# Reevaluation of Large Neighborhood Search for MAPF: Findings and Opportunities

**Jiaqi Tan**[1*], **Yudong Luo**[2*], **Jiaoyang Li**[3], **Hang Ma**[1]
[1]School of Computing Science, Simon Fraser University, Canada
[2]School of Computer Science, University of Waterloo, Canada
[3]Robotics Institute, Carnegie Mellon University, USA
{jiaqit, hangma}@sfu.ca, yudong.luo@uwaterloo.ca, jiaoyangli@cmu.edu

## Abstract

Multi-Agent Path Finding (MAPF) aims to arrange collision-free goal-reaching paths for a group of agents. Anytime MAPF solvers based on large neighborhood search (LNS) have gained prominence recently due to their flexibility and scalability, leading to a surge of methods, especially those leveraging machine learning, to enhance neighborhood selection. However, several pitfalls exist and hinder a comprehensive evaluation of these new methods, which mainly include: 1) Lower than actual or incorrect baseline performance; 2) Lack of a unified evaluation setting and criterion; 3) Lack of a codebase or executable model for supervised learning methods. To address these challenges, we introduce a unified evaluation framework, implement prior methods, and conduct an extensive comparison of prominent methods. Our evaluation reveals that rule-based heuristics serve as strong baselines, while current learning-based methods show no clear advantage on time efficiency or improvement capacity. Our extensive analysis also opens up new research opportunities for improving MAPF-LNS, such as targeting high-delayed agents, applying contextual algorithms, optimizing replan order and neighborhood size, where machine learning can potentially be integrated.

## 1 Introduction

Multi-Agent Path Finding (MAPF) refers to the problem of arranging collision-free paths for a group of agents (Stern et al. 2019). Many real-world applications involving multiple agents are closely related to MAPF, e.g., warehouse robots (Ma and Koenig 2017; Li et al. 2021c), aircraft-towing vehicles (Morris et al. 2016; Fines, Sharpanskykh, and Vert 2020), and navigation in video games (Ma et al. 2017).

MAPF is NP-hard to solve optimally (Yu and LaValle 2013). In recent years, anytime MAPF solvers based on large neighborhood search (LNS) (Li et al. 2021a) have gained prominence since previous centralized solvers often suffer from poor efficiency with low scalability despite their solution optimality, e.g, conflict-based search (CBS) (Sharon et al. 2015), or low solution quality despite their fast speed and good scalability, e.g., prioritized planning (PP) (Erdmann and Lozano-Perez 1987). Learning decentralized suboptimal policies via reinforcement learning has also been explored, but typically requires subtle environment design (Sartoretti et al. 2019; Ma, Luo, and Ma 2021; Ma, Luo, and Pan 2021). Among these approaches, MAPF-LNS has emerged as the leading method for finding fast and near-optimal solutions to large-scale MAPF problems within a time budget. It starts from a fast initial solution, often obtained using a fast suboptimal MAPF solver, e.g., PP, and incrementally improves the solution quality to near-optimal over time. In LNS, a subset of agents, called a neighborhood, is selected, and their paths are iteratively destroyed and repaired. MAPF-LNS has consistently ranked first in various competitions, including the 2023 Robot Runners (Jiang et al. 2024), AMLD 2021 Flatland 3 Challenge (Chen et al. 2023), and the 2020 NIPS Flatland Challenge (Li et al. 2021b), demonstrating its excellence in both speed and solution quality.

One key challenge of MAPF-LNS lies in selecting an improving neighborhood to efficiently minimize total delays. To address this, various strategies have been proposed, which generally fall into two categories: **rule-based** and **learning-based** methods. Rule-based strategies rely on predefined heuristics to generate neighborhoods (Li et al. 2021a), while learning-based strategies predict the optimal neighborhood generated by rule-based strategies (Huang et al. 2022; Yan and Wu 2024) or dynamically select one of the rule-based strategies based on environmental conditions (Phan et al. 2024). However, as an emerging research topic, no current work systematically examines the efficiency of different MAPF-LNS methods, especially the new advances using machine learning. Upon examination, we find several pitfalls in their evaluation, which impede a reasonable comparison. This includes: 1) **Underreported or incorrect performance**. We observe that the final delays of rule-based methods reported in Huang et al. (2022) are usually significantly higher than those in Li et al. (2021a). Phan et al. (2024) directly import these values from Huang et al. (2022). Additionally, Yan and Wu (2024) adopts final delays from Li et al. (2021a) but uses slightly different maps for evaluation, leading to inconsistencies. For example, the result of map 'random-32-32-20' in Li et al. (2021a) has been incorrectly adopted as the result for 'random-32-32-10' by Yan and Wu (2024). Such discrepancies make it difficult to draw reliable conclusions from current results. 2) **Lack of a unified setting**. Various factors potentially influence the efficiency of MAPF-LNS, such as the initial solution and path replan solver. Initial

[*]These authors contributed equally.

solutions generally vary among different MAPF-LNS papers, and unsolved scenarios are usually discarded. While PP serves as the default replan solver in most MAPF-LNS methods, priority-based search (PBS) (Ma et al. 2019) is used by Yan and Wu (2024). Different evaluation metrics are also employed in different papers, such as area under the curve (AUC) (Li et al. 2021a), win/loss (Huang et al. 2022), and average gap (Yan and Wu 2024). This lack of a unified setting makes direct comparison difficult. 3) **Lack of a codebase or executable model**. The performance of supervised learning heavily depends on data quality and parameter tuning. However, codebases, running instructions, or executable models are generally missing for supervised learning methods, making it challenging to reproduce their results.

To address these issues, we propose a unified evaluation setting under the same benchmark and hyperparameter configurations. We investigate and standardize several key aspects in MAPF-LNS, including initial solutions, replan solvers, and time-counting schemes during evaluation, which are not fully studied, obscure, or incorrect in previous works. We then implement and reevaluate prior methods in this unified framework. Our key finding is that rule-based heuristics for neighborhood selection are still strong baselines compared to learning-based methods in terms of time efficiency and improvement capacity. Our analysis also leads to several interesting future directions for improving MAPF-LNS, which are less explored in the previous MAPF-LNS literature, e.g., targeting high-delayed agents, applying contextual algorithms, optimizing replan order and neighborhood size, where machine learning can potentially be integrated.

# 2 Preliminaries

## 2.1 Background: MAPF and MAPF-LNS

The MAPF variants are summarized by Stern et al. (2019). In this work, we follow the common settings: 1) considering vertex and swapping conflicts, i.e., agents can not occupy the same vertex or traverse the same edge in opposite directions simultaneously; 2) agents 'stay at target' instead of disappearing; 3) minimizing the sum of individual costs, i.e., the total time steps for all agents to reach their targets.

**MAPF** is formally defined as follows. The input is a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, along with a set of $N$ agents $A = \{a_1, ..., a_N\}$. Each agent $a_i$ is assigned a start vertex $s_i \in V$ and a goal (target) vertex $g_i \in V$ ($g_i$ is accessible from $s_i$). At each discrete time step, an agent can either *move* to an adjacent vertex or *wait* at its current vertex. Consequently, the path $p_i$ of $a_i$ consists of a sequence of vertices that are adjacent or identical. A *solution* is a set of collision-free paths, one for each agent from $s_i$ to $g_i$. Let $d(s_i, g_i)$ denote the length of the shortest path between $s_i$ and $g_i$, and $l(p_i)$ denote the length of path $p_i$. Then, the delay of path $p_i$ is $delay(p_i) = l(p_i) - d(s_i, g_i)$. Note that $l(p_i)$ counts the edges of both *move* and *wait* actions. The task is to find a *solution* $P = \{p_i\}_{i=1}^{N}$ that minimizes the sum of costs $\sum_{i=1}^{N} l(p_i)$, which equals to minimize the sum of delays $\sum_{i=1}^{N} delay(p_i)$.

**LNS** is a type of improvement heuristic that iteratively reoptimizes a solution by the *destroy* and *repair* operations until some stopping condition is met (Pisinger and Ropke 2010). In the *destroy* operation, it breaks a part of the solutions, named a *neighborhood*. In *repair* operation, it solves the reduced problem by treating the remaining part as fixed.

**MAPF-LNS** framework operates as follows: given a MAPF instance, an initial (non-optimal) *solution* $P_0$ is obtained via a non-optimal solver. In each iteration $k$, a subset of agents $\tilde{A} \subset A$ is selected based on a criterion. $\tilde{A}$ is called a neighborhood. The paths of agents in $\tilde{A}$ are then removed from previous solution $P_{k-1}$, resulting in $P_{k-1}^{-} = \{p_i \in P_{k-1} | a_i \notin \tilde{A}\}$. Subsequently, those paths of $\tilde{A}$ are replanned by an algorithm to avoid collisions with each other and with the paths in $P_{k-1}^{-}$. If the resulting solution has a smaller sum of delays than $P_{k-1}$, it is accepted as $P_k$, otherwise, $P_k$ remains as $P_{k-1}$.

## 2.2 Neighborhood Selection

Selecting the neighborhood is crucial to the success of MAPF-LNS. In this section, we give an overview of the existing selection strategies.

**Rule-based strategies:** There are four major rule-based strategies in the literature: **RandomWalk**, **Intersection**, **Random**, and **Adaptive**, proposed by Li et al. (2021a). Different rule-based strategies improve the current solution from different perspectives. *RandomWalk* lets an unoptimized high-delayed agent move towards a shorter path and collect colliding agents and itself as a neighborhood. *Intersection* focuses on improving solutions around intersection vertices (vertices with a degree greater than two) by adding agents that visit the same intersection to the neighborhood. *Random* strategy selects agents uniformly at random from the set of agents, ensuring a broad exploration of the solution space. Although simple, this method introduces sufficient diversity, preventing the algorithm from getting stuck in local optima, and is widely used in LNS (Demir, Bektaş, and Laporte 2012; Song et al. 2020). *Adaptive* dynamically switches between RandomWalk, Intersection, and Random strategies, adjusting their sampling probability weights based on relative success in improving the current solution.

**Learning-based methods:** There are three prominent learning-based strategies: **SVM-LNS** (Huang et al. 2022) (denoted as SVM), **Neural-LNS** (Yan and Wu 2024) (denoted as NNS), and **Bandit-LNS** (Phan et al. 2024) (denoted as Bandit). *SVM* and *NNS* are supervised learning methods, where a ranking model is trained to predict the best neighborhood over a set of neighborhood candidates generated by rule-based strategies. *Bandit* incorporates a bandit algorithm to select rule-based strategies and neighborhood size as bi-level arms. The reward signal is the delay improvement.

## 2.3 Discussion on Selection Strategies

**RandomWalk** (Li et al. 2021a). The main idea of this heuristic is to prioritize replanning for agents with high delays. On investigating this method, we find some designs of its algorithm may hinder re-optimizing high-delayed agents, e.g., once a high-delayed agent is selected in one round, it is added to a tracking list that will not be selected in next round. Also, agents are randomly chosen if the neighborhood

size is not reached after one RandomWalk search. We make two modifications to RandomWalk by removing the tracking list and sampling agents according to their delays. We name this modified heuristic as **RandomWalkProb**, and we find these simple modifications lead to significant improvement in several domains. (See Sec. 6.2 in Appendix on details of RandomWalkProb.)

**SVM** (Huang et al. 2022) and **NNS** (Yan and Wu 2024). Both two utilize supervised learning to rank neighborhood candidates given the query information, i.e., some solution $P_k$ in iteration $k$. This implies the queries $\{P_k\}_{k=1}^{T}$ are treated as independent and identically distributed (i.i.d.) during training. However, they are non i.i.d. in practice since $P_{k+1}$ highly depends on $P_k$ within the sequential optimization of LNS. In the literature of learning to rank, several works suggest explicitly capturing the temporal information among queries to increase the robustness and generalization ability of rank models (Yu et al. 2019; Li, Wang, and McAuley 2020).

**Bandit** (Phan et al. 2024). MAPF-LNS can be framed as a contextual bandit problem, where optimal actions (e.g., strategies and neighborhood sizes) should be determined based on the context information (e.g., the map and the solution $P_k$ at iteration $k$). Different contexts can be treated as distinct states. However, Bandit-LNS employs non-contextual bandit algorithms, which operate under the assumption of a single state or no state at all, and thus select optimal actions without considering contextual information. This creates a theoretical inconsistency, as the problem setting (contextual) does not align with the algorithm applied (non-contextual).

# 3 A Unified Setting for Evaluation

In this section, we elaborate our unified setting for MAPF-LNS evaluation[1]. We investigate and standardize several key aspects of MAPF-LNS before conducting a comprehensive comparison among existing methods.

## 3.1 Environments

MAPF algorithms are generally evaluated on MAPF benchmark suite[2], which provides 2D grid maps of different layouts simulating various real-world environments, such as warehouses and empty rooms. Among many maps in the suite, we choose six representative maps (i.e., commonly chosen maps in the aforementioned papers and cover diverse layouts) from each MAPF benchmark category: empty-32-32 of size $32 \times 32$ (denoted as empty), random-32-32-20 of size $32 \times 32$ (denoted as random), warehouse-10-20-10-2-1 of size $161 \times 63$ (denoted as warehouse), ost003d of size $194 \times 194$, den520d of size $256 \times 257$, and Paris_1_256 (denoted as Paris) of size $256 \times 256$. We utilize the '25 random scenarios' included in the suite, where each scenario offers a distinct set of agent start and goal locations for a given map and specified number of agents. For methods requiring training data, e.g., SVM-LNS and Neural-LNS, we generate additional scenarios using the same map layouts with new random start-goal pairs for model training, such that all methods are evaluated on the same 25 scenarios in the benchmark.

[1]Code and data are available at: https://github.com/ChristinaTan0704/mapf-lns-unified

[2]https://movingai.com/benchmarks/mapf/index.html

## 3.2 Initial Solution

As an anytime algorithm, we would expect MAPF-LNS to quickly find an initial feasible solution and then improve its quality to near-optimal as time progresses. Therefore, we set the time limit for finding the initial solution to 10 seconds by following Li et al. (2021a).

Three representative suboptimal MAPF solvers, discussed by Li et al. (2021a), are considered as potential initial solvers: Explicit Estimation CBS (EECBS) (Li, Ruml, and Koenig 2021), Prioritized Planning (PP) (Erdmann and Lozano-Perez 1987) with a random priority ordering, and Parallel Push and Swap (PPS) (Sajid, Luna, and Bekris 2012). However, as highlighted by Li et al. (2021a) (see Fig 3 of Li et al. (2021a)), none of the three successfully solve all 25 scenarios across varying numbers of agents within the time limit, yielding unsolved scenes being discarded. To ensure that initial solutions are available for all 25 scenarios within 10s, we adopt the following two methods as initial solvers.

1) LNS2 (Li et al. 2022). It is an improved version of PP. It repeatedly repairs the collisions met by PP until the paths become collision-free. 2) LaCAM2 (Okumura 2023). It was recently proposed as a fast suboptimal MAPF method. Though it is faster than LNS2, its solution quality is generally worse than LNS2.

## 3.3 Replan Solver

The replan solver is invoked iteratively to update paths for the neighborhood and refine the solution in real-time, making a fast and efficient solver highly desirable. Except for Yan and

| Initial Solver: LNS2; Time limit: 60s | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | Final Delay | | Iter (x1k) | | | N | Final Delay | | Iter (x1k) | |
| | | PP | PBS | PP | PBS | | | PP | PBS | PP | PBS |
| empty | 300 | 431.7 | 436.9 | 8.98 | 0.44 | random | 150 | 350.1 | 346.9 | 7.39 | 0.63 |
| | 350 | 1109.8 | 1081.8 | 4.22 | 0.25 | | 200 | 959.6 | 875.5 | 2.88 | 0.19 |
| | 400 | 2570.1 | 2238.2 | 2.28 | 0.15 | | 250 | 2423.8 | 2301.4 | 1.99 | 0.05 |
| | 450 | 4873.5 | 4293.6 | 1.83 | 0.09 | | 300 | 5309.6 | 4533.1 | 1.47 | 0.03 |
| | 500 | 7817.6 | 6874.2 | 1.51 | 0.05 | | 350 | 8966.9 | 8076.5 | 1.57 | 0.02 |
| warehouse | 150 | 122.1 | 128.3 | 6.59 | 0.57 | ost003d | 200 | 183.9 | 897.6 | 1.75 | 0.06 |
| | 200 | 266.8 | 310.2 | 2.65 | 0.27 | | 300 | 915.5 | 4630.9 | 0.92 | 0.02 |
| | 250 | 477.6 | 760.3 | 1.83 | 0.15 | | 400 | 3230.7 | 8032.7 | 0.52 | 0.03 |
| | 300 | 832.7 | 1740.3 | 1.15 | 0.09 | | 500 | 9335.3 | 16709.3 | 0.21 | 0.01 |
| | 350 | 1495.0 | 3237.5 | 0.73 | 0.06 | | 600 | 17998.3 | 24525.7 | 0.15 | 0.01 |
| den520d | 500 | 899.6 | 6195.8 | 1.28 | 0.05 | Paris | 350 | 82.2 | 383.7 | 5.98 | 0.17 |
| | 600 | 1321.3 | 8485.5 | 1.72 | 0.06 | | 450 | 136.5 | 2274.2 | 6.44 | 0.11 |
| | 700 | 4436.5 | 16642.9 | 0.78 | 0.02 | | 550 | 219.3 | 4878.6 | 4.72 | 0.06 |
| | 800 | 7342.8 | 21909.0 | 0.61 | 0.02 | | 650 | 317.1 | 9304.6 | 4.49 | 0.04 |
| | 900 | 13032.0 | 29352.2 | 0.44 | 0.01 | | 750 | 614.9 | 14707.1 | 3.07 | 0.03 |

Table 1: Final delay and total iteration of using PP and PBS within 60s when initial solver is LNS2, neighborhood selection strategy is RandomWalk, and neighborhood size is 25. Cases where PBS performs better are highlighted in red. *Note*: *'N'* denotes the number of agents.

Wu (2024), most methods typically select PP as the replan solver due to its fast speed in completing a single iteration (e.g., faster than CBS and EECBS (Li et al. 2021a)), allowing for more iterations within the time limit. However, Yan and Wu (2024) argues that while Priority-Based Search (PBS) is more time-consuming per iteration, it can outperform PP in certain scenarios due to its greater improvement in a single iteration. To evaluate the efficiency of PP and PBS, we set the neighborhood size to 25 and the time limit to 60s (neighborhood size 25 is recommended in 3/5 cases by Yan and Wu (2024), and 60s is used in their plots). The time-counting criterion for evaluation is detailed in Sec. 3.5. The initial solver used is LNS2, with RandomWalk as the heuristic.

The final delays in different maps with various amount of agents by using PP and PBS are shown in Table 1. Though our results coincide with Yan and Wu (2024) that PBS is better in empty and random maps, it is significantly worse than PP in larger maps with more agents. We also include the number of iterations performed by PP and PBS in the table. We see that PP runs notably faster than PBS and thus can explore a substantially larger number of neighborhoods within the time limit. Therefore, we choose PP as the replan solver. (More comparison results between PP and PBS are shown in Table 11 and Table 12 in Appendix, which include cases where the initial solver is LaCAM2, and the time limit is 300s. These results also suggest that PP is better than PBS in most cases.)

### 3.4 Neighborhood Size and Number of Agents

Intuitively, a smaller neighborhood size accelerates each iteration but may yield limited improvements, whereas a larger neighborhood size slows down each iteration but has the potential to deliver more significant improvements. To evaluate those strategies, we test a variety of neighborhood sizes, i.e., $\{2, 4, 8, 16, 32\}$, by following Li et al. (2021a).

The number of agents in a map affects the congestion level of a MAPF problem. We select a broad spectrum of agent amounts in each map to encompass the range of numbers evaluated in previous papers. The number of agents evaluated for each map is summarized in Table 6 in Appendix.

### 3.5 Evaluation Criterion

Given the time-sensitive nature of MAPF-LNS, we mainly focus on the relationship between delay and time. Specifically, we report the **final delay** and **area under the curve** (AUC) of the delay-versus-time curve within a specified time limit. A common criterion is a time limit of 60s. However, we observe that the delay may not converge within 60s. To address this, we extend the time limit to 300s (we still report the results when the time limit is 60s). To reduce the influence of overhead from other operations, we only measure the time spent on the core processes of each method, i.e., the time used for destroying (remove agents) and repairing (replan paths). Note that SVN-LNS and Neural-LNS require calling the rule-based methods to propose neighborhood candidates and calling the trained model to predict the top neighborhood. This part of time is included in the time used by these two methods. The model prediction of Neural-LNS is performed on GPU to accelerate.

### 3.6 Implementation Details

We develop based on the codebase of Li et al. (2021a) to produce results for rule-based strategies, including RandomWalk, Intersection, Random, Adaptive, and RandomWalkProb. We use the codebase of Phan et al. (2024) to produce results for Bandit, where Thompson Sampling is the underlying bandit algorithm since it performs the best in our experiments. No open-source codebase is available for SVM, and neither executable models nor training data are provided for NNS. As a result, we implement and train both methods. Note that the original SVM and NNS use different initial solutions for different maps and are not clearly specified. We fix the initial solution to LNS2 (Li et al. 2022) when recovering their results. Please refer to Sec. 7 in Appendix for training details, e.g., dataset construction and hyperparameter tuning.

To validate the reliability of our reproductions of SVM and NNS, we first conduct the following sanity checks. *1) Matching the reported statistics.* We reproduce SVM and NNS by training models using instructions and parameters provided in their original papers. The time measurement scheme is not clearly detailed in Huang et al. (2022); Yan and Wu (2024). Thus, we compare the delay-versus-iteration of our reproduced results with reported ones, which is independent of hardware specifications and time-counting schemes. For NNS, we compare the reduction in delays after 100 iterations, starting from roughly the same initial delays in shared maps (we use PBS as the replan solver in this comparison for consistency with Yan and Wu (2024)). The results of our trained NNS and reported performances are shown in the bottom of Table 2. The maximum discrepancy in performance is only 8.9 steps per agent in `den520d`, and our reproduced model performs better in `empty` and `warehouse`, con-

**Average Rank of SVM**

| | | Rank / Total | |
|---|---|---|---|
| **Map** | **N** | **Reproduce** | **Report** |
| warehouse | 100 | 5.9/20 | 6.0/20 |
| ostd003d | 100 | 6.2/20 | 5.4/20 |
| den520d | 200 | 7.6/20 | 7.0/20 |
| Paris | 250 | 7.7/20 | 6.7/20 |

**Reduction in delays of NNS**

| | | Initial delays | | Delay elimination | |
|---|---|---|---|---|---|
| **Map** | **N** | **Reproduce** | **Report** | **Reproduce** | **Report** |
| empty | 300 | 3140.7 | 3200 | 2202 | 1850 |
| warehouse | 100 | 5731.8 | 5500 | 4908 | 4700 |
| ost003d | 100 | 8690.4 | 8600 | 4585.1 | 5000 |
| den520d | 200 | 20 510.2 | 20 500 | 12 721.3 | 14 500 |

Table 2: Reproduced and reported metrics of SVM (top) and NNS (bottom). *Note*: *'N'* is the number of agents. *'Reproduce'* is our reproduced results. *'Report'* is reported values of SVM (Huang et al. 2022) and NNS (Yan and Wu 2024).

firming the reliability of our reproduction. For SVM, delay-versus-iteration statistics are not provided, so we compare the "average ranking". Average ranking defined by Huang et al. (2022) is the average rank of the predicted top neighborhood out of 20 candidates. The results, shown in the top of Table 2, reveal that the discrepancy in average rank is no greater than one, further validating our reproduction. *2) Generalization ability*. It is costly and impossible to train separate models for different parameters, such as the number of agents or neighborhood size. Instead, we wish to train models under one parameter setting and evaluate their performance by generalizing to unseen configurations, a goal also emphasized in their original papers. To assess generalization, we conduct experiments comparing the performance of two models evaluated under the same setting but trained with one key parameter altered. The key parameters include neighborhood size, selection strategy, number of agents, and initial solver. We observe that the final delays produced by the generalized models (trained and tested under different configurations) are generally comparable to those of the ungeneralized models (trained and tested under the same configuration). See Table 10 in the appendix for detailed numerical results.

## 4 Findings and Future Directions

We perform evaluation for aforementioned approaches under the unified framework outlined in Sec. 3, which reveals six key insights that challenge existing research perspectives and highlight four promising directions for future work.

The results are mainly presented as running curves and value tables. For simplicity and clarity, we present representative results when the initial solver is LNS2, with a focus on the highest number of agents in each map, since those are the most congested and challenging cases. We also present cases with a medium number of agents, which are used to train SVM and NNS. Including the case of medium number makes it easier to access the performance of SVM and NNS. The results in other settings reflect similar observations and are deferred to Sec. 8.4 in Appendix. We also include the results of the reproduced SVM and NNS when they are trained according to their original papers. We add a prefix 'Ori-' (e.g., Ori-SVM, Ori-NNS) to distinguish with SVM and NNS trained under our unified setting.

For rule-based methods, the results are obtained using the best-performing neighborhood size. For evaluating SVM and NNS, we select the optimal combination of heuristic and neighborhood size for proposing neighborhood candidates. For example, in `empty` map with 500 agents, RandomWalk with neighborhood size 8 achieves the lowest delay within 300s. Consequently, we use RandomWalk to generate neighborhood candidates for SVM and NNS, and employ neighborhood size 8 during execution in that scenario. For evaluating Ori-SVM, and Ori-NNS, we adhere to the heuristic and neighborhood size specified in their original papers.

### 4.1 Key Insights and Analyses

**1) Rule-based strategies are strong competitors to learning-based strategies in terms of time efficiency**. Note that the final delays we report for rule-based strategies are

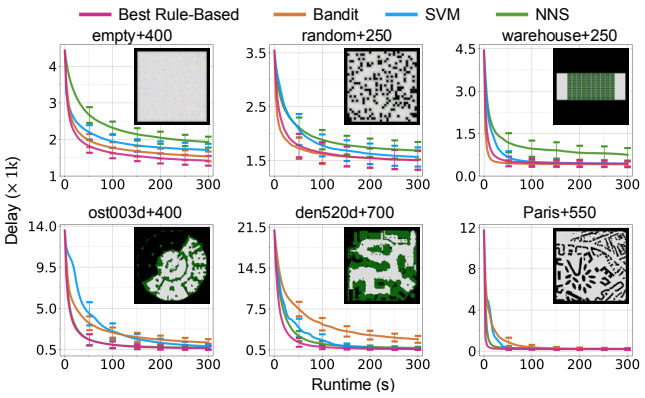


Figure 1: **Delay vs. Time in maps with a Medium number of agents**. Error bars represent the variance of delay across 25 different scenes. The best rule-based strategies are Adaptive for `empty` and `warehouse`, RandomWalk for `random`, and RandomWalkProb for `ost003d`, `den520d`, and `Paris`.

generally lower than those in previous studies, as we only measure the time spent on core processes. This same time measurement scheme is applied to learning-based strategies, ensuring a fair comparison. Table 4 presents the final delays and AUCs of delay-versus-time curves when the time limits are 300s and 60s (with the highest number of agents in each domain). Rule-based strategies achieve the best final delays in $83.3\%$ (20/24) of the cases. To minimize the impact of generalizing to untrained scenarios on the performance of SVM and NNS, we also investigate their performance on maps with a medium number of agents where training data are collected. The delay-versus-time curves for these scenarios are shown in Figure 1. For clarity, only the curves of the best-performing rule-based strategies are included. NNS matches the efficiency of the best rule-based strategy only in the `ost003d` map. In other domains, SVM and NNS are generally slower.

Our findings contradict those of learning-based studies. For example, Huang et al. (2022) claimed that SVM-based neighborhood prediction outperforms rule-based methods in terms of time efficiency. However, under our unified evaluation framework, SVM shows no clear advantage over rule-based ones. Additionally, Phan et al. (2024) directly adopted the results from Huang et al. (2022) and suggested that Bandit outperforms SVM. In our evaluation, however, Bandit is better than SVM in some cases but worse in others. We also notice that the final delays for rule-based methods reported in Huang et al. (2022) are significantly higher than those in Li et al. (2021a), and their time measurement schemes are not clearly described. This raises the possibility that rule-based strategies were under-reported or evaluated using inconsistent time measurement schemes. Yan and Wu (2024) focused solely on cases where PBS outperforms PP and aimed to improve these cases using deep learning (though our evaluation shows PP is generally better than PBS). Consequently, their application scope is limited. When applied to diverse scenarios in our evaluation, NNS does not demonstrate faster

performance than rule-based approaches.

**2) SVM and NNS incur high overheads compared to PP replanning**. To understand the time inefficiency of SVM and NNS, we analyze their additional overheads compared to rule-based methods. In each iteration, SVM and NNS introduce two main additional sources of overheads: *1) Proposition*, which involves generating a set of neighborhood candidates using rule-based strategies (20 candidates in SVM, and 100 in NNS); *2) Prediction*, which predicts the best neighborhood using trained ranking models. Table 3 summarizes the percentage of these overheads in total time used for maps with the highest number of agents.

The proposition overhead in SVM is negligible due to the small number of candidates (i.e., 20), but it increases greatly in NNS due to a larger candidate pool (i.e., 100). The prediction overhead is notably high for both SVM and NNS. This is because PP replanning is super fast per iteration, making the prediction speed of the machine learning models the bottleneck. We also report the overheads of Ori-NNS when PBS is used for replanning. In this case, the proposition and prediction overheads in NNS become relatively small because PBS takes a longer time to execute. This observation supports Yan and Wu (2024)'s approach of applying deep learning only in cases where PBS performs better, as the neural network overhead becomes acceptable with PBS replanning. However, this further highlights the limited application scope of NNS.

**3) The improvement capacity of supervised learning methods per iteration is limited**. As discussed above, SVM and NNS introduce high time overheads. Here, we investigate their improvement capacity per iteration, which is independent of the time overheads. We compare them with the best rule-based strategies using the *delay-versus-iteration* criterion in maps with medium number of agents. Note that the best rule-based strategies in these scenarios are employed to generate training data for SVM and NNS. The performance curves are shown in Figure 2.

SVM and NNS aim to predict and select the best neighborhood from the candidate pool for replanning in each iteration. However, as illustrated in Figure 2, only in `empty` and `den520d`, SVM is able to predict a better neighborhood than the one picked by rule-based methods. In other cases, the improvement capacity of supervised learning models fails to surpass that of rule-based strategies. Also, as indicated in Table 2, the trained ranking models struggle to accurately select the ground truth best neighborhood. This suggests that

| Methods | empty +500 | | random +350 | | warehouse +350 | | ost003d +600 | | den520d +900 | | Paris +750 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prop | Pred | Prop | Pred | Prop | Pred | Prop | Pred | Prop | Pred | Prop | Pred |
| SVM | 0.4 | 41.4 | 3.2 | 38.2 | 0.4 | 33.4 | 1.9 | 33.5 | 1.7 | 45.0 | 1.5 | 35.1 |
| NNS | 7.5 | 53.2 | 11.3 | 46.8 | 16.7 | 47.6 | 7.6 | 13.1 | 16.3 | 27.8 | 34.8 | 29.3 |
| Ori-NNS | 8.4 | 1.4 | 4.7 | 3.4 | 0.8 | 0.6 | 0.8 | 1.9 | 0.4 | 0.7 | 2.3 | 1.4 |

Table 3: Percentage of proposition and prediction overheads in SVM and NNS. Initial solver is LNS2. *Note*: Numbers are shown in percentage (%). *'Prop'* represents *proposition*. *'Pred'* represents *prediction*.

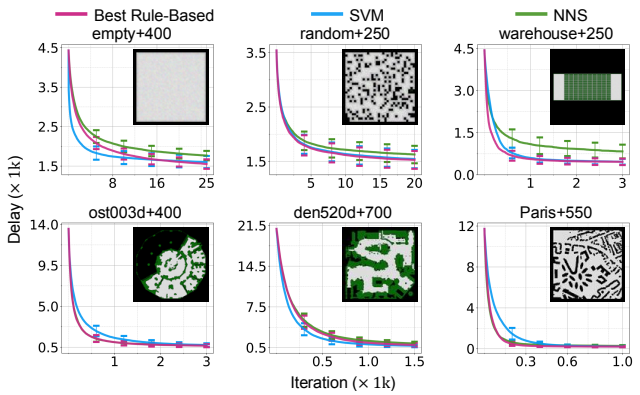


Figure 2: **Delay vs. Iteration in maps with a Medium number of agents**. Error bars represent the variance of delay across 25 different scenes. The best rule-based strategies are Adaptive for `empty` and `warehouse`, RandomWalk for `random`, and RandomWalkProb for `ost003d`, `den520d`, and `Paris`.

achieving a clear advantage over rule-based strategies requires a more powerful neighborhood ranking model with higher prediction accuracy.

**4) Under-performance of Bandit vs. Adaptive**. Bandit is another learning-based method, but it employs reinforcement learning instead of supervised learning. As a result, it requires no training data and dynamically identifies the best rule-based strategies in each iteration during execution. This allows us to view it as another variant of Adaptive, where bandit logits serve as weights for selecting rule-based strategies.

When comparing Bandit and Adaptive in Table 4, Bandit performs worse than Adaptive in 66.6% cases (16/24) in terms of final delays. The performance gap between them is particularly significant in map `den520d`. As discussed in Sec. 2.3, Bandit uses a non-contextual algorithm, whereas a contextual algorithm is more appropriate, which may explain its under-performance. However, we also observe that with a short time limit, i.e., 60s, Bandit achieves the lowest delay and AUC in maps like `random` and `warehouse`. This suggests that properly switching among rule-based strategies and neighborhood sizes can accelerate MAPF-LNS.

**5) RandomWalk (with its variant RandomWalkProb) shows robust performance across diverse scenarios**. The complete results for final delays across various maps and number of agents are provided in Sec. 8.4 in Appendix. In the majority of cases, RandomWalk (with its variant RandomWalkProb) achieves the best final delays. Even in scenarios where it does not rank first, its performance remains close to the best. This observation slightly contrasts with the findings of Li et al. (2021a), where Adaptive outperformed RandomWalk in more than half of the cases. While the time measurement scheme in Li et al. (2021a) is not clearly specified, under our unified evaluation framework, RandomWalk and its variant exhibit a clear advantage over other methods.

**6) Quality of the initial solution is not highly critical**. In our experiments, we study two initial solvers: LNS2 and LaCAM2. LaCAM2 is faster but generally produces low-quality

| Methods | Highest Number of Agents; Time: 300s | | | | | | | | | | | | Medium Number of Agents; Time: 300s | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | empty +500 | | random +350 | | warehouse +350 | | ost003d +600 | | den520d +900 | | Paris +750 | | empty +400 | | random +250 | | warehouse +250 | | ost003d +400 | | den520d +700 | | Paris +550 | |
| | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC | Delay | AUC |
| RW | 4050.5 | 145.4 | 4439.2 | 150.4 | 1041.8 | 41.3 | 6069.5 | 318.3 | 2290.1 | 166.7 | 404.7 | 25.4 | 1397.2 | 49.1 | 1504.5 | 50.4 | 433.1 | 15.3 | 761.8 | 48.3 | 934.1 | 80.3 | 203.6 | 9.1 |
| INT | 4205.4 | 143.9 | 4609.3 | 155.5 | 1949.2 | 84.6 | 8824.9 | 404.8 | 6776.0 | 347.7 | 1680.8 | 110.9 | 1513.7 | 53.7 | 1582.6 | 53.2 | 695.2 | 32.8 | 2048.3 | 100.6 | 3415.0 | 175.9 | 540.8 | 38.3 |
| RAND | 4438.9 | 155.5 | 4635.2 | 167.8 | 1515.9 | 66.2 | 8424.3 | 399.2 | 6447.9 | 345.6 | 1211.7 | 99.1 | 1703.2 | 59.6 | 1606.3 | 53.3 | 537.7 | 23.4 | 1668.8 | 86.3 | 2907.4 | 171.7 | 345.4 | 33.7 |
| ADP | 4093.8 | 140.4 | 4432.8 | 150.1 | 1073.0 | 44.1 | 6325.9 | 323.0 | 2611.1 | 187.9 | 396.9 | 33.7 | 1418.6 | 50.0 | 1527.5 | 50.6 | 435.6 | 16.2 | 746.1 | 51.0 | 1049.8 | 88.0 | 192.2 | 11.7 |
| RWP | 4051.3 | 143.9 | 4408.2 | 156.6 | 1134.9 | 44.7 | 4731.3 | 260.7 | 1387.2 | 130.6 | 375.6 | 21.8 | 1431.1 | 50.1 | 1534.4 | 50.9 | 443.8 | 16.4 | 652.3 | 39.3 | 620.2 | 48.3 | 183.8 | 7.4 |
| SVM | 5053.2 | 167.1 | 4800.2 | 175.8 | 1107.4 | 65.7 | 10104.8 | 459.1 | 1816.6 | 216.5 | 388.7 | 65.3 | 1588.3 | 64.5 | 1713.9 | 59.1 | 439.6 | 19.3 | 850.3 | 84.3 | 665.2 | 77.9 | 184.7 | 16.5 |
| NNS | 4803.2 | 179.7 | 5466.7 | 193.5 | 1871.1 | 77.1 | 5501.2 | 294.6 | 1821.4 | 168.1 | 468.2 | 36.6 | 1928.5 | 70.0 | 1683.7 | 57.5 | 749.0 | 31.4 | 679.5 | 41.2 | 814.0 | 68.2 | 217.5 | 12.2 |
| Bandit | 4318.5 | 150.8 | 4564.1 | 155.8 | 1047.6 | 38.1 | 6093.6 | 311.4 | 5343.4 | 308.8 | 576.9 | 70.6 | 1537.2 | 54.2 | 1507.3 | 49.3 | 414.2 | 14.1 | 1276.3 | 75.2 | 2297.3 | 150.2 | 205.3 | 20.7 |
| Ori-SVM | 4776.3 | 174.7 | 5105.2 | 186.3 | 1100.9 | 64.6 | 8604.1 | 457.2 | 6161.2 | 420.2 | 483.2 | 126.6 | 1640.4 | 63.9 | 1662.8 | 58.3 | 441.0 | 21.1 | 1086.3 | 106.5 | 1880.1 | 200.1 | 196.5 | 37.6 |
| Ori-NNS | 5305.6 | 186.4 | 6004.1 | 206.3 | 1094.5 | 56.1 | 11426.4 | 517.9 | 13367.1 | 624.8 | 2409.4 | 243.5 | 1464.8 | 56.8 | 1528.0 | 54.8 | 417.4 | 17.7 | 1483.2 | 109.0 | 4228.7 | 286.7 | 721.5 | 67.7 |
| | Highest Number of Agents; Time: 60s | | | | | | | | | | | | Medium Number of Agents; Time: 60s | | | | | | | | | | | |
| RW | 4949.4 | 34.9 | 5280.3 | 37.7 | 1331.4 | 14.1 | 13845.4 | 110.0 | 7159.6 | 84.2 | 559.5 | 14.5 | 1726.2 | 13.2 | 1744.9 | 12.5 | 472.7 | 4.5 | 1785.4 | 24.3 | 2583.3 | 38.4 | 216.2 | 4.1 |
| INT | 5117.5 | 35.9 | 5456.2 | 38.4 | 3346.2 | 26.9 | 16830.7 | 127.1 | 14664.4 | 127.0 | 4604.3 | 49.8 | 1900.5 | 14.3 | 1862.1 | 13.2 | 1333.8 | 11.7 | 3975.2 | 38.2 | 7100.6 | 68.3 | 1494.2 | 19.8 |
| RAND | 5376.6 | 36.2 | 5416.5 | 37.4 | 2420.4 | 23.5 | 16765.7 | 127.9 | 14546.4 | 124.7 | 4620.3 | 45.2 | 2105.9 | 15.2 | 1841.5 | 13.1 | 791.7 | 9.1 | 3482.2 | 34.5 | 7256.5 | 67.9 | 1485.0 | 18.7 |
| ADP | 4997.9 | 35.0 | 5290.9 | 37.2 | 1463.4 | 15.4 | 14271.5 | 108.2 | 8506.6 | 89.3 | 689.1 | 16.8 | 1768.8 | 13.3 | 1762.5 | 12.3 | 488.3 | 5.2 | 2013.8 | 25.3 | 3317.7 | 43.1 | 236.4 | 5.2 |
| RWP | 5024.8 | 35.5 | 5224.1 | 37.1 | 1455.6 | 14.3 | 11123.2 | 99.2 | 4547.5 | 67.6 | 549.0 | 10.7 | 1765.6 | 13.4 | 1761.4 | 12.5 | 534.6 | 5.1 | 1127.0 | 16.9 | 1498.7 | 28.9 | 200.6 | 2.9 |
| SVM | 6565.9 | 43.7 | 6543.7 | 46.4 | 3105.0 | 31.6 | 18765.4 | 130.6 | 10998.2 | 111.5 | 2267.7 | 50.0 | 2584.0 | 19.7 | 1990.0 | 14.8 | 600.5 | 8.0 | 4096.8 | 44.1 | 3562.8 | 46.6 | 243.5 | 11.8 |
| NNS | 6355.9 | 43.2 | 6877.1 | 46.4 | 2677.8 | 22.7 | 12267.2 | 102.9 | 6444.6 | 85.6 | 907.1 | 20.6 | 2461.4 | 18.1 | 2049.3 | 14.6 | 1018.5 | 9.4 | 1338.2 | 19.5 | 2115.6 | 38.8 | 292.2 | 5.8 |
| Bandit | 5473.0 | 38.0 | 5540.1 | 38.5 | 1216.4 | 12.0 | 13217.6 | 106.2 | 13552.1 | 115.6 | 3220.9 | 41.1 | 1916.4 | 14.3 | 1699.9 | 11.7 | 446.8 | 4.0 | 3127.5 | 31.4 | 6821.9 | 63.3 | 692.2 | 13.9 |
| Ori-SVM | 6566.5 | 44.6 | 7113.9 | 47.7 | 2971.2 | 30.5 | 20754.1 | 140.7 | 20674.6 | 153.4 | 7769.9 | 82.1 | 2416.5 | 18.9 | 2176.5 | 14.8 | 657.2 | 10.0 | 5713.4 | 53.8 | 11421.6 | 93.5 | 1628.3 | 29.6 |
| Ori-NNS | 6908.0 | 46.0 | 7514.5 | 48.9 | 2182.7 | 24.1 | 21677.9 | 144.8 | 26155.1 | 172.3 | 12831.9 | 97.3 | 2127.1 | 17.3 | 1973.9 | 14.6 | 531.7 | 7.0 | 5246.7 | 49.7 | 13704.0 | 102.6 | 3342.0 | 39.5 |

Table 4: Final delays and AUC across methods in maps with the highest and medium number of agents. Time limits are 300s and 60s, respectively. Initial solver is LNS2. *Note*: *RW*, *INT*, *RAND*, *ADP*, and *RWP* stand for *RandomWalk*, *Intersection*, *Random*, *Adaptive*, and *RandomWalkProb*, respectively. The number of agents follows the name of a map, i.e., after "+". Highlighted are the results ranked the first, and second.

solutions compared to LNS2 (see Table 5 for a comparison of their initial delays in maps with the highest number of agents). However, running various LNS methods starting from LaCAM2 initial solutions yields final delays similar to those starting from LNS2 initial solutions. The discrepancies in final delays for RandomWalkProb between LNS2 and LaCAM2 within 300s are consistently less than 350 across all maps. Even under a 60s limit, the discrepancies remain below 770, except for map `ost003d`. We also examine the final delays when using EECBS as the initial solver. Although EECBS provides higher-quality initial solutions than LNS2, it fails to solve all instances within 10s. Nevertheless, the differences in final delays for RandomWalkProb between LNS2 and EECBS within both 300s and 60s are consistently below 400. These findings suggest that delays decrease rapidly in the early stages of the LNS process. Consequently, fast and scalable solvers like LaCAM2 and LNS2, which can solve most MAPF instances with many agents, should be preferred even if their initial solutions are of lower quality (see Tables 17 and 18 in the appendix for numerical results on LaCAM2 and EECBS).

| Solver | empty +500 | random +350 | warehouse +350 | ost003d +600 | den520d +900 | Paris +750 |
|---|---|---|---|---|---|---|
| LNS2 | 8724.2 | 9305.4 | 8020.1 | 26806.3 | 31463.2 | 20460.5 |
| LaCAM2 | 13058.5 | 14969.3 | 22804.4 | 38632.1 | 51204.5 | 31249.1 |

Table 5: Delays of the initial solutions found by LNS2 and LaCAM2 in maps with the highest number of agent.

## 4.2 Outlooks on Improving MAPF-LNS

Our evaluation within the unified framework reveals that current learning-based methods do not exhibit a clear advantage over rule-based strategies in terms of time efficiency or improvement capacity. This is primarily due to high time overheads, inaccurate predictions, or the use of inappropriate algorithms in these methods. Nevertheless, our comprehensive analysis indicates several promising future directions for improving MAPF-LNS.

**1) Properly targeting high-delayed agents.** The core idea of RandomWalk is to optimize high-delayed agents in each iteration. The superior performance of RandomWalk and RandomWalkProb over others suggests that improving high-delayed agents is an efficient empirical heuristic. Intuitively, focusing on high-delayed agents aligns with the principles of greedy algorithms, which are widely recognized as powerful tools in combinatorial optimization (Papadimitriou and Steiglitz 1998). They provide efficient solutions to complex problems by making locally optimal choices at each step and strike a balance between solution quality and efficiency, espe-

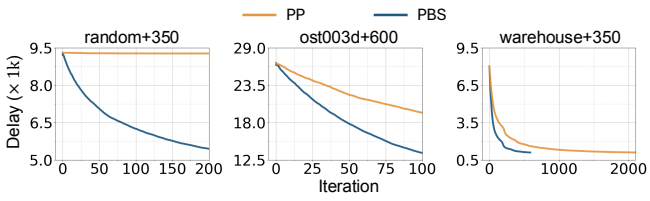


Figure 3: **Delay vs. Iteration when the replan solver is PBS or PP**. The neighborhood selection strategy is RandomWalk with a neighborhood size of 25 in all cases.

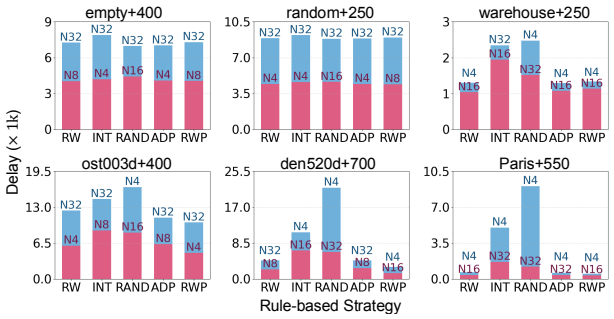


Figure 4: **Final Delays using the Best and Worst neighborhood size within** 300s. RW, INT, RAND, ADP, and RWP stand for RandomWalk, Intersection, Random, Adaptive, and RandomWalkProb. The blue and pink columns indicate the highest and lowest delays. The neighborhood sizes are labeled at the top of each subfigure.

cially for NP-hard problems. Different algorithm designs of RandomWalk and RandomWalkProb result in marginally different performance in experiments, other viable approaches for targeting high-delayed agents can be explored.

**2) Contextual bandit for sequential decision-making.** We observe that the supervised learning methods, e.g., SVM and NNS, incur high time overheads. In contrast, Bandit alternates among rule-based strategies in each iteration as sequential decision-making with minimal computational overhead. This makes Bandit particularly suitable for integration into the MAPF-LNS framework. Although Bandit employs an inappropriate non-contextual bandit algorithm, it performs well in two domains under a 60s time limit. This suggests that a well learned policy for selecting the best rule-based strategies at each time step can significantly enhance MAPF-LNS. Therefore, exploring contextual bandit algorithms is a promising direction. Contextual bandits can address the theoretical limitations of non-contextual approaches by incorporating contextual information into decision-making, potentially leading to better and robust empirical results.

**3) Learning the priority order of replan agents.** We observe that PBS performs better than PP on a per-iteration basis. This is illustrated by the delay-versus-iteration curves shown in Figure 3 for various scenarios. In `random` and `ostd003d`, the disparity in delay elimination between PBS and PP is huge. In `warehouse`, although PP initially reduces delays quickly, it still requires significantly more iterations to achieve the same final delay as PBS. The better per-iteration performance of PBS is due to its strategy of searching for partial priorities among replan agents. In contrast, PP randomly assigns full priority to agents. However, as discussed in Sec. 3.3, PBS struggles with time efficiency when evaluated on a runtime basis, as searching for partial priorities is computationally expensive. This highlights an opportunity for improvement: if a fast learning model can predict a reasonable priority order for replan agents, it can enhance time efficiency while improving solution quality.

**4) Identifying the suitable neighborhood size.** Neighborhood size is a critical factor for MAPF-LNS, but is underexplored in previous papers. Intuitively, a smaller neighborhood size allows for faster iterations but may limit the improvement in solution quality. Conversely, a larger neighborhood size can lead to more improvement per iteration, but at the cost of increased computational time. Thus, there is a trade-off between runtime efficiency and the improvement quality. To highlight the importance of neighborhood size, we compare the final delays achieved by rule-based strategies using the best and worst neighborhood sizes within a 300s time limit, as shown in Fig. 4. Since a neighborhood size of 2 is generally ineffective, we consider sizes from $\{4, 8, 16, 32\}$. In most cases, using the best size reduces final delays by approximately $50\%$ compared to the least favorable size. This difference suggests the potential for performance gains by learning to identify a suitable neighborhood size dynamically.

The only work that attempts to dynamically determine neighborhood size is Bandit. However, as previously discussed, it employs a non-contextual algorithm, which is unsuitable for this purpose. We observe that its performance is similar to selecting neighborhood sizes uniformly at random (we build a baseline, named Uni-Bandit, by modifying the second arm in Bandit to randomly choose neighborhood sizes, results in Table 9 in the appendix validate this similarity). Therefore, a more sophisticated approach, such as a contextual bandit or other advanced methods, is necessary for effective neighborhood size selection.

## 5 Conclusion

In this work, we conducted a comprehensive reevaluation of prominent MAPF-LNS methods, including recent advances leveraging machine learning. We identified several pitfalls in their evaluations and proposed a unified framework to address these challenges. Our results demonstrate that current learning-based methods fail to exhibit a clear advantage over simple rule-based heuristics, while RandomWalk and its variant RandomWalkProb, consistently deliver robust performance across diverse scenarios. Furthermore, our evaluation and extensive experiments highlight promising directions for advancing MAPF-LNS, such as targeting high-delayed agents, employing contextual algorithms for strategy selection, learning replanning agent orders, and dynamically identifying suitable neighborhood sizes. We believe this work will encourage future research to adopt more rigorous experimental designs and inspire innovative approaches to enhancing MAPF-LNS through machine learning.

## References


Chen, Z.; Li, J.; Harabor, D.; and Stuckey, P. J. 2023. Scalable Rail Planning and Replanning with Soft Deadlines. *arXiv preprint arXiv:2306.06455*.

Demir, E.; Bektaş, T.; and Laporte, G. 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. *Eur. J. Oper. Res.*, 223(2): 346–359.

Erdmann, M.; and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica*, 2: 477–521.

Fines, K.; Sharpanskykh, A.; and Vert, M. 2020. Agent-based distributed planning and coordination for resilient airport surface movement operations. *Aerospace*, 7(4): 48.

Huang, T.; Li, J.; Koenig, S.; and Dilkina, B. 2022. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 36, 9368–9376.

Jiang, H.; Zhang, Y.; Veerapaneni, R.; and Li, J. 2024. Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities. In *Proc. Int. Symp. Combinatorial Search (SoCS)*, volume 17, 234–242.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 4127–4135.

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 *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 36, 10256–10265.

Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021b. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*, volume 31, 477–485.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A bounded-suboptimal search for multi-agent path finding. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 35, 12353–12362.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021c. Lifelong multi-agent path finding in large-scale warehouses. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 35, 11272–11281.

Li, J.; Wang, Y.; and McAuley, J. 2020. Time interval aware self-attention for sequential recommendation. In *Proc. Int. Conf. Web Search and Data Mining (WSDM)*, 322–330.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with consistent prioritization for multi-agent path finding. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 33, 7643–7650.

Ma, H.; and Koenig, S. 2017. AI buzzwords explained: multi-agent path finding (MAPF). *AI Matters*, 3(3): 15–19.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T.; and Koenig, S. 2017. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Proc. AAAI Conf. Artif. Intell. Interactive Digital Entertainment (AIIDE)*, volume 13, 270–272.

Ma, Z.; Luo, Y.; and Ma, H. 2021. Distributed heuristic multi-agent path finding with communication. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 8699–8705. IEEE.

Ma, Z.; Luo, Y.; and Pan, J. 2021. Learning selective communication for multi-agent path finding. *IEEE Robot. Autom. Lett.*, 7(2): 1455–1462.

Morris, R.; Pasareanu, C. S.; Luckow, K.; Malik, W.; Ma, H.; Kumar, T. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *Workshops at the Thirtieth AAAI Conf. Artif. Intell. (AAAI)*.

Okumura, K. 2023. Improving LACAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, 243–251.

Papadimitriou, C. H.; and Steiglitz, K. 1998. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Phan, T.; Huang, T.; Dilkina, B.; and Koenig, S. 2024. Adaptive Anytime Multi-Agent Path Finding Using Bandit-Based Large Neighborhood Search. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 38, 17514–17522.

Pisinger, D.; and Ropke, S. 2010. Large neighborhood search. *Handbook of Metaheuristics*, 399–419.

Sajid, Q.; Luna, R.; and Bekris, K. 2012. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *Proc. Int. Symp. Combinatorial Search (SoCS)*, volume 3, 88–96.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. S.; Koenig, S.; and Choset, H. 2019. PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robot. Autom. Lett.*, 4(3): 2378–2385.

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

Song, J.; Yue, Y.; Dilkina, B.; et al. 2020. A general large neighborhood search framework for solving integer linear programs. *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 33: 20012–20023.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proc. Int. Symp. Combinatorial Search (SoCS)*, volume 10, 151–158.

Yan, Z.; and Wu, C. 2024. Neural Neighborhood Search for Multi-agent Path Finding. In *Proc. Int. Conf. Learn. Represent. (ICLR)*.

Yu, J.; and LaValle, S. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 27, 1443–1449.

Yu, L.; Zhang, C.; Liang, S.; and Zhang, X. 2019. Multi-order attentive ranking model for sequential recommendation. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, volume 33, 5709–5716.