Technical Report

Benchmarking Large Neighborhood Search for Multi-Agent Path Finding

1 Experiment Details

1.1 Environment Setup

Training. We train Neural on a cluster equipped with A40 GPUs, with a learning rate of 0.00001. and a batch size of 16 for $10,000$ to $100,000$ steps until the loss and validation score no longer improve for another $1,000$ steps. We use a smaller learning rate than that used in the original paper because we start from a different initial solution, and we find that a lower learning rate results in a more stable reduction in loss in our unified setting. The entire training process is completed in less than 24 hours. For the SVM, we train the model on Intel Xeon Gold 5218 CPUs for 100 iterations.

Inference. For all experiments, we use an Intel E5-2683 CPU with a memory limit of 2G. We use an NVIDIA P100 GPU for Our-Neural and Orig-Neural inference. The average GPU inference overhead of Neural is summarized in Table 4.

Initial Solutions, Replan Solvers, Neighborhood Size, Number of Agents The initial solutions, replan solvers, and neighborhood sizes used for different methods are summarized in Table 1. The number of agents evaluated in different maps are summarized in Table 2.

Table 1: MAPF Benchmark Evaluation Settings. 'La' and 'LN' are short for LaCAM2 (Okumura, 2023) and LNS2 (Li et al., 2022). PP is Prioritized Planning (Erdmann & Lozano-Perez, 1987). PBS is Parallel Push and Swap (Sajid et al., 2012). EECBS is Explicit Estimation CBS (EECBS) (Li et al., 2021). Prefix "Our-" refers methods in our setting. Prefix "Orig-" refers methods in their original settings.

Method	Initial Solution	Replan Solver	Neighborhood size
RandomWalk	La, LN	PP	$\{4, 8, 16, 32\}$
Intersection	La, LN	PP	$\{4, 8, 16, 32\}$
Random	La, LN	PP	$\{4, 8, 16, 32\}$
Adaptive	La, LN	PP	$\{4, 8, 16, 32\}$
name	La, LN	PP	$\{4, 8, 16, 32\}$
Our-SVM-LNS	La, LN	PP	$\{4, 8, 16, 32\}$
Our-Neural-LNS	La, LN	PP	$\{4, 8, 16, 32\}$
Our-Bandit-LNS	La, LN	PP	$\{2, 4, 8, 16, 32\}$ by second arm
Orig-SVM-LNS	PP, PPS, EECBS	PP	Uniformly select from 5 to 16
Orig-Neural-LNS	PP, PPS	PBS	$\{10, 25, 50\}$ for different maps
Orig-Bandit-LNS	PP, PPS, EECBS	PP	$\{2, 4, 8, 16, 32\}$ by second arm

Table 2: Number of agents evaluated in different maps

Map	Agent number
empty-32-32 (empty)	300, 350, 400, 450, 500
random-32-32-20 (random)	150, 200, 250, 300, 350
warehouse-10-20-10-2-1 (warehouse)	150, 200, 250, 300, 350
ost003d	200, 300, 400, 500, 600
den520d	500, 600, 700, 800, 900
Paris_1_256 (Paris)	350, 450, 550, 650, 750

1.2 Training of SVM-LNS and Neural-LNS under Original Settings

We summarize the training settings of original SVM-LNS and Neural-LNS, and compare the performance we get with the reported performance to validate our implementation.

1.2.1 Original SVM-LNS (Orig-SVM-LNS)

Training data: We use the suggested agent numbers by Huang et al. (2022) for training if the map exists in the original paper (shown in the second column of Table **??**). For maps not evaluated in Huang et al. (2022), we use 300 agents for empty-32-32 and 150 agents for random-32-32-20. As suggested, 16 scenes of each map are used for training.

Training procedure: In each iteration, 20 neighborhood candidates are proposed, with each one by RandomWalk or Intersection with equal probability and with $|\tilde{A}|$ randomly sampled from 5 to 16. The ground truth ranking information for these 20 candidates is determined by the delay improvement if each neighborhood is removed and replaned. Each neighborhood is described by a vector of handcraft features with 128 dimensions (see Table 1 of Huang et al. (2022)). Linear SMV^{rank} (Joachims, 2002) models are then trained based on handcraft features and ranking labels for 100 iterations.

Choosing best model: Validation data are collected by solving 4 scenes of a map for 100 iterations using rule-based strategies as in training. In each iteration, the best neighborhood is selected. We calculate the average rank on the validation set to select the best model checkpoint. Here, 'average ranking' means the mean ranking of the best neighborhood predicted by the model appearing in the ground truth ranking over the validation dataset.

1.2.2 Original Neural-LNS (Orig-Neural-LNS)

Table 3: Training Data Collection Settings for Orig-Neural-LNS

Orig-Neural-LNS				
Map	Strategy	NB	Iteration	Scenes
empty	Random	50	50	5000
random	RandomWalk	25	50	5000
warehouse	RandomWalk	25	25	5000
ost003d	RandomWalk	10	25	1000
den520d	RandomWalk	25	50	5000
Paris	RandomWalk	25	50	4000

Table 4: **Average GPU Overhead for Neural**. The table displays the average GPU overhead for various numbers of agents and map settings across all network forward rounds.

Map	n	Overhead	Map	n	Overhead	Map	n	Overhead
empty	300	0.016	random	150	0.014	warehouse	150	0.042
	350	0.017		200	0.015		200	0.043
	400	0.019		250	0.021		250	0.043
	450	0.022		300	0.024		300	0.044
	500	0.028		350	0.026		350	0.044
ost003d	200	0.020	den520d	500	0.033	Paris	350	0.038
	300	0.020		600	0.041		450	0.038
	400	0.024		700	0.041		550	0.038
	500	0.024		800	0.055		650	0.040
	600	0.027		900	0.045		750	0.042

Training data: As suggested by the author, we use a medium number of agents to collect the training set. We run 25 to 50 iterations for each map, continuing until there is no further decrease in delay for an additional 5 iterations. In each iteration, 100 neighborhood candidates are proposed using the suggested rule-based strategy and neighborhood size, and the best neighborhood is selected. We attempt to train the model with varying amounts of data, ranging from $1,000$ to $9,000$ in increments of $1,000$, and then select the model that achieves the best performance with the least training data.

The exact number of iterations, scenes, neighborhood sizes, and removal strategies used for each map are summarized in Table 3.

Training procedure: For each map, the model is trained on the corresponding training set. We stop the training when the loss converges and the average ranking on the validation set no longer improves for another $1,000$ steps.

Choosing best model: We run an additional 25 scenes for the same number of iterations as used in training data collection for each map to collect validation data. In each iteration, the best neighborhood is selected. We calculate the average rank on the validation set to select the best model checkpoint.

1.3 Training of SVM-LNS and Neural-LNS under Unified Setting

1.3.1 Our-SVM-LNS

The basic training and validation for Our-SVM-LNS are the same as Orig-SVM-LNS. The differences are: 1) Our-SVM-LNS are trained on maps with a medium number of agents. 2) The rule-based strategy and corresponding neighborhood size are chosen as the best combination in that map.

1.3.2 Our-Neural-LNS

Training data: We use a medium number of agents for each map to collect training data. The replan solver is PP. Using PP as the replan solver generally takes much more iterations than using PBS to converge. Thus, to collect relatively the same amount of data samples, we use fewer scenes for each map. The exact number of iterations, scenes, neighborhood sizes, and removal strategies used for each map are summarized in Table 5

Table 5: Training Data Collection Settings for Ours-Neural-LNS

Ours-Neural-LNS				
Map	Strategy	NB	Iteration	Scenes
empty	Adaptive	8	1400	300
random	RandomWalk	8	1000	100
warehouse	Adaptive	32	200	200
ost003d	RandomWalkProb	16	400	250
den520d	RandomWalkProb	16	500	200
Paris	RandomWalkProb	32	200	350

Training procedure: The procedure for training is the same as Orig-Neural-LNS.

Choosing best model: We use an additional 4 scenes and run for the same number of iterations as the training for each map to collect validation data. We calculate the average rank on the validation set to select the best model checkpoint.

1.4 Additional Result on PP vs. PBS

We report the total iterations, final delays, and area under the curve (AUC) of delay versus time within the time limit 60s and 300s in Table 6. The results where PBS is better than PP is highlighted in red. For final delays, PP is better than PBS in 72.5% (87/120) cases. For AUC, PP is better than PBS in 81.7% (98 / 120) cases. Even though PBS is better than PP in "random" map, the final delays and AUC are relatively close. In general, PP runs significantly faster than PBS and thus can explore a substantially larger amount of neighborhoods within the time limit.

Remark. Our results conflict with the claim in Yan & Wu (2024) that "PP typically has worse solution quality to time tradeoff than PBS". However, the comparison results between PP and PBS are not given for all maps in Yan & Wu (2024), and the time counting details are not described.

2 Complete Experiment Result

2.1 Delay and AUC with Different Initial Solver

Table 7 and Table 8 shows the results with a medium and largest number of agents using LaCAM2 as the initial solver. Table 9 presents the results with a medium number of agents using EECBS as the initial solver, where only two maps are fully solvable by EECBS within 10 seconds. Tables 10, 11, 12, 13 show the complete results with runtime limit 300 seconds using LNS2 and LaCAM2 as the initial solvers under all agent number settings.

3 Additional Experiment Result

3.1 Evaluation on the Effectiveness of Bandit

Bandit is the first work to attempt choosing the best neighborhood size. We test the effectiveness of its neighborhood selection arm by replacing it with random uniform neighborhood size selection; specifically, we draw a neighborhood size from $2, 4, 8, 16, 32$ at each iteration. Table 14 shows the comparison of Delay and AUC between Bandit and our uniform sample Bandit (Uni-Bandit). The results indicate that the neighborhood selection arm is not effective and sometimes even worse than random uniform neighborhood size selection, highlighting the need for future improvements.

Table 6: Total iterations, final delays, and AUC in different evaluation cases within a time limit of **60s** and **300s**, using PP and PBS as replan solvers. The neighborhood selection strategy is RandomWalk with a neighborhood size of 25. 'In' refers to the algorithm used for finding initial solutions. For both final delays and AUC, lower values are better. The settings where PBS performs better are highlighted in red, for all other settings, PP is superior.

Run Time Limit: 60s

Map	In	n	Iter (x1k) PP	Iter (x1k) PBS	Final delays PP	Final delays PBS	AUC (x10k) PP	AUC (x10k) PBS
empty	LaCAM2	300	8.21	0.48	439.6	424.5	3.89	6.07
		350	4.13	0.22	1,127.6	1,286.8	9.4	14.2
		400	2.24	0.12	2,663.1	2,982.4	20.2	27.1
		450	2.15	0.11	5,110.7	5,211.0	36.9	41.4
		500	2.40	0.08	8,400.6	8,815.3	59.2	64.5
	LNS2	300	8.98	0.44	431.7	436.9	3.5	4.8
		350	4.22	0.25	1,109.8	1,081.8	8.7	9.8
		400	2.28	0.15	2,570.1	2,238.2	18.7	17.5
		450	1.83	0.09	4,873.5	4,293.6	32.7	29.9
		500	1.51	0.05	7,817.6	6,874.2	49.3	45.2
warehouse	LaCAM2	150	6.41	0.60	116.8	133.1	1.4	6.6
		200	2.94	0.25	259.8	319.3	7.3	7.6
		250	1.74	0.13	486.7	941.4	18.6	21.7
		300	1.06	0.07	845.2	2,987.7	39.6	44.9
		350	0.65	0.03	1,625.8	7,963.7	82.4	83.7
	LNS2	150	6.59	0.57	122.1	128.3	2.3	2.8
		200	2.65	0.27	266.8	310.2	6.8	6.5
		250	1.83	0.15	477.6	760.3	16.7	16.2
		300	1.15	0.09	832.7	1,740.3	33.7	30.4
		350	0.73	0.06	1,495.0	3,237.5	54.7	51.2
den520d	LaCAM2	500	1.44	0.04	871.6	8,082.4	18.5	77.1
		600	1.07	0.02	2,266.5	17,753.3	35.3	129.3
		700	0.86	0.02	4,396.1	24,979.7	57.9	175.6
		800	0.53	0.01	9,205.8	35,921.6	96.9	234.4
		900	0.49	0.01	12,900.4	45,686.7	124.4	291.1
	LNS2	500	1.28	0.05	899.6	6,195.8	15.9	52.9
		600	1.72	0.06	1,321.3	8,485.5	29.8	79.2
		700	0.78	0.02	4,436.5	16,642.9	49.1	111.9
		800	0.61	0.02	7,342.8	21,909.0	73.2	142.2
		900	0.44	0.01	13,032.0	29,352.2	105.6	181.4

Map	In	n	Iter (x1k) PP	Iter (x1k) PBS	Final delays PP	Final delays PBS	AUC (x10k) PP	AUC (x10k) PBS
random	LaCAM2	150	6.21	0.58	352.1	343.1	2.5	3.6
		200	3.01	0.20	952.5	874.9	7.3	7.6
		250	2.22	0.07	2,449.5	2,443.6	18.6	21.7
		300	2.03	0.04	5,318.2	5,693.2	39.6	44.9
		350	2.61	0.00	14,729.1	14,630.2	82.4	83.7
	LNS2	150	7.39	0.63	350.1	346.9	2.3	2.8
		200	2.88	0.19	959.6	875.5	6.8	6.5
		250	1.99	0.05	2,423.8	2,301.4	16.7	16.2
		300	1.47	0.03	5,309.6	4,533.1	33.7	30.4
		350	1.57	0.02	8,966.9	8,076.5	54.7	51.2
ost003d	LaCAM2	200	1.73	0.09	198.5	1,074.5	3.7	12.4
		300	0.93	0.07	988.8	3,117.1	13.9	35.6
		400	0.57	0.03	3,285.9	9,320.1	37.3	81.6
		500	0.19	0.01	12,164.9	21,539.9	98.6	145.5
		600	0.13	0.01	27,290.3	35,498.7	188.7	221.2
	LNS2	200	1.75	0.06	183.9	897.6	2.8	9.5
		300	0.92	0.02	915.5	4,630.9	12.3	35.2
		400	0.52	0.03	3,230.7	8,032.7	32.4	62.3
		500	0.21	0.01	9,335.3	16,709.3	72.3	107.8
		600	0.15	0.01	17,998.3	24,525.7	125.2	152.2
Paris	LaCAM2	350	7.48	0.16	99.8	817.8	1.8	18.5
		450	6.69	0.10	134.3	3,032.1	6.2	79.9
		550	5.37	0.05	213.7	8,664.9	10.6	119.2
		650	4.58	0.03	298.0	15,771.8	18.5	165.3
		750	3.57	0.02	483.6	24,171.6	18.5	165.3
	LNS2	350	5.98	0.17	82.2	383.7	1.0	8.4
		450	6.44	0.11	138.7	2,274.2	1.8	22.8
		550	4.72	0.06	219.3	4,878.6	4.1	46.0
		650	4.49	0.04	317.1	9,304.6	4.6	14.7
		750	3.07	0.03	614.9	14,707.1	14.8	104.5

Run Time Limit: 300s

Map	In	n	Iter (x10k) PP	Iter (x10k) PBS	Final delays PP	Final delays PBS	AUC (x10k) PP	AUC (x10k) PBS
empty	LaCAM2	300	4.29	0.21	367.93	332.33	13.31	14.58
		350	2.07	0.12	840.33	692.63	31.70	33.88
		400	1.09	0.06	1971.57	1544.34	73.21	75.04
		450	1.07	0.04	3899.78	3258.99	140.35	135.04
		500	1.28	0.03	6615.22	6092.54	232.28	231.80
	LNS2	300	4.64	0.19	363.99	338.95	12.64	13.45
		350	2.09	0.12	853.13	695.69	31.16	28.84
		400	1.10	0.07	1920.56	1400.21	69.98	57.49
		450	1.00	0.04	3749.88	3010.31	128.33	112.73
		500	0.98	0.02	6447.52	5352.49	216.74	186.70
warehouse	LaCAM2	150	3.01	0.19	113.44	126.99	4.14	9.70
		200	1.48	0.12	247.88	262.47	9.40	16.07
		250	0.93	0.08	435.51	445.94	18.17	38.03
		300	0.55	0.05	678.67	741.87	31.86	74.78
		350	0.35	0.03	1082.86	1471.08	61.14	154.76
	LNS2	150	2.11	0.19	117.88	123.87	3.74	6.44
		200	0.98	0.11	247.48	261.31	8.25	11.05
		250	0.70	0.07	431.72	462.32	15.25	21.93
		300	0.55	0.05	694.66	757.45	26.39	41.51
		350	0.36	0.03	1042.26	1365.12	43.40	75.23
den520d	LaCAM2	500	7.41	0.24	313.61	2545.75	28.71	174.77
		600	5.34	0.14	612.69	5572.76	59.27	355.08
		700	4.52	0.12	1110.30	8460.08	104.28	525.67
		800	2.79	0.08	2410.77	16784.33	198.78	832.76
		900	2.75	0.06	3648.21	27560.22	274.32	1148.42
	LNS2	500	6.64	0.24	296.55	2551.81	26.00	146.33
		600	5.23	0.16	583.83	5301.91	52.54	241.87
		700	4.07	0.12	1196.94	8145.64	97.44	384.84
		800	3.27	0.11	2076.04	12242.85	159.80	535.81
		900	2.33	0.03	3979.49	23163.43	262.41	807.24

Map	In	n	Iter (x10k) PP	Iter (x10k) PBS	Final delays PP	Final delays PBS	AUC (x10k) PP	AUC (x10k) PBS
random	LaCAM2	150	3.11	0.24	329.46	321.35	10.62	11.40
		200	1.58	0.10	789.77	723.64	27.32	25.79
		250	1.24	0.03	1857.06	1618.02	67.80	66.54
		300	1.21	0.02	4102.65	3602.96	147.89	145.24
		350	1.06	0.01	8553.05	7610.30	315.27	310.00
	LNS2	150	3.78	0.23	333.89	328.94	10.48	10.81
		200	1.53	0.10	812.82	741.66	27.44	25.24
		250	1.21	0.03	1841.19	1651.25	65.43	61.16
		300	0.95	0.01	4253.84	3527.62	146.07	123.06
		350	0.87	0.01	8225.15	6477.20	260.50	221.69
ost003d	LaCAM2	200	0.87	0.04	154.04	278.75	7.60	24.60
		300	0.50	0.03	381.95	1160.64	26.76	82.84
		400	0.37	0.02	1009.37	4071.87	76.53	207.71
		500	0.13	0.01	4618.86	9758.70	275.91	483.51
		600	0.07	0.00	15629.46	26768.67	691.40	960.09
	LNS2	200	0.50	0.02	152.40	273.43	6.69	19.75
		300	0.40	0.01	384.24	1976.41	24.38	106.45
		400	0.31	0.01	1047.86	4422.10	72.15	192.26
		500	0.15	0.01	3671.84	10219.26	204.61	417.48
		600	0.08	0.00	11333.95	19107.23	465.10	668.65
Paris	LaCAM2	350	35.13	0.70	99.32	98.29	4.23	22.87
		450	31.77	0.48	130.44	273.60	6.32	65.28
		550	25.61	0.33	203.97	1520.53	11.17	168.08
		650	22.87	0.21	267.48	4650.54	17.19	302.78
		750	16.96	0.15	366.11	8294.67	27.73	506.11
	LNS2	350	20.20	0.71	80.40	100.92	2.96	11.59
		450	24.70	0.50	136.48	216.70	5.15	38.58
		550	12.81	0.33	205.32	1159.81	9.14	107.83
		650	20.25	0.23	280.04	3657.95	14.13	203.35
		750	9.60	0.17	414.18	6561.66	25.51	323.93

Table 7: Final delays and AUC (divided by 10k) of different methods with best neighborhood size, evaluated on maps with medium number of agents within 300s and 60s. The agent numbers are shown after the name of a map. Initial solutions are provided by LaCAM2. Highlighted are the results ranked first, and second.

Methods	**empty**+400		**random**+250		**warehouse**+250		**ost003d**+400		**den520d**+700		**Paris**+550	
	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC
	Init solution: LaCAM2, Time: 300s											
RandomWalk	1447.93	52.87	1472.74	50.15	430.02	18.10	430.02	18.10	933.10	86.97	198.04	10.72
Intersection	1593.40	58.25	1537.20	53.64	698.50	47.96	1849.61	97.46	3163.02	169.04	776.46	58.17
Random	1756.18	61.44	1613.49	54.29	562.40	29.80	1488.17	91.63	2782.67	176.17	466.87	48.55
Adaptive	1476.49	53.10	1506.69	50.95	424.93	19.31	805.22	55.20	1075.19	91.26	194.79	15.25
RandomWalkProb	1484.35	53.88	1494.11	50.56	446.69	20.09	622.85	38.92	622.43	55.41	183.27	9.73
Our-SVM	1692.37	83.42	1521.21	60.13	444.61	41.10	762.46	96.38	660.71	138.12	184.72	33.00
Our-Neural	2094.76	81.62	1747.14	63.64	782.08	38.25	675.32	46.35	816.58	85.94	228.87	17.10
Bandit	1579.02	57.57	1456.33	48.61	426.15	16.62	1253.27	78.42	2439.18	165.78	210.23	28.72
Orig-SVM	1857.49	97.55	1604.75	67.43	445.08	36.29	1083.73	123.70	2258.95	327.11	200.56	105.33
Orig-Neural	1614.31	74.40	1698.53	70.88	961.45	127.21	5277.44	310.16	15529.00	726.01	2643.73	261.44
	Init solution: LaCAM2, Time: 60s											
RandomWalk	1845.71	15.22	1698.27	13.15	470.07	7.20	1717.20	27.23	2641.97	46.77	217.23	5.90
Intersection	2030.12	16.65	1830.54	14.51	1731.44	24.94	3618.77	41.14	6254.74	71.26	2269.39	27.09
Random	2130.49	16.44	1832.18	14.00	813.23	14.31	3750.10	41.21	7336.47	76.57	2110.06	27.09
Adaptive	1861.78	14.81	1733.45	13.13	485.38	8.66	2023.81	29.13	3418.03	51.79	236.39	7.43
RandomWalkProb	1877.6	15.07	1711.63	13.14	537.08	8.84	1143.82	21.44	1494.63	36.19	202.53	5.20
Our-SVM	3679.88	30.54	2274.59	19.86	1144.09	28.78	5073.09	52.66	6070.30	97.46	545.62	27.87
Our-Neural	2981.60	24.63	2246.29	18.19	1201.94	16.08	1541.18	25.71	2933.96	56.16	350.31	10.91
Bandit	2027.85	16.12	1652.80	12.23	466.79	6.09	3118.42	35.25	7198.20	75.06	960.80	20.78
Orig-SVM	4503.57	34.83	2587.10	23.63	872.97	24.43	6332.78	69.58	19781.82	158.77	7313.95	78.02
Orig-Neural	2942.81	27.94	2674.51	23.30	7424.49	60.45	14720.52	102.22	30196.75	191.74	13938.81	99.11

Table 8: Final delays and AUC (divided by 10k) of different methods with best neighborhood size, evaluated on maps with largest number of agents within 300s and 60s. The agent numbers are shown after the name of a map. Initial solutions are provided by LaCAM2. Highlighted are the results ranked first, and second.

Methods	**empty**+500		**random**+350		**warehouse**+350		**ost003d**+600		**den520d**+900		**Paris**+750	
	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC
	Init solution: LaCAM2, Time: 300s											
RandomWalk	4237.71	155.66	4341.57	169.08	1043.00	56.83	7513.99	450.57	2486.09	201.86	373.15	27.46
Intersection	4448.52	162.94	4622.23	184.56	2010.98	158.73	10394.78	529.43	6425.98	351.11	1821.18	123.22
Random	4828.23	171.83	4683.93	186.63	1509.02	92.99	12466.14	613.46	6923.52	396.91	1362.59	115.02
Adaptive	4374.48	156.47	4406.49	163.77	1054.99	63.12	8216.92	469.61	2819.86	225.56	385.84	39.40
RandomWalkProb	4335.71	159.82	4367.54	166.60	1130.83	67.02	5874.98	383.28	1381.97	151.59	366.01	23.87
Our-SVM	5194.85	223.50	5162.68	241.24	1042.40	100.43	13066.61	618.02	2058.55	335.36	369.93	108.04
Our-Neural	5808.02	221.09	5886.46	241.76	1907.01	101.40	8469.09	465.00	2067.14	217.58	498.84	47.80
Bandit	4772.62	173.84	4598.53	169.33	1067.58	49.67	7244.61	400.06	5595.39	346.67	627.28	87.12
Orig-SVM	5793.57	243.96	5983.60	262.86	1097.62	112.19	10153.89	625.01	15199.04	723.88	616.29	261.45
Orig-Neural	6085.27	242.91	7438.82	312.23	5255.12	361.57	25804.48	962.95	31693.98	1246.97	13831.51	656.43
	Init solution: LaCAM2, Time: 60s											
RandomWalk	5605.03	43.83	5535.21	46.40	1421.02	30.36	20549.41	162.98	8274.09	108.85	1845.70	15.20
Intersection	5895.29	45.63	6027.22	50.65	7091.60	72.70	21652.95	171.51	13090.97	138.60	2030.10	16.60
Random	6075.96	43.22	5764.85	46.66	2764.39	49.76	22508.10	173.33	16636.17	154.71	2130.40	16.40
Adaptive	5631.04	41.84	5544.01	45.50	1577.07	33.52	20281.28	160.24	9786.31	114.84	1861.70	14.80
RandomWalkProb	5790.5	44.6	5463.1	45.6	1514.3	34.2	17560.6	153.3	4788.8	86.8	462.8	13.5
Our-SVM	8973.59	63.69	10463.71	75.41	4450.79	63.28	26121.51	188.01	18143.86	185.35	5088.31	85.13
Our-Neural	8475.42	60.29	9710.57	71.69	3366.54	44.95	20302.89	162.10	9199.73	126.87	1080.62	32.78
Bandit	6294.53	48.15	5943.98	49.94	1334.33	22.40	16993.81	146.55	14398.93	144.19	3742.99	52.95
Orig-SVM	9866.16	67.50	10815.58	76.92	5331.10	71.10	29866.42	204.27	30208.88	191.02	18276.64	146.27
Orig-Neural	9953.82	68.31	12978.23	83.86	17479.58	119.62	36312.50	224.55	47864.21	297.39	27245.04	175.07

Table 9: Final delays and AUC (divided by 10k) of different methods with best neighborhood size, evaluated on maps with medium number of agents within 300s and 60s. The agent numbers are shown after the name of a map. Initial solutions are provided by EECBS. Highlighted are the results ranked first, and second.

Methods	**empty**+400		**random**+250		**warehouse**+250		**ost003d**+400		**den520d**+700		**Paris**+550	
	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC	Delay	AUC
Init solution: EECBS, Time: 300s												
RandomWalk	1116.1	38.47	-	-	402.03	12.88	-	-	-	-	-	-
Intersection	1228.2	41.12	-	-	470.4	15.60	-	-	-	-	-	-
Random	1311.3	43.47	-	-	440.4	14.07	-	-	-	-	-	-
Adaptive	1170.1	39.03	-	-	408.8	13.14	-	-	-	-	-	-
RandomWalkProb	1202.1	39.86	-	-	430.5	14.16	-	-	-	-	-	-
Our-SVM	1252.44	47.82	-	-	408.12	13.24	-	-	-	-	-	-
Our-Neural	1506.54	51.73	-	-	524.0	17.30	-	-	-	-	-	-
Bandit	1250.0	42.32	-	-	404.76	12.69	-	-	-	-	-	-
Orig-SVM	1313.6	51.84	-	-	417.2	13.69	-	-	-	-	-	-
Orig-Neural	1409.7	53.51	-	-	497.1	18.30	-	-	-	-	-	-
Init solution: LaCAM2, Time: 60s												
RandomWalk	1313.84	9.42	-	-	436.93	2.94	-	-	-	-	-	-
Intersection	1412.44	10.13	-	-	549.7	3.72	-	-	-	-	-	-
Random	1438.77	9.86	-	-	474.47	3.25	-	-	-	-	-	-
Adaptive	1342.73	9.32	-	-	446.62	3.07	-	-	-	-	-	-
RandomWalkProb	1369.38	9.72	-	-	488.39	3.32	-	-	-	-	-	-
Our-SVM	1826.71	14.57	-	-	450.96	3.13	-	-	-	-	-	-
Our-Neural	1823.47	13.10	-	-	599.86	4.07	-	-	-	-	-	-
Bandit	1473.95	10.60	-	-	425.42	2.81	-	-	-	-	-	-
Orig-SVM	1968.86	15.70	-	-	455.75	3.48	-	-	-	-	-	-
Orig-Neural	1946.74	15.82	-	-	694.49	4.74	-	-	-	-	-	-

Table 10: Final delays of different methods with best neighborhood size, evaluated on maps with differing numbers of agents within 300 seconds. Initial solutions are provided by LNS2. Highlighted are the results ranked first, and second.

map	n	RNWLK	INTC	RAND	ADP	RNWLKPB	Our-S	Our-N	Bandit	Orig-S	Orig-N
empty	300	358.0	406.4	435.1	369.0	369.9	381.5	724.6	386.3	395.6	325.9
	350	750.9	814.6	922.4	770.1	769.0	800.7	1209.9	811.5	807.7	734.3
	400	1397.2	1513.7	1703.2	1418.6	1431.1	1588.3	1928.5	1537.2	1640.4	1464.8
	450	2551.0	2695.0	2908.1	2577.6	2585.3	2766.0	3136.3	2753.7	2936.5	2876.6
	500	4050.5	4205.4	4438.9	4093.8	4051.3	5053.2	4803.2	4318.5	4776.3	5305.6
random	150	332.6	352.6	357.2	330.1	337.4	331.4	360.5	330.1	338.8	323.3
	200	762.0	811.2	831.1	771.3	784.4	786.5	900.2	779.1	790.8	728.5
	250	1504.5	1582.6	1606.3	1527.5	1534.4	1713.9	1683.7	1507.3	1662.8	1528.0
	300	2687.7	2839.7	2885.9	2737.2	2777.2	2885.9	3045.2	2746.0	3005.0	3207.8
	350	4439.2	4609.3	4635.2	4432.8	4408.2	4800.2	5466.7	4564.1	5105.2	6004.1
warehouse	150	117.2	123.5	122.6	109.0	113.0	114.1	244.7	107.9	112.7	164.7
	200	244.2	317.7	286.6	242.8	252.1	245.3	469.6	239.4	252.6	269.1
	250	433.1	695.2	537.7	435.6	443.8	439.6	749.0	414.2	441.0	417.4
	300	663.9	1161.5	950.3	683.7	729.3	681.3	1501.2	669.5	703.2	664.8
	350	1041.8	1949.2	1515.9	1073.0	1134.9	1107.4	1871.1	1047.6	1100.9	1094.5
ost003d	200	150.8	290.7	194.0	149.2	147.6	148.6	165.0	158.2	155.0	245.4
	300	327.3	929.5	632.0	337.6	298.0	315.4	338.4	532.8	338.2	582.6
	400	761.8	2048.3	1668.8	746.1	652.3	850.3	679.5	1276.3	1086.3	1483.2
	500	2051.4	4460.2	3611.6	2094.8	1515.2	2928.2	2012.2	3059.8	3092.7	4673.8
	600	6069.5	8824.9	8424.3	6325.9	4731.3	10104.8	5501.2	6093.6	8604.1	11426.4
den520d	500	293.5	1414.3	906.1	306.2	248.5	252.5	313.5	607.8	394.3	732.3
	600	536.9	2293.2	1656.1	583.1	396.7	389.0	496.1	1247.0	972.6	1786.5
	700	934.1	3415.0	2907.4	1049.8	620.2	665.2	814.0	2297.3	1880.1	4228.7
	800	1476.1	4535.4	4639.0	1685.4	883.2	1003.4	1102.0	3330.2	3876.9	7766.6
	900	2290.1	6776.0	6447.9	2611.1	1387.2	1816.6	1821.4	5343.4	6161.2	13367.1
Paris	350	80.0	121.5	83.8	75.5	76.4	71.9	78.6	1066.5	81.7	193.5
	450	127.4	273.6	172.7	124.9	118.8	120.7	142.6	737.1	123.2	130.8
	550	203.6	540.8	345.4	192.2	183.8	184.7	205.3	721.5	196.5	721.5
	650	278.1	963.6	650.2	270.2	257.1	262.0	313.1	1022.0	286.8	307.5
	750	404.7	1680.8	1211.7	396.9	375.6	388.7	468.2	576.9	483.2	2409.4

Table 11: AUC (divided by 10k) of different methods with best neighborhood size, evaluated on maps with differing numbers of agents within 300 seconds. Initial solutions are provided by LNS2. Highlighted are the results ranked first, and second.

map	n	RNWLK	INTC	RAND	ADP	RNWLKPB	Our-S	Our-N	Bandit	Orig-S	Orig-N
empty	300	12.2	13.8	15.2	12.6	12.6	15.4	26.9	13.3	18.2	12.9
	350	25.6	28.6	31.5	26.5	27.7	32.9	42.9	28.5	32.8	28.9
	400	49.1	53.7	59.6	50.0	50.1	64.5	70.0	54.2	63.9	56.8
	450	90.5	95.4	101.8	90.7	91.1	105.0	112.2	96.5	109.9	107.5
	500	145.39	143.9	155.5	140.4	143.9	167.1	179.7	150.8	174.7	186.4
random	150	10.5	11.4	11.3	10.4	10.8	10.6	11.9	10.3	11.0	10.4
	200	25.4	26.2	27.0	25.6	25.4	27.3	30.4	24.9	26.3	24.2
	250	50.4	53.2	53.3	50.6	50.9	59.1	57.5	49.3	58.3	54.8
	300	93.8	98.6	100.1	93.9	94.6	104.9	108.4	92.0	107.0	115.5
	350	150.4	155.5	167.8	150.1	156.6	175.8	193.5	155.8	186.3	206.3
warehouse	150	3.7	7.0	4.8	3.6	3.6	4.6	8.9	3.6	4.0	5.5
	200	8.1	15.6	12.2	8.4	8.7	9.7	17.7	8.0	9.8	10.5
	250	15.3	32.8	23.4	16.2	16.4	19.3	31.4	14.1	21.1	17.7
	300	25.6	53.7	42.0	29.1	27.9	34.3	57.3	23.8	36.8	31.1
	350	41.3	84.6	66.2	44.1	44.7	65.7	77.1	38.1	64.6	56.1
ost003d	200	6.8	14.4	9.9	6.3	5.5	6.6	6.6	8.3	9.9	12.0
	300	19.9	45.6	36.5	21.3	13.9	23.9	15.8	33.4	34.3	39.2
	400	48.3	100.6	86.3	51.0	39.3	84.3	41.2	75.2	106.5	109.0
	500	142.9	223.1	193.7	141.0	103.4	217.0	121.8	163.6	223.4	270.5
	600	318.3	404.8	399.2	323.0	260.7	459.1	294.6	311.4	457.2	517.9
den520d	500	22.8	72.4	57.9	26.1	14.1	28.0	20.4	47.8	60.7	87.8
	600	44.4	116.0	101.5	50.2	26.5	46.3	37.0	89.2	117.2	173.4
	700	80.3	175.9	171.7	88.0	48.3	77.9	68.2	150.2	200.1	286.7
	800	123.0	234.8	258.2	131.9	75.4	142.8	99.7	209.4	312.9	431.3
	900	166.7	347.7	345.6	187.9	130.6	216.5	168.1	308.8	420.2	624.8
Paris	350	2.9	9.4	6.5	3.3	2.7	6.3	10.7	4.6	7.4	38.9
	450	4.9	22.2	17.2	6.6	4.5	12.2	7.1	11.0	20.3	42.9
	550	9.1	38.3	33.7	11.7	7.4	16.5	12.2	20.7	37.6	67.7
	650	14.0	66.7	57.1	17.7	12.2	35.7	18.7	37.0	78.2	128.7
	750	25.4	110.9	99.1	33.7	21.8	65.3	36.6	70.6	126.6	243.5

Table 12: Final delays of different methods with best neighborhood size, evaluated on maps with differing numbers of agents within 300 seconds. Initial solutions are provided by LaCAM2. Highlighted are the results ranked first, and second.

map	n	RNWLK	INTC	RAND	ADP	RNWLKPB	Our-S	Our-N	Bandit	Orig-S	Orig-N
empty	300	350.5	402.1	441.9	369.0	368.1	374.8	790.5	389.3	397.0	371.0
	350	740.3	824.2	930.2	758.3	759.4	823.0	1296.4	844.4	880.2	826.4
	400	1447.9	1593.4	1756.2	1476.5	1484.4	1692.4	2094.8	1579.0	1857.5	1614.3
	450	2638.0	2779.5	3040.7	2684.1	2681.5	3020.0	3598.3	2828.1	3506.0	3318.3
	500	4237.7	4448.5	4828.2	4374.5	4335.7	5194.9	5808.0	4772.6	5793.6	6085.3
random	150	327.4	344.3	348.4	324.6	331.0	332.9	361.7	326.5	337.6	330.2
	200	751.1	805.8	836.8	770.6	767.8	780.8	887.4	732.4	777.5	770.6
	250	1472.7	1537.2	1613.5	1506.7	1494.1	1521.2	1747.1	1456.3	1604.8	1698.5
	300	2547.4	2769.2	2811.8	2622.6	2663.6	2800.1	3171.0	2674.4	3084.9	3435.9
	350	4341.6	4622.2	4683.9	4406.5	4367.5	5162.7	5886.5	4598.5	5983.6	7438.8
warehouse	150	114.0	127.6	121.3	111.4	112.8	112.0	188.7	110.3	114.8	206.5
	200	247.5	309.6	281.0	244.2	249.1	248.8	420.8	239.4	250.6	375.5
	250	430.0	698.5	562.4	424.9	446.7	444.6	782.1	426.2	445.1	961.5
	300	680.6	1264.3	973.9	673.7	703.5	673.7	1423.9	695.9	716.2	2248.7
	350	1043.0	2011.0	1509.0	1055.0	1130.8	1042.4	1907.0	1067.6	1097.6	5255.1
ost003d	200	155.6	299.4	183.3	153.3	147.4	147.5	162.3	170.1	154.7	496.8
	300	333.8	868.4	611.8	333.4	296.1	297.9	339.3	496.5	359.8	1698.9
	400	714.7	1849.6	1488.2	805.2	622.9	762.5	675.3	1253.3	1083.7	5277.4
	500	2341.8	4338.9	3900.2	2067.5	1637.6	4386.8	2379.1	3023.9	3719.0	12525.6
	600	7514.0	10394.8	12466.1	8216.9	5875.0	13066.6	8469.1	7244.6	10153.9	25804.5
den520d	500	311.1	1433.0	972.7	311.5	241.6	240.1	322.6	764.6	440.5	3437.9
	600	546.3	2289.3	1649.3	597.4	386.6	381.0	513.4	1235.4	1353.4	8231.3
	700	933.1	3163.0	2782.7	1075.2	622.4	660.7	816.6	2439.2	2259.0	15529.0
	800	1548.6	4437.5	4337.2	1574.1	911.5	1003.6	1323.8	3671.6	4906.7	21966.5
	900	2486.1	6426.0	6923.5	2819.9	1382.0	2058.6	2067.1	5595.4	15199.0	31694.0
Paris	350	87.4	130.6	84.4	78.1	75.8	79.0	181.7	83.6	80.0	1285.3
	450	132.5	274.9	174.6	123.3	118.4	119.7	144.4	133.4	133.0	1483.0
	550	198.0	776.5	466.9	194.8	183.3	184.7	228.9	210.2	200.6	2643.7
	650	276.7	960.1	758.5	275.4	257.5	260.6	319.2	337.4	319.8	7697.9
	750	373.2	1821.2	1362.6	385.8	366.0	369.9	498.8	627.3	616.3	13831.5

Table 13: AUC (divided by 10k) of different methods with best neighborhood size, evaluated on maps with differing numbers of agents within 300 seconds. Initial solutions are provided by LaCAM2. Highlighted are the results ranked first, and second.

map	n	RNWLK	INTC	RAND	ADP	RNWLKPB	Orig-S	Orig-N	Bandit	Our-S	Our-N
empty	300	12.30	13.96	15.75	13.01	13.02	24.99	18.86	13.87	16.44	30.72
	350	25.87	29.14	32.77	26.95	27.02	52.64	40.35	30.19	40.51	48.71
	400	52.87	58.25	61.44	53.10	53.88	97.55	74.40	57.57	83.42	81.62
	450	96.16	100.94	108.19	97.18	98.40	166.13	143.87	103.12	140.33	138.05
	500	155.66	162.94	171.83	156.47	159.82	243.96	242.91	173.84	223.50	221.09
random	150	10.44	11.35	11.20	10.47	10.69	12.26	11.89	10.34	11.67	12.13
	200	25.48	26.41	27.50	25.21	26.12	30.54	30.86	23.83	32.90	31.21
	250	50.15	53.64	54.29	50.95	50.56	67.43	70.88	48.61	60.13	63.64
	300	91.89	99.68	100.09	92.77	95.27	138.76	147.19	92.31	119.79	122.59
	350	169.08	184.56	186.63	163.77	166.60	262.86	312.23	169.33	241.24	241.76
warehouse	150	4.29	7.63	5.52	4.24	4.19	8.01	22.75	4.23	8.91	10.21
	200	9.16	19.09	13.95	10.17	10.07	18.44	57.70	9.03	16.10	21.61
	250	18.10	47.96	29.80	19.31	20.09	36.29	127.21	16.62	41.10	38.25
	300	34.30	86.34	55.97	34.74	36.94	70.50	214.65	29.37	69.74	78.90
	350	56.83	158.73	92.99	63.12	67.02	112.19	361.57	49.67	100.43	101.40
ost003d	200	7.52	15.66	10.34	6.77	5.89	14.99	44.82	9.45	8.84	7.01
	300	18.76	43.39	35.94	19.59	15.13	52.23	143.11	33.45	27.63	17.39
	400	49.27	97.46	91.63	55.20	38.92	123.70	310.16	78.42	96.38	46.35
	500	191.43	243.15	231.61	157.80	137.00	312.88	573.88	176.63	314.07	171.56
	600	450.57	529.43	613.46	469.61	383.28	625.01	962.95	400.06	618.02	465.00
den520d	500	27.63	72.94	62.82	29.11	16.36	102.51	272.11	64.37	33.35	26.09
	600	52.28	118.85	109.20	55.62	31.23	228.15	469.45	95.94	75.86	50.19
	700	86.97	169.04	176.17	91.26	55.41	327.11	726.01	165.78	138.12	85.94
	800	128.40	244.98	266.43	132.94	95.21	499.13	952.23	244.88	200.86	147.81
	900	201.86	351.11	396.91	225.56	151.59	723.88	1246.97	346.67	335.36	217.58
Paris	350	3.97	10.39	8.31	4.38	3.57	20.02	78.72	6.69	21.32	14.23
	450	6.45	22.83	18.98	7.87	5.73	46.59	134.53	15.00	15.98	9.47
	550	10.72	58.17	48.55	15.25	9.73	105.33	261.44	28.72	33.00	17.10
	650	16.89	74.01	72.28	20.71	14.05	145.73	459.33	50.05	70.01	28.38
	750	27.46	123.22	115.02	39.40	23.87	261.45	656.43	87.12	108.04	47.80

Table 14: 'Uni-Bandit' represents our modified version of Bandit, where it randomly selects a neighborhood size from $2, 4, 8, 16, 32$ at each iteration instead of using the neighbourhood selection arm. The better results are marked in bold. The performance of Uni-Bandit and Bandit is very close. This suggests that a better strategy for choosing neighborhood size could lead to improvements.

		Delay		AUC				Delay		AUC	
map	n	Bandit	Uni-Bandit	Bandit	Uni-Bandit	map	n	Bandit	Uni-Bandit	Bandit	Uni-Bandit
empty	300	**386.3**	391.7	**13.3**	13.6	random	150	**330.1**	330.5	10.3	**10.3**
	350	**811.5**	812.4	**28.5**	28.5		200	779.1	**778.7**	24.9	**24.8**
	400	**1537.2**	1547.1	**54.2**	54.7		250	**1507.3**	1525.7	**49.3**	49.7
	450	**2753.7**	2761.2	96.5	**96.0**		300	**2746.0**	2760.5	**92.0**	92.0
	500	4318.5	**4302.1**	150.8	**149.5**		350	**4564.1**	4565.1	**155.9**	155.1
warehouse	150	**107.9**	111.8	**3.6**	3.7	ost003d	200	**158.2**	163.8	8.3	**8.3**
	200	239.4	**234.3**	8.0	**7.9**		300	532.8	**484.3**	33.4	**31.2**
	250	414.2	**414.0**	**14.2**	14.5		400	**1276.3**	1327.3	75.2	**75.0**
	300	**669.5**	677.2	**23.8**	24.1		500	3059.8	**2873.5**	163.6	**155.8**
	350	1047.7	**1042.6**	38.2	**39.1**		600	**6093.7**	6430.8	**311.4**	321.9
den520d	500	607.8	**593.6**	47.8	**47.3**	Paris	350	**71.9**	74.0	**4.6**	5.0
	600	1247.0	**1234.2**	89.2	**87.9**		450	130.8	**120.8**	11.0	**10.1**
	700	2297.4	**2195.9**	150.2	**144.2**		550	**205.3**	212.5	**20.7**	21.4
	800	**3330.3**	3607.8	**209.4**	221.5		650	307.5	**303.0**	37.0	**36.9**
	900	**5343.5**	5421.0	**308.9**	310.6		750	577.0	**523.6**	70.6	**66.2**

References

Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

Taoan Huang, Jiaoyang Li, Sven Koenig, and Bistra Dilkina. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, pp. 9368–9376, 2022.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142, 2002.

Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 12353–12362, 2021.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10256–10265, 2022.

Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pp. 243–251, 8 2023.

Qandeel Sajid, Ryan Luna, and Kostas Bekris. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *Proceedings of the International Symposium on Combinatorial Search*, volume 3, pp. 88–96, 2012.

Zhongxia Yan and Cathy Wu. Neural neighborhood search for multi-agent path finding. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=2NpAw2QJBY.