1), ADDRESS, across a diverse range of benchmark scenarios. It is also comparatively most effective at large-scale scenarios, presenting the lowest SOD and AUC.

Although DyMAB framework is tailored to MAPF, its underlying methodology is generalizable to other combinatorial domains involving concurrent heuristic selection, such as Vehicle Routing Problems(VRP), Multi-Agent Pickup and Delivery (MAPD).

Furthermore, even if DyMAB effectively selects among a given set of heuristics, its overall performance is still fundamentally dependent on the quality and diversity of the provided neighborhood heuristics (H_{sync}, H_{entr}, H_{rand}). The development of even more effective heuristics could further enhance the performance. The current DyMAB framework focuses on non-stationarity within the search process for a static MAPF instance. A natural extension would be to adapt DyMAB to handle fully dynamic MAPF problems, where the environment itself changes over time (e.g., new obstacles, changing agent goals). From a modeling perspective, another promising direction is to consider richer formulations for heuristic selection. While we currently model heuristic selection as a non-stationary MAB problem, the ALNS search process inherently involves state transitions that align

Table 3. Average iterations and AUCs for all algorithms. "Iter" refers to the number of iterations. Bold indicates the best value for AUC.

Map	m	MAPF-LNS		MAPF-LNS2		Thompson		UCB1		DyMAB(α-UCB)		DyMAB(ϵ-Greedy)	
		Iter	AUC	Iter	AUC	Iter	AUC	Iter	AUC	Iter	AUC	Iter	AUC
Random	100	16384	1134	154801	1027	192583	1705	185417	1638	62079	1021	61369	**1014**
	200	4250	12054	70176	5059	82540	19550	79658	18928	22128	4559	21937	**4535**
	300	1964	95324	38010	18925	42659	116424	41248	112617	13474	14475	13454	**14260**
Ost003d	200	297	269581	11839	40477	17576	140851	18106	134394	2282	51300	2306	**50964**
	400	87	1408665	5241	505958	6544	1062745	6754	1012631	1249	430504	1253	**429507**
	600	35	2841247	2479	1887567	2907	2826113	2983	2706886	635	**1666151**	628	1671842
Dense	200	105	166418	12501	10766	16700	46760	16863	44696	3223	**9818**	3061	10125
	500	24	1397331	3968	405961	4547	932364	4586	892128	1144	**246785**	1048	261397
	800	11	3050654	1829	2019158	1874	2820094	1901	2701834	736	**1263804**	671	1315039
Berlin	200	84	80823	16029	**3528**	19857	19330	18790	18625	5913	6451	6007	5475
	500	17	897086	4768	124255	5184	439184	4888	428000	1613	56956	1641	**56260**
	800	9	2089424	2125	994106	2125	1724541	2003	1663081	901	479939	917	**474174**
	1000	7	3163219	1373	2027183	1274	2744739	1224	2633097	671	1205120	685	**1194283**
Linköping	100	34	40711	2791	**1195**	3643	6631	3477	6366	2031	19846	1775	14580
	500	4	1252670	846	**169996**	1001	350159	932	346250	270	328572	279	324671
	1000	2	3717274	333	2225756	363	3089092	344	2939158	113	**2038848**	113	2080958
	1500	2	6807155	155	5759002	160	6687516	150	6220139	66	**5445558**	64	5461020
	2000	2	8891196	71	8987055	64	8626836	61	8079584	38	8979265	35	**8532032**

Table 4. SOC and AUC with different large-scale maps (m = 3500 agents, 500-second time budget). All values are divided by 1000. Bold indicates the best value for SOC and AUC; italic indicates the second-best.

Map	MAPF-LNS2		UCB1		Thompson		DyMAB(α-UCB)		DyMAB(ϵ-Greedy)		LaCAM	LaCAM*	ADDRESS	
	SOC	AUC	SOC	AUC	SOC	AUC	SOC	AUC	SOC	AUC	SOC	SOC	SOC	AUC
Warehouse	942.10	147109	942.91	147357	942.91	147343	**931.31**	**140329**	*937.99*	142100	1069.89	1069.89	938.36	*141082*
Boston	*3453.63*	152310	3467.10	152834	3467.10	152836	**3450.02**	*127233*	3460.53	128139	3659.15	3658.54	3457.87	**125487**
Moscow	3570.87	168560	3582.12	177995	3582.12	178942	**3555.74**	**146424**	3578.85	151337	3796.03	3797.79	*3562.48*	*151229*
New York	3527.64	116448	3529.80	118252	3529.80	118754	**3513.80**	**73414**	3527.61	*84457*	3739.48	3734.40	*3517.13*	84699
Paris	3418.99	148593	3432.21	141770	3432.18	156264	**3406.73**	**127768**	3431.01	*130885*	3616.78	3616.78	*3415.79*	132372
Shanghai	3628.19	113580	3636.51	143298	3637.65	138719	*3621.27*	**99255**	3638.58	100647	3877.59	3868.83	**3620.23**	*100477*
Sydney	**3575.25**	159462	3595.47	174229	3595.47	175640	*3583.26*	**142397**	3587.79	*144329*	3822.20	3821.29	3587.48	144438

with a Markov Decision Process (MDP) or a contextual bandit. Consequently, the scope of our theoretical analysis guarantees performance relative to the best temporal sequence of heuristics, rather than an optimal state-dependent policy. Future work utilizing a full MDP formulation would explicitly account for this policy-dependent evolution of the solution state, while a contextual bandit approach could condition heuristic selection on features of the current solution, potentially yielding more targeted adaptive behavior.

Acknowledgments

This work is supported in part by Sweden's Innovation Agency Vinnova (Projects 2022-02671 DyMuDRoP and 2024-01322 AHA-IMPUT) and the Excellence Center at Linköping-Lund in Information Technology (ELLIIT).

References

R. Allesiardo, R. Feraud, and O.-A. Maillard. June 2017. "The Non-stationary Stochastic Multi-armed Bandit Problem." *International Journal of Data Science and Analytics*, 3, (June 2017). doi:10.1007/s41060-017-0050-5.

M. Baranwal, L. Marla, C. Beck, and S. M. Salapaka. 2022. "A unified Maximum Entropy Principle approach for a large class of routing problems." *Elsevier*, 171.

M. Barer, G. Sharon, R. Stern, and A. Felner. 2014. "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem." In: *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS)*. SoCS, NLD, 961–962. https://doi.org/10.1609/socs.v5i1.18315.

O. Besbes, Y. Gur, and A. J. Zeevi. 2014. "Optimal Exploration-Exploitation in a Multi-Armed-Bandit Problem with Non-stationary Rewards." *ArXiv*, abs/1405.3316. https://api.semanticscholar.org/CorpusID:9425285.

E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony. 2015. "ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding." In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*. IJCAI, South America, 740–746. https://www.ijcai.org/Proceedings/15/Papers/110.pdf.

E. Demir, T. Bektaş, and G. Laporte. 2012. "An adaptive large neighborhood search heuristic for the Pollution-Routing Problem." *European Journal of Operational Research*.

M. Erdmann and T. Lozano-Perez. 1986. "On multiple moving objects." In: *Proceedings. 1986 IEEE International Conference on Robotics and Automation*. Vol. 3. IEEE, San Francisco, CA, 1419–1424. doi:10.1109/ROBOT.1986.1087401.

A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek. 2017. "Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges." *AAAI*, 8, 9.

A. Garivier and E. Moulines. 2008. "On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems." *arXiv: Statistics Theory*. https://api.semanticscholar.org/CorpusID:15513611.

T. Huang, J. Li, S. Koenig, and B. Dilkina. June 2022. "Anytime Multi-Agent Path Finding via Machine Learning-Guided Large Neighborhood Search." *Proceedings of the AAAI Conference on Artificial Intelligence*, 36, 9, (June 2022), 9368–9376. doi:10.1609/aaai.v36i9.21168.

B. Jones, J. Brennan, Y. Chen, and J. Filipek. 2021. "Multi-Armed Bandits with Non-Stationary Means." *Semantic Scholar*.

J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig. Aug. 2021. "Anytime Multi-Agent Path Finding via Large Neighborhood Search." In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*. Association for the Advancement of Artificial Intelligence (AAAI), Montreal, Canada, (Aug. 2021), 4127–4135. doi:10.24963/ijcai.2021/568.

J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig. June 2022. "MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search." *Proceedings of the AAAI Conference on Artificial Intelligence*, 36, 9, (June 2022), 10256–10265. doi:10.1609/aaai.v36i9.21266.

J. Li, W. Ruml, and S. Koenig. May 2021. "EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding." *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, 14, (May 2021), 12353–12362. doi:10.1609/aaai.v35i14.17466.

Y. Liu, X. Kuang, and B. V. Roy. 2013. "A Definition of Non-Stationary Bandits." *Arxiv*. doi:10.48550/arXiv.2302.12202.

T.-Y. Ma. 2011. "A cross entropy multiagent learning algorithm for solving vehicle routing problems with time windows." *Springer*, 6971, 59–73.

K. Okumura. Aug. 2023a. "Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding." In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. Ed. by E. Elkind. Main Track. International Joint Conferences on Artificial Intelligence Organization, (Aug. 2023), 243–251. doi:10.24963/ijcai.2023/28.

K. Okumura. 2023b. "LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding." *Proceedings of the AAAI Conference on Artificial Intelligence*, 37. doi:10.1609/aaai.v37i10.26377.

T. Phan, T. Huang, B. Dilkina, and S. Koenig. Mar. 2024. "Adaptive Anytime Multi-Agent Path Finding Using Bandit-Based Large Neighborhood Search." *Proceedings of the AAAI Conference on Artificial Intelligence*, 38, 16, (Mar. 2024), 17514–17522. doi:10.1609/aaai.v38i16.29701.

T. Phan, B. Zhang, S.-H. Chan, and S. Koenig. Apr. 2025. "Anytime Multi-Agent Path Finding with an Adaptive Delay-Based Heuristic." 39, 22, (Apr. 2025), 23286–23294. eprint: https://thomyphan.github.io/files/2025-aaai-preprint1.pdf. doi:https://doi.org/10.1609/aaai.v39i22.34495.

M. Phillips and M. Likhachev. 2011. "SIPP: Safe interval path planning for dynamic environments." In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, Shanghai, China, 5628–5635. doi:10.1109/ICRA.2011.5980306.

H. Qi, Y. Wang, and L. Zhu. 2023. "Discounted Thompson Sampling for Non-Stationary Bandit Problems." *arXiv*. doi:10.48550/arXiv.2305.10718.

Q. Sajid, R. Luna, and K. E. Bekris. 2012. "Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives." *AAAI*. doi:https://doi.org/10.1609/socs.v3i1.18243.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. "Conflict-based search for optimal multi-agent pathfinding." *Artificial Intelligence*, 219, 40–66. doi:https://doi.org/10.1016/j.artint.2014.11.006.

P. Shaw. 1998. "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems." In: *Principles and Practice of Constraint Programming — CP98*. Ed. by M. Maher and J.-F. Puget. Springer Berlin Heidelberg, Berlin, Heidelberg, 417–431. ISBN: 978-3-540-49481-2. doi:10.1007/3-540-49481-2_30.

J. Song, ravi lanka, Y. Yue, and B. Dilkina. 2020. "A General Large Neighborhood Search Framework for Solving Integer Linear Programs." *NeurIPs*.

"Multi-Agent Path Finding – An Overview." *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*. Springer International Publishing, Cham, 96–115. ISBN: 978-3-030-33274-7. doi:10.1007/978-3-030-33274-7_6.

R. Stern et al.. 2021. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks." *Proceedings of the International Symposium on Combinatorial Search*, 10, 151–158. doi:10.1609/socs.v10i1.18510.

F. Trovo, S. Paladino, M. Restelli, and N. Gatti. 2020. "Sliding-Window Thompson Sampling for Non-Stationary Settings." *Journal of Artificial Intelligence Research(JAIR)*. doi:10.1613/jair.1.11407.

S. Windras Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. Rifai. June 2022. "A survey of adaptive large neighborhood search algorithms and applications." *Computers and Operations Research*, 146, (June 2022), 105903. doi:10.1016/j.cor.2022.105903.

Received 30 May 2025; accepted 22 June 2026