

Learning to Select Promising Initial Solutions for Large Neighborhood Search-Based Multi-Agent Path Finding^{*}

Marc Huber¹[0009-0003-2862-6232], Günther R. Raidl¹[0000-0002-3293-177X], and Christian Blum²[0000-0002-1736-3559]

¹ Institute of Logic and Computation, TU Wien, Austria
`{mhuber,raidl}@ac.tuwien.ac.at`

² Artificial Intelligence Research Institute (IIIA-CSIC), Campus of UAB, Spain
`christian.blum@iiaa.csic.es`

Abstract. Anytime Multi-Agent Path Finding (MAPF) is a promising paradigm for finding fast and (near-)optimal solutions to large-scale multi-agent systems within a fixed time budget. The currently leading approach builds on Large Neighborhood Search (LNS), which iteratively optimizes a quickly generated initial solution by repeatedly selecting and replanning paths of subsets of agents using randomized destroy heuristics and Prioritized Planning (PP). In this study, we examine the impact of initial solutions on the quality of final solutions in a state-of-the-art LNS-based anytime MAPF algorithm. Our findings demonstrate that its effectiveness is significantly influenced by the choice of the initial solution. Building on this insight, we propose to run PP many times to create a larger pool of potential initial solutions, from which we then select by means of an offline-trained Machine Learning (ML) model a most promising solution to run the LNS on. Empirical results on well-established MAPF benchmark instances show that the ML model successfully selects a most promising solution from the pool of potential initial solutions. This leads to improved performance of the state-of-the-art LNS-based anytime MAPF method in terms of both the final solution quality and the Area Under the Curve when initiated from the selected solution.

Keywords: Anytime Multi-Agent Path Finding · Machine Learning · Large Neighborhood Search.

1 Introduction

Finding a set of collision-free paths for a group of agents in a shared environment has many important contemporary real-world applications, including automated warehouses [17], robotics, and autonomous vehicles [16]. In computer science, this

^{*} This project is partially funded by the Doctoral Program “Vienna Graduate School on Computational Optimization”, Austrian Science Foundation (FWF), grant W1260-N35.

problem is known as the Multi-Agent Path Finding (MAPF) problem [14]. Desirable solutions to the MAPF problem are those which are conflict-free, but also minimize some objective function. However, finding optimal solutions under various objective functions, such as the sum of costs, has proven to be NP-hard [18], as the state-space grows exponentially with the number of agents. In practice, attempts to solve MAPF instances with a few hundred or more agents using exact or reasonably bounded suboptimal MAPF algorithms typically requires far too much time or memory. In contrast, fast constructive heuristics usually only find low-quality solutions or no feasible solutions in tightly constrained scenarios at all. To address these issues, researchers have explored anytime MAPF approaches [3,7,6] which combine the strengths of both worlds. Anytime algorithms aim to find feasible solutions quickly and then continuously work on improving them as long as time remains, possibly converging to optimality if time allows.

In the context of anytime MAPF algorithms, Large Neighborhood Search (LNS)-based MAPF is a promising approach that relatively quickly finds solutions to the MAPF instances and iteratively improves them to near-optimality by repeatedly selecting and replanning paths of subsets of agents using destroy and repair heuristics [7]. The subset of agent paths selected for replanning is referred to as a neighborhood. If the newly found solution is superior to the incumbent solution, the incumbent solution is replaced by the new solution. This iterative process continues until an allocated time budget is exhausted.

MAPF-LNS [7] is the currently leading LNS-based approach for anytime suboptimal MAPF solving. It first produces an initial solution quickly using an efficient MAPF construction heuristic. Then, MAPF-LNS repeatedly applies randomized destroy heuristics and replans paths of subsets of agents using Prioritized Planning (PP) [12] with random priorities to further improve the solution until the allotted time is exhausted.

While MAPF-LNS has been empirically demonstrated to scale up to large instances with several hundreds of agents, we observed that the quality of the final solution is significantly influenced by the initial solution, even though the quality of the initial solution is not correlated much with the quality of the final solution.

This article proposes ISS-MAPF-LNS, the Initial-Solution-Selecting-MAPF-LNS, which aims to enhance MAPF-LNS’s ability to find high-quality solutions by initiating the LNS from a more promising initial solution that is selected from a larger pool. The novel aspect of ISS-MAPF-LNS is an offline-trained machine learning (ML) model that selects from a pool of initial solutions generated by PP a most promising solution to run the LNS on. In more detail, a LambdaMART [2] model is trained to compute *scores* for the pool of PP-produced initial solutions and we select the most promising solution based on these scores. Thus, the learned scoring function reflects how promising each initial solution candidate is to ultimately get a final solution of highest quality when applying the LNS on it.

To assess the performance of ISS-MAPF-LNS, we conduct a series of experiments on five well-established MAPF benchmark instances. The results demon-

strate that the trained ML model indeed selects a solution being among the most promising ones from the pool of potentially initial solutions, such that the subsequent LNS leads to a lower-cost final solution than MAPF-LNS applied to an average solution. Moreover, the ML model trained on a specific map with 50 agents scales well to the same map and also to unseen maps with hundreds of agents, leading to improved performance over MAPF-LNS on many instances.

The remainder of this article is organized as follows: Section 2 formally introduces the MAPF problem, and Section 3 reviews related work. Section 4 empirically analyzes the impact of initial solutions on the quality of final solutions in MAPF-LNS and presents our ISS-MAPF-LNS. Results of computational experiments are discussed in Section 5. Finally, in Section 6, we conclude and outline promising avenues for future research.

2 Problem Definition

Multi-agent path finding (MAPF) encompasses a wide variety of variants, as outlined in [14]. In this study, we focus on the simplest and most studied variant that considers three key elements: (1) vertex and swapping conflicts, (2) the “stay at target” assumption, and (3) the sum of costs objective function. Time and space are discretized. The input to the MAPF problem is a connected, undirected, unweighted graph $G = (V, E)$, along with a set of m agents $A = \{a_1, \dots, a_m\}$. Each agent $a_i \in A$ has associated a start vertex $s_i \in V$ and a goal vertex $g_i \in V$ (which may coincide). It is assumed that each start vertex is distinct from all other agents’ start vertices and also each goal vertex is distinct from all others. At each discrete time step t , every agent is located at one of the graph vertices and can either move to a neighboring vertex or wait at its current vertex. A path $p_i = (p_{i,0}, \dots, p_{i,l(p_i)})$ for agent a_i is a sequence of neighboring vertices, where $l(p_i)$ is the length of the path p_i . An edge $(p_{i,t}, p_{i,t+1}) \in E$ indicates a move action, while a vertex $p_{i,t} = p_{i,t+1} \in V$ indicates a wait action. The start vertex is $p_{i,0} = s_i$, while the goal vertex is $p_{i,l(p_i)} = g_i$. It is assumed that an agent remaining at its goal vertex is considered as a wait action if and only if it will subsequently move away from its goal vertex in a subsequent time step before finally returning later. The distance between two vertices, $d(x, y)$, is defined as the length of the shortest path from x to y . The delay of path p_i is $\text{delay}(p_i) = l(p_i) - d(s_i, g_i)$. A solution is a set of paths, one for each agent, such that the agents can follow these paths simultaneously without colliding with each other. The objective is to find a solution $P = \{p_i \mid a_i \in A\}$ that minimizes its sum of costs, denoted as $\text{SOC} = \sum_{p_i \in P} l(p_i)$, or, alternatively, its sum of delays, denoted as $\text{SOD} = \sum_{p_i \in P} \text{delay}(p_i)$.

3 Related Work

A fast constructive approach to MAPF is Prioritized Planning (PP) [12], which generates a solution by sorting the agents according to some priority function and subsequently planning a shortest path for each agent iteratively such that

agents with lower priority avoid collisions with the already planned paths of higher-priority agents. While being fast, PP often produces low-quality solutions or does not find any feasible solution in tightly constrained scenarios.

Research on anytime MAPF has enjoyed an increasing interest in recent years, due to its practical relevance in finding (near-)optimal solutions within a reasonable time budget. This family of algorithms is particularly advantageous in dynamic and large-scale environments where computational efficiency is crucial [15].

In its early stages, several A*-based algorithms have been proposed. These approaches achieve the anytime behavior either by repeatedly calling A* for increasing subproblems [13,15] or by iteratively tightening the bounds of focal search [9], as suggested by Cohen et al. [3]. However, these approaches are primarily effective for instances that are not congested, as the complexity and number of potential conflicts increase significantly in densely populated environments, leading to excessive computational overhead.

In a recent publication, Li et al. [7] proposed MAPF-LNS, which effectively scales to large instances with hundreds of agents and represents the current leading anytime MAPF approach to date. Given a MAPF instance, the algorithm first invokes an efficient suboptimal MAPF algorithm to quickly find an initial solution P . Subsequently, in each iteration, it selects paths of a subset of agents $A_s \in A$ using a randomized destroy heuristic, deletes their paths $P_s^- = \{p_i \in P \mid a_i \in A_s\}$ from P , and calls PP with random priorities to find a set of collision-free paths P_s^+ that also do not collide with paths still in P . If the paths in P_s^+ have smaller sum of cost than the paths in P_s^- , MAPF-LNS adds paths P_s^+ to P and P_s^- otherwise. This procedure is repeated until the allocated time budget is exhausted.

Building on the popularity of MAPF-LNS, researchers have proposed several extensions to the original model. Li et al. [8] suggested MAPF-LNS2, which begins with an infeasible solution and repeatedly replans paths of subsets of agents in order to find paths with fewer conflicts. This process is repeated until all paths are conflict-free or the allotted time has elapsed. Huang et al. [4] proposed MAPF-ML-LNS, which learns a ranking function for a collection of paths, $\mathcal{P} = \{P_1^-, P_2^-, \dots, P_n^-\}$, generated by the destroy heuristics in MAPF-LNS, such that replanning increases the solution quality more. Specifically, given an incumbent solution, MAPF-ML-LNS generates a collection of paths \mathcal{P} , using two randomized destroy heuristics. Then, it applies an offline-trained ranking function to \mathcal{P} and replans the paths in descending order of the predicted scores. If a solution with smaller sum of costs is found, it discards \mathcal{P} and continues to the next iteration. In contrast, Lam et al. [6] incorporated MAPF-LNS as primal heuristic in branch-and-cut-and-price. This approach combines the strengths of LNS with branch-and-cut-and-price techniques and allows for more efficient resolution of large-scale MAPF problems while still retaining optimality guarantees. Phan et al. [10] proposed BALANCE, a bi-level multi-armed bandit scheme that dynamically adapts the selection of destroy heuristics and neighborhoods during the search process. This method aims to avoid expensive prior efforts such as

Table 1. Variance and correlation results obtained by MAPF-LNS across various scenarios and maps.

Scenario	random-32-32-20 $m = 150, N = 16$			empty-32-32 $m = 400, N = 8$			ost003d $m = 300, N = 8$		
	σ	μ_σ	r	σ	μ_σ	r	σ	μ_σ	r
1	10.23	14.68	0.21	81.63	70.01	0.39	156.49	133.41	0.05
2	5.99	11.98	0.24	60.04	71.11	0.05	135.85	157.30	0.62
3	6.83	12.06	0.13	71.68	85.46	0.01	58.74	96.42	-0.14
4	8.96	13.35	-0.02	49.08	59.00	0.43	66.23	93.40	0.16
5	11.21	16.58	0.15	93.45	77.15	0.51	62.13	122.33	0.04
6	7.94	13.91	-0.47	83.26	67.35	0.32	139.85	133.44	0.15
7	12.35	21.03	0.06	69.39	79.00	0.20	108.35	154.04	0.27
8	9.89	20.08	-0.16	111.42	96.67	0.67	149.77	139.49	0.24
9	12.05	18.28	0.08	74.31	73.12	-0.01	112.73	141.86	0.07
10	7.32	13.35	-0.45	52.35	66.10	0.11	152.15	165.48	-0.05
11	7.63	15.08	-0.21	81.09	76.11	0.07	107.96	151.27	0.16
12	7.23	16.47	-0.19	94.79	81.45	0.46	23.00	50.97	0.23
13	5.42	10.88	0.02	46.37	68.30	0.32	112.97	134.84	-0.22
14	9.71	15.38	-0.07	66.96	73.13	0.20	104.65	140.33	0.31
15	11.12	19.49	0.10	43.33	66.08	-0.12	94.41	112.23	0.01
16	9.87	17.81	0.12	76.37	66.28	0.49	83.29	109.03	0.11
17	11.24	18.22	-0.14	61.80	65.84	0.32	114.85	157.16	0.22
18	6.13	12.99	-0.04	59.82	66.97	-0.00	79.40	115.59	0.14
19	6.23	12.46	-0.25	50.33	75.82	0.39	44.46	61.49	0.05
20	8.21	12.57	-0.18	69.73	69.43	0.13	128.85	156.66	0.54
21	6.34	9.40	0.16	56.05	68.82	0.41	98.22	120.50	0.22
22	7.66	16.65	0.19	56.08	76.58	0.30	184.45	175.21	0.51
23	8.63	17.79	-0.19	52.39	66.71	0.54	81.83	128.74	0.08
24	6.57	14.47	-0.05	87.53	87.22	0.19	75.14	116.55	-0.08
25	11.83	15.98	0.39	82.12	85.12	0.08	65.23	72.32	0.44

data acquisition, model training, and feature engineering by adjusting strategies on the fly based on real-time performance feedback.

Concerning specifically the impact of initial solutions in LNS on the final solution quality, we are unaware of any specific approach in the literature that utilizes ML to learn how to select promising initial solutions for LNS.

4 Initial-Solution-Selecting LNS for MAPF

Large Neighborhood Search (LNS) proposed by Shaw [11] is a popular meta-heuristic widely used to find near-optimal solutions to hard combinatorial optimization problems within a fixed time budget. It has gained prominence for its ability to escape local optima and explore vast regions of the solution space for a huge number of applications. LNS achieves this by using more appropriate and effective algorithms to find better solutions in a larger neighborhood instead of iterating through all the neighbors.

In contrast to traditional optimization heuristics that often depend heavily on the quality of the initial solution, LNS can exhibit unexpected behavior where poor initial solutions occasionally lead to superior final outcomes compared to starting with high-quality solutions. This phenomenon, akin to the sensitivity to initial conditions observed in chaotic dynamical systems, suggests that the trajectory of the search process in LNS is profoundly influenced by its starting point.

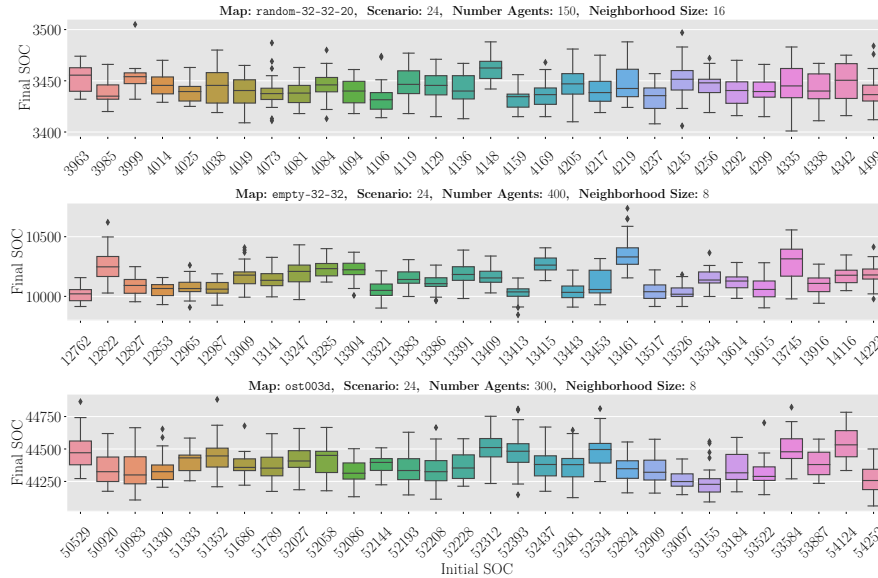


Fig. 1. Impact of initial solutions on the quality of final solutions in MAPF-LNS.

To examine this behavior in the context of anytime LNS-based MAPF, we conduct a series of experiments on the following three grid maps from the MAPF benchmark set [14] with the predefined 25 random scenarios, specifically: **random-32-32-20** with $m = 150$ agents, neighborhood size $N = 16$, **empty-32-32** with $m = 400$ agents, $N = 8$, and **ost003d** with $m = 300$ agents, $N = 8$. For each instance, we generate 30 initial solutions using PP with random priorities. Each initial solution is then subjected to 30 MAPF-LNS runs, with a time limit of 60 seconds and $N = \{8, 16\}$.

Table 1 presents the results of these experiments. They show the overall standard deviation (σ) of final SOCs, the mean of the standard deviations (μ_σ) of final SOCs in relation to the initial solutions, and the correlation coefficient (r) between final SOCs and initial SOCs across the aforementioned three maps and 25 different scenarios.

The correlation coefficient between the initial and final SOCs varies across the different scenarios and maps, with values ranging from as low as -0.47 to as high as 0.67. This wide range of correlation coefficients, often close to zero or even negative, indicates that the quality of the initial solution does not have a strong linear relationship with the quality of the final solution. Specifically:

- For **random-32-32-20**, the correlation coefficients range from -0.47 to 0.39, with many scenarios showing near-zero or negative correlations.
- For **empty-32-32**, the correlation coefficients range from -0.12 to 0.67, also displaying a wide range with several values close to zero.

- For `ost003d`, the correlation coefficients range from -0.22 to 0.62, again indicating a lack of consistent correlation.

An examination of the overall standard deviation of final SOC’s and the mean standard deviation of final SOC’s in relation to initial solutions provides insight into the impact of the initial solutions:

- Across all maps and scenarios, the mean of the standard deviations is generally higher than the overall standard deviation. This pattern indicates that while the final SOC’s exhibit some variability within each group of initial solutions, there is greater variability across the different initial solution groups. This higher variability between groups signifies that the initial solution plays a significant role in determining the range of possible final SOC’s.

In addition to the conducted variance and correlation analysis, a visualization of the results on scenario 24 is presented as box plots in Figure 1. It demonstrates that the quality of the final solution is not correlated much with the quality of the initial solution. Specifically, we observed that initial solutions with lower SOC do not necessarily lead to higher final SOC. These results suggest that there are more intricate relationships between initial solutions and the final solution quality of the LNS, and there is potential to improve the final solution quality of MAPF-LNS by starting the LNS from a more cleverly selected initial solution.

Building on this insight, we use data-driven methods to learn a ranking model for selecting promising initial solutions generated by PP on which to run the LNS on. The key idea is that by warm-starting MAPF-LNS from a more promising initial solution, it can more efficiently and effectively explore the search space, leading to solutions of higher quality. Consequently, we refer to our approach as Initial-Solution-Selecting-MAPF-LNS (ISS-MAPF-LNS).

Algorithm 1 shows a pseudocode for the training of the ranking model for ISS-MAPF-LNS. The input is a set of training instances \mathcal{I} , each consisting of a fixed grid map and a scenario generated by randomly selecting m start and goal vertices on the grid map. For each instance $I \in \mathcal{I}$, the training algorithm generates T initial solutions $(P_I^t)_{t=1, \dots, T}$ using PP with a random priorities.

From each initial solution P_I^t , MAPF-LNS is invoked K times to produce a set of final solutions S . Subsequently, a feature function $\phi : (I, P_I^t) \rightarrow [0, 1]^p$ is applied to derive a p -dimensional feature vector representing meaningful information for P_I^t . This feature vector, along with the median of the objective values of the solutions in S , are stored as training data set D .

Once the training data collection is complete, D is divided into a training set D_{train} and a test set D_{test} . Hyperparameter tuning is then performed on D_{train} using an automated hyperparameter tuning framework that adjusts the parameters θ of the ranking model π_θ . Finally, the model is validated on D_{test} , and the performance, measured by the normalized discounted cumulative gain (nDCG), is recorded.

The feature function $\phi : (I, P_I^t)$ is computed in accordance with Huang et al. [4], where for each agent $a_i \in A$ a set of 16 agent features is considered as listed in

Algorithm 1 Training Algorithm

```

1: Input: Training instance set  $\mathcal{I}$ , nr. of initial solutions  $T$ , nr. of final solutions  $K$ 
2: Output: trained ranking model  $\pi_\theta$ , nDCG
3:  $\pi_\theta \leftarrow$  ranking model
4:  $D, D_{\text{train}}, D_{\text{test}} \leftarrow \emptyset$ 
5: for  $I \in \mathcal{I}$  do
6:   for  $t = 1$  to  $T$  do
7:      $S \leftarrow \emptyset$ 
8:      $P_I^t \leftarrow \text{runPP}(I)$  // generate initial solution
9:     for  $k = 1$  to  $K$  do
10:       $S \leftarrow S \cup \text{runLNS}(I, P_I^t)$  // generate final solution quality
11:    end for
12:     $D \leftarrow D \cup (\phi(I, P_I^t), \text{Median}(S))$  // store features and median of solutions
13:  end for
14: end for
15:  $D_{\text{train}}, D_{\text{test}} \leftarrow \text{Split}(D)$ 
16:  $\pi_\theta \leftarrow \text{HyperparameterTuning}(\pi_\theta, D_{\text{train}})$ 
17: nDCG  $\leftarrow \text{Validate}(\pi_\theta, D_{\text{test}})$ 
18: return  $\pi_\theta$ , nDCG

```

Table 2. Subsequently, the minimum, maximum, sum, and average of each feature value across all agents are also determined, resulting in $4 \times 16 = 64$ features in total. Finally, all 64 feature values are normalized to the range of $[0, 1]$ by applying min-max-normalization.

With regard to the ranking model, LambdaMART [2] is a state-of-the-art learning-to-rank approach that has proven to be highly successful in solving diverse real-world ranking problems. While our primary objective is to accurately predict the most promising initial solution, which also alignment with the objectives of simpler binary classification approaches, we encountered significant challenges when training a binary classifier due to issues related to imbalanced data. These challenges persist even when the problem is relaxed to predicting the top three most promising initial solutions. LambdaMART, however, excels in this context by focusing on the ranking order, thereby mitigating the impact of data imbalance. It emphasizes the correct placement of top solutions, ensuring that the most relevant items are identified and ranked appropriately. This ranking-centric approach enables LambdaMART to overcome the limitations of traditional binary classifiers, providing more accurate and reliable predictions, rendering it suitable for the task at hand.

LambdaMART takes as input a feature matrix, relevance scores, and query IDs, where the query IDs group instances that belong to the same query. To transform our data into this format, we assign each feature vector of an instance $I \in \mathcal{I}$ a unique query ID and convert the solution qualities within each instance to relevance scores ranging from one to T .

Table 2. Features of agent $a_i \in A$ with respect to instance I and solution P_I .

Features	Count
Distance $d(s_i, g_i)$ between the start vertex s_i and the goal vertex g_i of agent a_i .	1
Row and column numbers of the start vertex s_i and goal vertex g_i of agent a_i .	4
Degree of the goal vertex g_i of agent a_i .	1
Delay $\text{delay}(p_i)$ of the agent a_i .	1
Ratio $\text{delay}(p_i)/d(s_i, g_i)$ between the delay of agent a_i and the distance between a_i 's start and goal vertices.	1
The minimum, maximum, sum, and average of the heat values of the vertices along agent a_i 's path p_i . The heat value of a vertex $v \in V$ is the number of time steps that v is occupied by an agent. If agent a_i revisits a vertex multiple times before reaching its goal, the heat value of that vertex is counted multiple times in both the sum and the average.	4
The number of time steps that agent a_i spends on a vertex with degree j ($1 \leq j \leq 4$) before reaching its goal vertex.	4

5 Experimental Evaluation

This section presents a comparative evaluation of the performance of ISS-MAPF-LNS and MAPF-LNS. It begins with a detailed account of the experimental setup, followed by description of the performed model training. Finally, the obtained results are presented and discussed.

5.1 Experimental Setup

We implemented ISS-MAPF-LNS¹ in C++ as an extension of the existing MAPF-LNS2 framework². The gradient boosting framework LightGBM [5] was employed for training LambdaMART [2] models. The evaluation was performed on five representative grid-based maps from the MAPF benchmark suite [14], specifically: **empty-8-8** (8×8), **empty-32-32** (32×32), **random-32-32-20** (denoted as **random**, 32×32), **warehouse-10-20-10-2-1** (denoted as **warehouse**, 161×63), and **ost003d** (194×194). For each map and number of agents, the corresponding 25 predefined random scenarios from the MAPF benchmark suite were utilized to determine the start and goal vertices of the agents. Unless otherwise specified, PP with random priorities was employed for generating initial solutions, and a total runtime limit of 60 seconds along with neighborhood sizes comparable to those used by Li et al. [7] were applied. All experiments were conducted in single-threaded mode on a machine equipped with an AMD EPYC 7402 processor running at 2.80 GHz, with a memory limit of 8 GB.

¹ <https://github.com/isomorphist/ISS-MAPF-LNS>

² <https://github.com/Jiaoyang-Li/MAPF-LNS2>

5.2 Model Training

One of three LambdaMART models was trained on map `random` with $m = 50$, and $N = 16$ by applying Algorithm 1 with $|\mathcal{Z}| = 1000$, and parameters $T = 30$, and $K = 41$. In this process, the dataset \mathcal{D} was randomly split w.r.t. the query IDs into $|\mathcal{D}_{\text{train}}| = 90\%$ training data and $|\mathcal{D}_{\text{test}}| = 10\%$ test data. This ensured that samples with the same query IDs were contained in the same dataset. Hyperparameter tuning for LambdaMART was conducted on the training dataset $\mathcal{D}_{\text{train}}$ using Optuna [1] with the objective of maximizing the nDCG@3 metric and GroupKFold (k=5) cross-validation with groups corresponding to the query IDs. The training procedure was performed in an analogous manner on the map `warehouse` with $m = 150$ and $N = 16$, as well as on the map `random` with $m = 50$ and $N = 8$, resulting in a total of three trained models: $M_{r_{16}}$, $M_{w_{16}}$, and M_{r_8} .

5.3 Testing Procedure

During the testing phase, PP was executed 30 times in ISS-MAPF-LNS to produce a pool of initial solutions, while it was applied only once in MAPF-LNS. The total time required for the generation of initial solutions and the time required for the prediction of scores were subtracted from the total runtime limit of 60 seconds, with the remaining time allocated for running the LNS from the selected initial solution. The following questions were addressed through experimentation:

1. Can we train a ranking model that performs well on the same grid map with the same and different numbers of agents?
2. Can we train a ranking model that generalizes to unseen grid maps with different numbers of agents?





To evaluate these questions, we employed the model $M_{r_{16}}$ in ISS-MAPF-LNS to select promising initial solutions for the maps `random` and `empty-8-8`, the model $M_{w_{16}}$ for the map `warehouse`, and the model M_{r_8} for the maps `ost003d` and `empty-32-32`. For each map and number of agents, 60 ISS-MAPF-LNS runs and 60 MAPF-LNS runs were performed, resulting in a total of 60×25 evaluations for each approach on each map and number of agents.

5.4 Results

Table 3 presents the mean values of the results obtained in our experimental evaluation of ISS-MAPF-LNS (abbreviated as ISS) and MAPF-LNS (abbreviated as LNS), focusing on the following four key metrics:

- **Initial SOD:** Initial Sum of Delays (SOD) of the selected initial solution.
- **Final SOD:** Final SOD after optimization.

Table 3. MAPF-Benchmark results obtained by ISS and LNS.

Map	m	N	Initial SOD		Final SOD		AUC		$t_{\text{init}} [s]$	
			ISS	LNS	ISS	LNS	ISS	LNS	ISS	LNS
empty-8-8	16	16	10.560	10.722	2.520	2.520	151.257	151.262	0.007	0.000
	24	16	34.280	35.057	10.427	10.430	629.894	629.600	0.018	0.001
	32	16	93.760	82.563	25.133	25.185	1545.085	1548.529	0.088	0.003
empty-32-32 	300	8	2372.84	2371.52	440.09	442.37	33518.82	33632.40	2.69	0.09
	350	8	3231.12	3230.97	939.54	942.58	70978.30	71157.98	5.42	0.18
	400	8	4363.44	4354.11	1805.67	1791.05	132203.40	130981.41	11.61	0.37
	450	8	6170.04	6045.67	3134.15	3101.54	219306.41	216669.17	29.69	0.98
ost003d 	100	8	1441.40	1242.43	43.67	41.27	4021.85	3698.54	3.46	0.10
	200	8	3902.96	4000.67	217.78	220.86	30693.53	31512.05	11.43	0.35
	300	8	7970.24	7985.14	932.51	968.17	120037.50	123283.20	22.36	0.74
	400	8	13055.96	13118.59	3212.46	2931.14	312943.46	303192.39	41.12	1.35
random 	50	16	106.84	119.77	24.24	24.24	1466.29	1466.62	0.12	0.00
	100	16	467.74	462.95	125.76	125.83	7809.09	7813.90	0.45	0.01
	150	16	1048.85	1026.20	342.63	342.75	22113.44	22106.02	1.37	0.05
	200	16	1959.77	1958.99	815.28	818.59	54776.02	55069.65	4.31	0.15
	250	16	3496.92	3515.45	1667.38	1669.86	116124.45	115630.32	17.71	0.61
warehouse 	150	16	1844.67	1853.64	115.86	116.32	8481.24	8370.05	3.32	0.10
	200	16	3131.47	3024.86	258.14	257.94	19925.99	19661.84	6.68	0.21
	250	16	4494.93	4534.01	464.60	462.98	38343.22	38725.28	12.20	0.43
	300	16	6228.19	6044.16	751.20	756.10	66391.85	66866.06	19.59	0.67
	350	16	7914.13	7696.34	1200.63	1203.22	110245.29	110509.65	30.02	1.07

- **AUC:** Area Under the Curve (AUC), defined as the integral of the SOD Graph, starting from the time the initial solution is selected until the specified time limit is reached.
- $t_{\text{init}}[s]$: Runtime required to generate initial solutions.

The results demonstrate that for the map `empty-8-8` with 16, 24, and 32 agents, ISS tends to favor initial solutions with lower initial SODs compared to LNS. The values for 16 and 24 agents are 10.560 vs. 10.722, respectively, and 34.280 vs. 35.057 for 32 agents. However, for 32 agents, the initial SOD for LNS is 82.563, while that of ISS is 93.760. With regard to the final SOD, both algorithms achieve identical final SODs for 16 agents. However, the final SOD for ISS is marginally superior for 24 agents and slightly better for 32 agents, with values of 25.133 compared to 25.185 for LNS. Regarding the AUC values, ISS outperforms LNS in two out of three cases. It shows superior performance for 32 agents and slightly better performance for 16 agents compared to LNS.

On the map `empty-32-32` with 300, 350, 400, and 450 agents, LNS yields lower initial SODs across all number of agents compared to ISS. However, ISS outperforms LNS slightly in terms of final SOD values for 300 and 350 agents. For instance, the final SOD for 300 agents is 440.09 for ISS compared to 442.37 for LNS. However, LNS achieves significantly superior final SOD values for 400 and 450 agents. With regard to the AUC values, ISS yields significantly lower values for 300 and 350 agents than LNS, whereas LNS achieves significantly lower values for 400 and 450 agents.

For the map `ost003d` with 100, 200, 300, and 400 agents, ISS favors lower initial SODs for 200, 300, and 400 agents compared to LNS. With regard to the final SODs, ISS achieved significantly lower values for 200, and 300 agents,

with particularly notable performance for 300 agents. The final SOD for 300 agents was 932.51 for ISS, in contrast to 968.17 for LNS. Moreover, ISS achieves significantly lower AUC values for 200 and 300 agents than LNS, whereas LNS yields significantly lower AUC values for 100 and 400 agents.

On the map `random` with 50, 100, 150, 200, and 250 agents, ISS achieves superior final SODs across all scenarios despite having higher initial SODs in four out of five cases. This indicates that the final solution quality is indeed dependent on the initial solution. For instance, the final SOD for 150 agents was 342.63 for ISS and 342.75 for LNS, but ISS has a significantly higher initial SOD of 1048.85 compared to 1026.20 for LNS. The AUC values for ISS are lower in three out of five cases, which highlights the benefit of initiating the optimization process from a promising initial solution.

For the map `warehouse` with 150, 200, 250, 300, and 350 agents, LNS achieves lower initial SODs in three out of five cases, while ISS outperforms LNS in three out of five cases in terms of final SOD. In particular, ISS exhibits significantly superior results for higher numbers of agents. For example, the final SOD for 350 agents is 1200.63 for ISS compared to 1203.22 for LNS. Regarding the AUC values, ISS achieves significantly superior solutions in three out of five cases lower than LNS, while LNS yields superior solutions for 150 and 200 agents.

With regard to $t_{\text{init}}[s]$, for all maps and number of agents, the initialization times are approximately 30 times higher for ISS compared to LNS. This reflects the additional effort required to generate multiple initial solutions.

Key Insights:

- **Initial Solution Quality:** In many cases, the initial solutions generated by ISS exhibit lower quality. However, the final solutions achieved by ISS are of a higher quality, particularly in constrained environments such as on maps `warehouse` and `random`.
- **Final Solution Quality and AUC:** In 13 out of 21 cases, the final SODs found by ISS were lower than those of LNS. Moreover, the AUC values obtained by ISS were in 12 out of 21 cases lower than those of LNS. This demonstrates the effectiveness of using LambdaMART to select promising initial solutions.
- **Computational Overhead:** The primary trade-off for the superior performance of ISS is the increased time required to generate initial solutions. This overhead is particularly evident in larger and more complex maps.
- **Seen maps:** (1) `random`: ISS performs particularly well, achieving lower final SODs in four out of five scenarios compared to LNS. This indicates the efficacy of the learned model in handling familiar environments. (2) `warehouse`: ISS shows robust performance, with lower final SODs in three out of five groups compared to LNS. Despite the longer initial solution generation time, ISS effectively optimized the solutions.
- **Unseen Maps:** ISS remains competitive with LNS on unseen maps like `empty-32-32` and `ost003d`, which highlights the generalization capability of the trained models.

The results indicate that the quality of the final solution in LNS is strongly correlated with its initial solution. ISS leverages its ability to select a promising solution from a diverse pool of generated initial solutions, which, although time-consuming, results in better overall optimization. The extended initial solution time for ISS is a trade-off that pays off in terms of final solution quality, particularly in complex and large maps like `warehouse`.

The strength of ISS lies in its ability to learn from training data and apply this learning to both seen and unseen scenarios, providing a robust solution even when starting late in the LNS process. This demonstrates the potential of integrating ML techniques with traditional optimization methods in MAPF problems.

6 Conclusion and Future Work

In this work, we addressed the significant dependency of the final solution quality of MAPF-LNS on its initial solution. Despite the fact that the initial solution quality is not strongly correlated with the final solution quality, the choice of an initial solution has a substantial impact on the performance of MAPF-LNS. We proposed ISS-MAPF-LNS, a novel extension of MAPF-LNS. This approach involves executing PP with random priorities multiple times to generate a large pool of potential initial solutions. From this pool, the most promising initial solution is selected using LambdaMART, a state-of-the-art learning-to-rank algorithm, to run the LNS on.

LambdaMART was trained offline on two well-known maps, and the experimental results demonstrate that the trained model generalizes well to different numbers of agents on both seen and unseen grid maps. This indicates that our approach can effectively leverage ML to enhance the performance of MAPF-LNS.

In order to improve the robustness and generalization capability of ISS-MAPF-LNS, future research should focus on creating a more diverse set of training data. By including a wider variety of maps and scenarios during the training phase, we may reduce the model’s dependence on specific map features and enhance its ability to generalize to different environments. This broader training dataset may help the model to better understand the diverse characteristics of various maps, leading to more reliable performance in new scenarios.

One of the primary challenges identified is the computational overhead associated with generating multiple initial solutions. An alternative or additional ML-based approach is to employ machine learning techniques to directly predict promising destroy sets, which are crucial components of the LNS optimization process.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proc. of the 25th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (2019)

2. Burges, C.J.C.: From RankNet to LambdaRank to LambdaMART: An overview. Tech. rep., Microsoft Research (2010)
3. Cohen, L., Greco, M., Ma, H., Hernandez, C., Felner, A., Kumar, T.K.S., Koenig, S.: Anytime focal search with applications. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI). pp. 1434–1441 (2018)
4. Huang, T., Li, J., Koenig, S., Dilkina, B.: Anytime multi-agent path finding via machine learning-guided large neighborhood search. In: Proc. of the 36th AAAI Conf. on Artificial Intelligence (AAAI). pp. 9368–9376 (2024)
5. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* **30**, 3146–3154 (2017)
6. Lam, E., Harabor, D.D., Stuckey, P.J., Li, J.: Exact anytime multi-agent path finding using branch-and-cut-and-price and large neighborhood search. In: Proc. of the 33th Int. Conf. on Autom. Planning and Sched. (ICAPS). pp. 254–258 (2023)
7. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: Anytime Multi-Agent Path Finding via Large Neighborhood Search. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI). pp. 4127–4135 (2021)
8. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In: Proc. of the 36th AAAI Conf. on Artificial Intelligence (AAAI). pp. 10256–10265 (2022)
9. Pearl, J., Kim, J.H.: Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-4**, 392–399 (1982)
10. Phan, T., Huang, T., Dilkina, B., Koenig, S.: Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. In: Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI). pp. 17514–17522 (2024)
11. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.F. (eds.) *Principles and Practice of Constraint Programming — CP98*. LNCS, vol. 1520, pp. 417–431. Springer (1998)
12. Silver, D.: Cooperative Pathfinding. In: Proc. of the First AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment (AAAI). pp. 117–122 (2005)
13. Standley, T., Korf, R.: Complete algorithms for cooperative pathfinding problems. In: Proc. of the 22th Int. Joint Conf. on AI (IJCAI). pp. 668–673 (2011)
14. Stern, R., Sturtevant, N.R., Felner, A., Koenig, S., Ma, H., Walker, T.T., Li, J., Atzmon, D., Cohen, L., Kumar, T.K.S., Boyarski, E., Barták, R.: Multi-agent pathfinding: Definitions, variants, and benchmarks. In: Proceedings of the 12th International Symposium on Combinatorial Search (SoCS). pp. 151–159 (2019)
15. Vedder, K., Biswas, J.: X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs. *Artificial Intelligence* **291**, 103417 (2021)
16. Veloso, M., Biswas, J., Coltin, B., Rosenthal, S.: CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI). pp. 4423–4429 (2015)
17. Wurman, P.R., D’Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine* **29**(1), 9 (2008)
18. Yu, J., LaValle, S.M.: Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI). pp. 1443–1449 (2013)