MAPF-EVO: LLM-DRIVEN EVOLUTION OF CONFLICT ELIMINATION HEURISTICS FOR MULTI-AGENT PATH FINDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Neighborhood Search (LNS) is one of the leading approaches for solving large-scale Multi-Agent Path Finding (MAPF), but its performance depends heavily on destroy heuristics—the strategy for selecting which agents to replan. Existing methods mostly rely on hand-crafted, fixed heuristics that cannot adapt well to different problem instances, whereas learning-based methods such as Neural Neighborhood Search (NNS) require training a separate model for each map type, which limits their generality. Therefore, we propose a Large Language Model (LLM) driven evolutionary framework (MAPF-Evo) that automatically designs interpretable destroy heuristics for LNS. By feeding the solver's runtime state back to the LLM, MAPF-Evo evolves heuristics that leverage detailed conflict-structure information to balance solution quality and speed. Experiments on five types of map (40 scenarios in total) show that, compared with the recent best method, MAPF-LNS2, adding the heuristic evolved by MAPF-Evo to the original heuristic pool reduces average remaining conflicts by about 45.0% and improves resolution speed by 11.6%. Gains are especially significant on structurally complex maps such as maze and room. This work is the first to introduce LLM-driven heuristic design into LNS-based MAPF solving, offering a more general and less manually tuned solution to this problem.

1 INTRODUCTION

Multi-agent path finding (MAPF) plans collision-free paths for multiple agents in a shared environment (Stern et al., 2019). It is a core computational problem in automated warehousing, traffic management, and multi-robot coordination. Optimal MAPF is NP-hard (Yu & LaValle, 2013). With hundreds to thousands of agents, exact and bounded-suboptimal algorithms face scalability bottlenecks and often struggle to find feasible solutions within limited time and memory budgets. To address the scalability challenge, Large Neighborhood Search (LNS) provides an iterative destroy-and-repair framework that balances solution quality and computational efficiency (Shaw, 1998). In MAPF, the anytime LNS framework iteratively selects subsets of agent paths for replanning to improve solution quality (Li et al., 2021). MAPF-LNS2 applies this idea to large-scale MAPF, starting from potentially conflicting paths and repeatedly replanning agent subsets until a collision-free solution is found (Li et al., 2022). Its scalability and strong performance have made MAPF-LNS2 a representative baseline and a foundation for subsequent MAPF-LNS methods.

However, each iteration's effectiveness depends on the destroy heuristic, which selects agents for replanning. Existing approaches to designing this component face fundamental limitations. Hand-crafted fixed heuristics, such as the Target, CollisionGraph, and Random strategies in MAPF-LNS2 (Li et al., 2022), rely on expert intuition and cannot adapt to the structural properties of specific problem instances. Offline methods such as Machine Learning-guided Large Neighborhood Search (MAPF-ML-LNS) (Huang et al., 2022) use handcrafted features to train ranking models that predict candidate neighborhoods' improvement potential. However, they require substantial pre-collected data and have limited generalization. Neural Neighborhood Search (NNS) (Yan & Wu, 2024) uses a convolution-attention architecture to learn neighborhood scores end-to-end from subproblem spatiotemporal structure. However, it depends on specific map sizes and problem configurations, typically requiring new data collection and retraining when settings change. More fundamentally, these

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

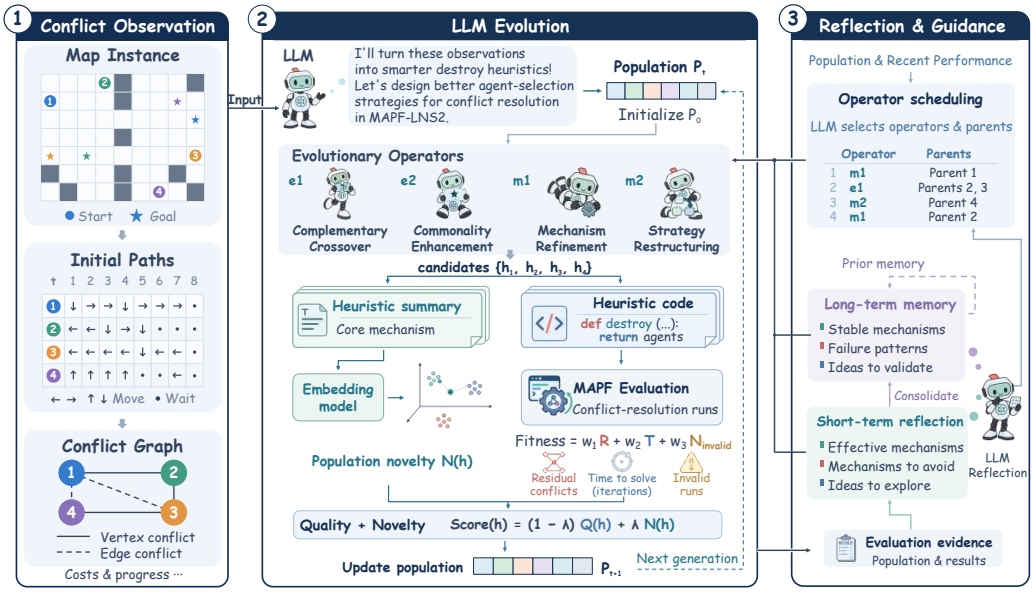


Figure 1: **Our MAPF-Evo framework.** Using inputs such as conflict observations, solver feedback, and the current population, the LLM selects parents and operators for crossover or mutation. Candidate destroy heuristics are evaluated in MAPF-LNS2. Evaluation quality and embedding-based novelty guide population updates. Short- and long-term reflection guide parent and operator selection in the next generation.

methods only rerank or select neighborhoods generated by fixed rules, leaving the candidate space constrained by handcrafted designs. As a result, current methods adapt mainly by selecting or reranking predefined strategies and have yet to extend this capability to discovering new heuristics. How to automatically construct and continuously evolve destroy strategies using search feedback remains an open problem.

In this work, we further observe that many subsequent MAPF-LNS extensions focus on improving the Sum of Costs (SOC) of an already collision-free solution, while assuming that initial conflicts have been resolved by existing heuristics. However, conflict elimination is a necessary prerequisite for the entire solving pipeline. If the initial conflicts cannot be eliminated within the available budget, subsequent solution-quality optimization cannot begin (Salzman & Stern, 2020; Surynek, 2022). Recent advances in LLM-driven heuristic design offer a promising way to reduce the dependence on hand crafted rules (Romera-Paredes et al., 2024; Liu et al., 2024; Ye et al., 2024; Wang et al., 2025a). Inspired by these advances, we propose MAPF-Evo. Our framework moves beyond selecting candidate neighborhoods to designing the rules that construct them. It uses solver feedback to automatically generate and evolve destroy heuristics for MAPF conflict elimination.

Our main contributions are as follows:

- We propose MAPF-Evo, which combines an LLM with solver feedback to automatically synthesize and evolve destroy heuristics for MAPF conflict resolution.
- We design a fitness function that accounts for both residual conflicts and the search effort needed to find the first feasible solution, guiding heuristic evolution under a limited budget.
- We achieve the highest gains in conflict resolution efficiency and success rates over a pool of hand-crafted heuristics across multiple MAPF benchmark map types, demonstrating the potential of integrating self-evolving heuristics with LLMs for solving MAPF.

2 BACKGROUND

2.1 MULTI-AGENT PATH FINDING

MAPF is defined by $\mathcal{P} = (G, \mathbf{s}, \mathbf{g})$, where $G = (V, E)$ is an undirected unweighted graph, and $\mathbf{s}$ and $\mathbf{g}$ specify all agents' start and goal locations. At each discrete timestep, an agent moves along an edge or waits at its current vertex. Each path connects an agent's start to its goal. A vertex conflict occurs when two agents simultaneously occupy the same vertex. An edge conflict occurs when two agents traverse an edge in opposite directions within the same timestep. A set of paths is feasible if it contains no conflicts. While standard MAPF also considers optimizing path cost subject to collision-free constraints, this paper focuses on quickly eliminating conflicts from an initial set of conflicting paths and obtaining a feasible solution.

2.2 LARGE NEIGHBORHOOD SEARCH

Large Neighborhood Search (LNS) iteratively destroys and repairs part of a solution to improve it (Shaw, 1998). MAPF-LNS2 applies this framework to potentially conflicting initial paths (Li et al., 2022). In its conflict-elimination stage, a destroy heuristic selects k agents at each iteration. Their paths are removed and replanned using Prioritized Planning (PP) with Safe Interval Path Planning with Soft constraints (SIPPS) (Silver, 2005; Phillips & Likhachev, 2011), while all other paths remain fixed. Updates are accepted if the number of colliding agent pairs (CP) does not increase. The process stops when $CP = 0$ or the iteration budget is exhausted. MAPF-LNS2 provides three hand-crafted destroy heuristics: Target, CollisionGraph, and Random. Adaptive Large Neighborhood Search (ALNS) selects among them based on historical performance (Ropke & Pisinger, 2006). The destroy heuristic determines the replanning scope and directly affects conflict elimination efficiency. We build on this procedure to automatically generate and evolve more effective destroy heuristics.

3 RELATED WORK

3.1 MAPF PLANNERS

MAPF-LNS (Li et al., 2021) was the first to introduce LNS to MAPF. It improves solution quality by iteratively destroying and replanning the paths of selected agents. MAPF-LNS2 (Li et al., 2022) starts from an initial path set that may contain conflicts and reduces the number of colliding pairs to obtain a collision-free solution. It uses ALNS to adaptively select among Target, CollisionGraph, and Random. Subsequent work mainly improves the selection of existing heuristics or candidate neighborhoods. MAPF-ML-LNS (Huang et al., 2022) uses hand-crafted features to train a linear ranking model offline that predicts the improvement potential of candidate neighborhoods. Bandit-based Adaptive Large Neighborhood search Combined with Exploration (BALANCE) (Phan et al., 2024) uses a two-level multi-armed bandit scheme to jointly select destroy heuristics and neighborhood sizes online. NNS (Yan & Wu, 2024) learns a neighborhood scoring function using convolutional and attention mechanisms. Truncated Adaptive Counterfactual K-ranked LEarning (TACKLE) (Phan et al., 2026) combines counterfactual learning with structural information from hand-crafted strategies to select promising seed agents online. These methods make selection more adaptive, but candidate neighborhood construction still relies on predefined rules.

Another line of work directly incorporates learning models into MAPF planners. LNS2+RL (Wang et al., 2025b) introduces multi-agent reinforcement learning into low-level replanning. It switches back to prioritized planning in later search iterations to balance coordination and computational efficiency. MAPF-GPT (Andreychuk et al., 2025) trains a Transformer model on large-scale expert trajectories through imitation learning. It supports decentralized planning without additional heuristics or communication and demonstrates zero-shot generalization. LaGAT (Jain et al., 2026) integrates an enhanced Message-Aware Graph Attention Network (MAGAT) policy into the lazy constraints addition search (LaCAM) framework, combining the strengths of learned guidance and search in dense scenarios.

3.2 LLM-DRIVEN HEURISTIC DESIGN

The program synthesis capabilities of LLMs offer new opportunities for automatic heuristic design. FunSearch (Romera-Paredes et al., 2024) combines LLMs with evolutionary search to discover new mathematical constructions through program search. Evolution of Heuristics (EoH) (Liu et al., 2024) uses LLMs as evolutionary operators to iteratively generate and improve combinatorial optimization heuristics. HSEvo (Dat et al., 2025) combines harmony search with genetic operations to balance population diversity and search convergence. Multi-objective Evolution of Heuristic (MEoH) (Yao et al., 2025) formulates heuristic design as a multi-objective optimization problem that considers both solution quality and computational efficiency. It also uses differences between heuristics to guide population management. These studies demonstrate the potential of LLMs to automatically generate and improve combinatorial optimization heuristics. Building on this direction, we use solver feedback to generate and evolve destroy heuristics for MAPF.

4 MAPF-EVO

MAPF-Evo combines an LLM with solver feedback to automatically design destroy heuristics for conflict resolution in MAPF-LNS2. As shown in Figure 1, the LLM generates candidates through crossover and mutation. The solver evaluates their conflict-resolution performance, and population selection considers both solving quality and semantic novelty. Short-term reflection and long-term memory use evaluation feedback to guide subsequent generation.

4.1 DESTROY HEURISTIC REPRESENTATION

Each candidate consists of an executable destroy program h and a natural language description d_h. The description summarizes the design idea, core selection rule, and inputs used by the candidate code. It provides the basis for measuring semantic novelty in Section 4.3. The program integrates with MAPF-LNS2 through a common interface and selects agents for replanning based on the runtime state.

Let $\mathcal{A}$ denote a set of n agents, and let r denote the solver's search iteration. The inputs available to the heuristic include instance information, current paths, conflict structure, search progress, and past feedback, among others. These inputs are collectively denoted by o_r. Given a selection size k, the candidate returns a replanning set $S_r = h(o_r, k) \subseteq \mathcal{A}$ containing $\min(k, n)$ agents. Conflict structure is a central component of o_r. The conflict graph $\mathcal{G}_r^c = (\mathcal{A}, E_r^c)$ represents agents as nodes. An edge connects two agents whose current paths conflict. The interface provides adjacency information, the number of conflict partners for each agent, and connected components. For each colliding agent pair, it also records the first detected conflict, including its time, type, and location.

Additional inputs include map topology, current paths, and spatiotemporal occupancy statistics. Search progress and recent replanning feedback reveal stagnation and whether previous selections yielded accepted improvements. The program can combine this information with history retained within the current run to adjust subsequent selections.

4.2 FEEDBACK-DRIVEN HEURISTIC EVOLUTION

The LLM generates initial candidates from the task description and the common interface. Evaluation and selection form the initial population P_0. Candidate evaluation and population updates follow the scoring mechanism in Section 4.3.

Candidate generation uses four evolutionary operators. Complementary crossover (e_1) combines complementary selection mechanisms from two parents. Commonality enhancement (e_2) extracts their shared design idea and addresses its limitations. Mechanism refinement (m_1) retains a single parent's main selection approach while improving key steps. Strategy restructuring (m_2) revisits the parent's selection rationale to explore different ways of grouping agents for replanning.

Each generation, the LLM jointly selects an operator and its parents for each call. Scheduling considers mechanism descriptions and fitness values in the current population. It also uses recent candidate improvements, selection outcomes, operator performance, and short-term reflection. Operators e_1 and e_2 use two parents, while m_1 and m_2 use one. An operator may be called multiple times

within a generation, and different calls may share parents. This allows feedback to guide candidate generation. The LLM generates candidate programs and descriptions from operator instructions and parent programs and descriptions. Crossover operators focus on combining existing mechanisms. Mutation operators focus on local improvement or redesign of a single heuristic. Together, they expand the range of candidate designs.

After evaluation, the LLM generates short-term reflection by comparing parent and offspring programs and their per-scenario results. It identifies effective mechanisms, designs that may cause performance degradation, and directions for exploration. The LLM then updates long-term memory with reusable insights from this reflection. It extracts experience that can be reused across subsequent generations and updates the memory. Short-term reflection focuses on recent generation outcomes, while long-term memory accumulates experience across generations. Both guide candidate generation.

4.3 Solver Evaluation and Scoring

Each candidate h is evaluated on a fixed set of training scenarios $\mathcal{D} = \{I_1, \ldots, I_M\}$. During evaluation, MAPF-LNS2 replans the paths of agents in S_r. It applies its existing acceptance criterion to decide whether to update the current solution. Candidate performance is measured over the full conflict-resolution process. Fitness considers both residual conflicts and the iteration cost of finding the first feasible solution. These terms capture progress on unsolved scenarios and efficiency on solved scenarios, respectively.

For scenario I_j, the conflict score $R_j(h)$ is computed based on the ratio of the final to the initial number of colliding agent pairs. It is zero when all conflicts are eliminated. The iteration-cost score $T_j(h)$ is obtained by normalizing the number of replanning iterations needed to find the first feasible solution. It is set to 1 if the scenario is not solved within budget. The normalization assigns a strictly lower score to a run solved within budget than to an unsolved run. Let $R(h) = M^{-1} \sum_{j=1}^{M} R_j(h)$ and $T(h) = M^{-1} \sum_{j=1}^{M} T_j(h)$. The overall fitness is

$$F(h) = w_1 R(h) + w_2 T(h) + w_3 N_{\text{invalid}}(h), \tag{1}$$

where $N_{\text{invalid}}(h)$ counts scenarios with timeouts, crashes, or output constraint violations. The weights w_1, w_2, and w_3 control the contributions of residual conflicts, iteration cost, and execution failures, respectively. Lower fitness indicates better overall performance.

Fitness is normalized and reversed to obtain a quality score $Q(h)$, with larger values indicating better solving performance. To maintain heuristic diversity during evolution, selection also considers the semantic novelty of candidate descriptions. The LLM produces each description alongside the program to explain how the candidate uses the solver state to select agents. Embedding distances between descriptions therefore serve as a proxy for differences in heuristic design. Let ϕ denote a text embedding encoder, and let $z_h = \phi(d_h)$ denote the description embedding. Let $\mathcal{U}$ be the set of heuristics already selected for the next generation. Candidate novelty is defined as

$$N(h \mid \mathcal{U}) = \min_{g \in \mathcal{U}} \left(1 - \frac{z_h^\top z_g}{\|z_h\|_2 \|z_g\|_2}\right). \tag{2}$$

This is the minimum cosine distance between the candidate description and the selected descriptions. A larger value indicates greater semantic distance even from the closest selected description. This criterion reduces the retention of similar designs in the population. Quality and novelty jointly determine the selection score:

$$\text{Score}(h \mid \mathcal{U}) = (1 - \lambda) Q(h) + \lambda N(h \mid \mathcal{U}), \tag{3}$$

where λ is adjusted according to the fitness spread in the candidate pool. When candidates have similar fitness, a larger novelty weight encourages exploration. When fitness differences are larger, selection places greater emphasis on solving quality.

The next population is selected from the parents and newly generated valid candidates. Selection first retains the candidate with the lowest fitness to initialize $\mathcal{U}$. It then repeatedly adds the remaining candidate with the highest combined score until the target population size is reached. After each addition, novelty is recomputed relative to the updated $\mathcal{U}$. This process preserves the best current heuristic while considering the quality and diversity contributions of the other candidates. At the end of evolution, the valid heuristic with the lowest fitness is returned for use in MAPF-LNS2.

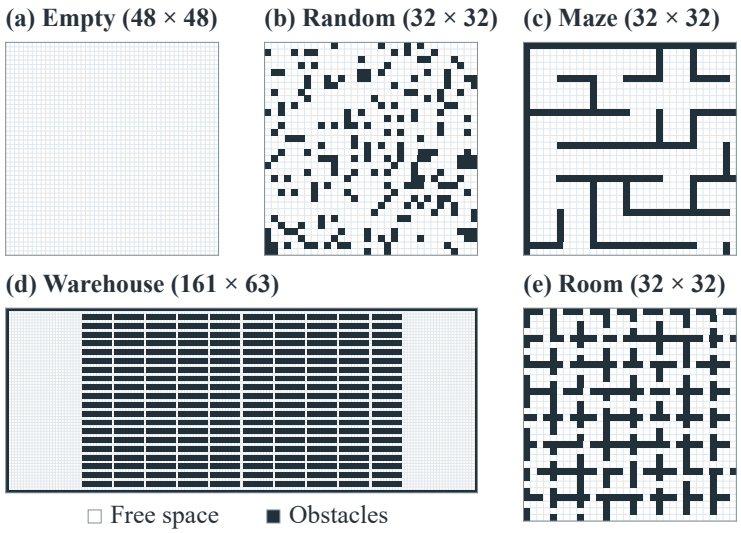


Figure 2: **Benchmark maps used in the experiments.** The five maps are (a) Empty, (b) Random, (c) Maze, (d) Warehouse, and (e) Room. White cells represent free space, and dark cells represent obstacles. Each panel reports the map dimensions (width $\times$ height). The maps cover open space, scattered obstacles, maze corridors, warehouse aisles, and connected rooms.

Table 1: **Conflict elimination performance under a 100-second budget.** best-only uses only the best evolved heuristic, while best-ALNS adds it to the original ALNS heuristic pool. Values are means $\pm$ sample standard deviations of per-seed aggregate results across three solver seeds.

Method	Success rate (%)	Conflicting pairs	Resolution time (s)
MAPF-LNS2	62.50 ± 0.50	154.85 ± 31.69	48.44 ± 0.72
BALANCE	55.00 ± 3.82	158.25 ± 22.43	59.09 ± 0.48
NNS	32.50 ± 0.00	279.50 ± 31.72	79.83 ± 0.63
LNS2+RL	55.00 ± 0.00	570.88 ± 20.06	72.24 ± 0.54
TACKLE	47.50 ± 2.50	164.65 ± 47.13	62.95 ± 1.95
MAPF-Evo (best-only)	60.00 ± 1.44	104.00 ± 43.02	46.19 ± 1.20
MAPF-Evo (best-ALNS)	$\mathbf{65.00 \pm 0.00}$	$\mathbf{85.15 \pm 34.12}$	$\mathbf{42.80 \pm 0.60}$

5 Experimental Setup

We briefly describe the experimental setup in the main text and provide full details in Appendix C. The experiments evaluate how effectively MAPF-Evo resolves conflicts in an initial path set and finds a feasible solution. They also examine how key mechanisms affect the heuristic evolution process.

Benchmarks and Data Splits. The experiments use five maps from the Moving AI MAPF benchmark (Stern et al., 2019) with substantial conflicts after initialization: Empty (48×48), Random (32×32), Maze (32×32), Room (32×32), and Warehouse (161×63). Figure 2 shows their layouts. The training set contains 50 scenarios (10 per map) for heuristic evolution and candidate evaluation. The test set contains 40 separate scenarios (8 per map), used exclusively for final evaluation.

Baselines and Initialization. We compare MAPF-Evo with MAPF-LNS2 (Li et al., 2022), NNS (Yan & Wu, 2024), BALANCE (Phan et al., 2024), TACKLE (Phan et al., 2026), and LNS2+RL (Wang et al., 2025b). These baselines cover conventional large neighborhood search, learning-

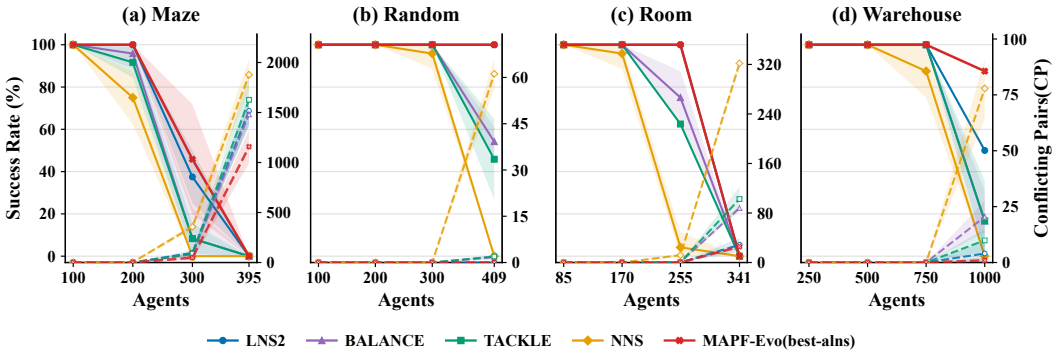


Figure 3: **Conflict elimination performance across agent counts under a 100-second budget.** Solid lines show success rates (left axes), and dashed lines show mean final numbers of colliding agent pairs (right axes). Empty is omitted because all methods achieve 100% success with no residual conflicts at every tested agent count.

guided neighborhood selection, and reinforcement learning-assisted path repair. We adapt NNS, BALANCE, and TACKLE from path-cost optimization to conflict elimination by using the reduction in colliding agent pairs after replanning as the improvement measure. All methods start from the same initial paths with conflicts, generated by a common PP/SIPP initialization procedure. Implementation sources, adaptation details, and parameter settings are provided in Appendix C.2.

Evaluation Metrics. The primary metrics are success rate, the final number of colliding agent pairs, and conflict-resolution time. Success rate is the fraction of test runs that find a collision-free solution within the given time budget. The final number of colliding agent pairs counts agent pairs with remaining vertex or edge conflicts at termination. Each pair is counted only once, even if it has multiple conflicts. This metric captures conflict-resolution progress on unsolved scenarios. Conflict-resolution time is measured from the common initial path set until the first collision-free solution is found. For runs that remain unsolved, the recorded time is set to the time budget.

6 Experimental Results

6.1 Conflict Elimination Performance

To assess conflict elimination under limited time budgets, we evaluate MAPF-Evo and all baselines with repair budgets of 100 and 200 seconds. Table 1 reports the overall results at 100 seconds. Results at 200 seconds and for individual maps are provided in Appendix C.3. MAPF-Evo (best-only) leaves fewer colliding agent pairs on average than MAPF-LNS2 but solves fewer instances. Under the 100-second budget, MAPF-Evo (best-ALNS) achieves the highest success rate, the lowest mean final number of colliding agent pairs, and the shortest mean repair time among all compared methods. Relative to MAPF-LNS2, it increases the success rate by 2.50 percentage points and reduces the latter two metrics by 45.01% and 11.64%, respectively.

These results show that combining the evolved heuristic with the original heuristics yields better overall performance than using it alone. Different heuristics select different agent subsets for joint replanning. They may therefore suit different conflict structures and search states. ALNS adjusts their selection probabilities based on past reductions in colliding agent pairs. This favors effective strategies while retaining alternative ways to construct neighborhoods. The combination may exploit complementary strengths, increasing the chance of finding a collision-free solution within the time budget. These findings motivate future work on evolving a set of complementary heuristics to further improve conflict elimination through their combined use.

Table 2: **Conflict elimination performance of heuristics generated by different LLMs under two configurations.** Higher success rates and lower final conflict counts and repair times are better. Icons distinguish open-source 🤗 and closed-source 🤯 models.

Model	Configuration	Success rate (%)	conflict pairs	repair time (s)
DeepSeek-V4.1 Flash 🤗	best-only	35.00	346.10	67.28
	best-ALNS	62.50	100.98	46.27
GLM-5.3 Flash 🤗	best-only	60.00	137.08	50.16
	best-ALNS	62.50	117.73	46.18
GPT-5.6 Terra 🤯	best-only	45.00	181.43	59.44
	best-ALNS	**67.50**	83.50	43.34
Gemini-3.8 Flash 🤯	best-only	60.00	102.13	46.19
	best-ALNS	65.00	**82.98**	**42.77**

Table 3: **Mean heuristic call proportions in best-ALNS for each LLM.** EvolvedBest denotes the evolved best heuristic. Target, CollisionGraph, and Random are the original heuristics.

Model	Target	CollisionGraph	Random	EvolvedBest
DeepSeek-V4.1 Flash 🤗	20.87%	21.61%	23.45%	**34.07%**
GLM-5.3 Flash 🤗	21.83%	21.65%	22.39%	**34.13%**
GPT-5.6 Terra 🤯	20.36%	20.22%	22.15%	**37.26%**
Gemini-3.8 Flash 🤯	21.58%	20.83%	23.90%	**33.69%**

6.2 Performance Across Agent Counts

To assess how MAPF-Evo handles changes in agent count, the experiments use four agent counts on each of the five maps described above. The conflict elimination budget is fixed at 100 seconds. Figure 3 compares the success rates and mean final numbers of colliding agent pairs of MAPF-Evo (best-ALNS) and the baselines. Curves and shaded regions indicate means and sample standard deviations, respectively.

Figure 3 shows similar performance across methods at low agent counts. The advantages of MAPF-Evo (best-ALNS) become more apparent as the number of agents increases. It achieves the highest or joint-highest mean success rate and the lowest or joint-lowest mean final conflict count at every tested agent count on the four displayed maps. At the largest tested agent count in Warehouse, its success rate reaches 87.50%, compared with 50.00% for MAPF-LNS2. It also leaves fewer residual conflicts on the challenging Maze map. These gains may stem from the evolved best heuristic complementing the original strategies. This combination may help address the more complex conflict relationships that arise with more agents. However, some residual conflicts remain at the largest tested agent counts in Room and Warehouse. This leaves room to improve the ability to find fully collision-free solutions.

6.3 Heuristic Performance Across LLMs

To assess MAPF-Evo across different LLMs, the experiments use DeepSeek-V4.1 Flash, GPT-5.6 terra, GLM-5.3 Flash, and Gemini-3.8 Flash to generate heuristics. The final selected heuristic from each model is evaluated under best-only and best-ALNS. Table 2 shows that adding these heuristics to the original ALNS pool increases success rates to 62.50%–67.50% for all four models. It also reduces the mean final number of colliding agent pairs and mean repair time for each model. These results indicate that MAPF-Evo's effectiveness is not limited to a single LLM.

We further examine how often ALNS calls each heuristic. As shown in Table 3, the evolved best heuristic has the highest mean call proportion for all four models, ranging from 33.69% to 37.26%. ALNS adjusts selection probabilities based on past conflict elimination performance. Better-performing strategies therefore tend to receive more calls. Together with the performance

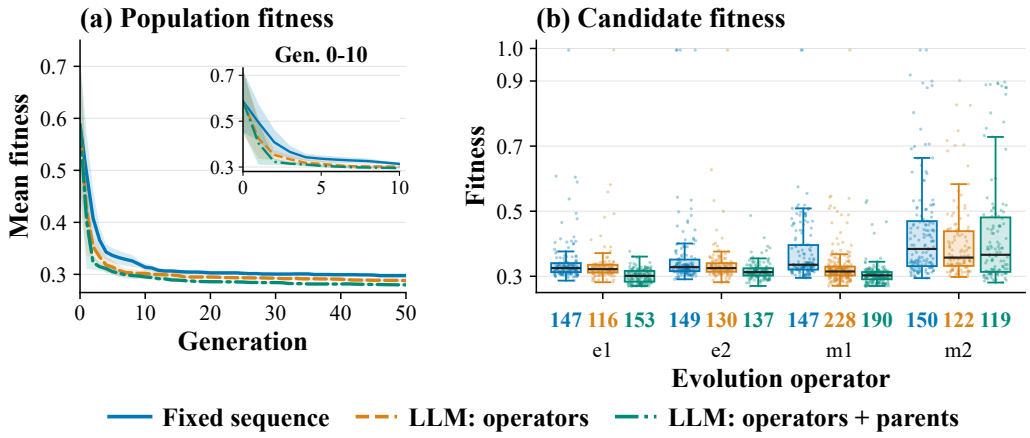


Figure 4: **Ablation of evolutionary scheduling strategies.** (a) Mean population fitness over generations. Curves and shaded regions indicate the mean and sample standard deviation across three runs. (b) Fitness distributions of valid candidates from each operator. Numbers above the operator labels indicate the total valid candidates under each scheduling strategy. Lower fitness is better.

gains of best-ALNS, these higher call proportions support the role of the evolved best heuristic in complementing the original neighborhood construction strategies.

6.4 Ablation of Evolutionary Scheduling

To examine the role of evolutionary scheduling, the experiments compare three schemes: fixed scheduling, LLM scheduling of operators only, and LLM joint scheduling of operators and parents. All schemes start from the same initial population and run independently three times for 50 generations each. Fixed scheduling calls e_1, e_2, m_1, and m_2 once per generation in this order. Under operator-only scheduling, the LLM allocates calls to each operator, while a predefined rule selects parents. Joint scheduling selects both operators and parents.

Figure 4(a) shows that operator-only and joint scheduling reduce mean population fitness by 3.24% and 5.96%, respectively, relative to fixed scheduling. The final best fitness follows the same trend, with joint scheduling achieving the best result. Figure 4(b) shows that joint scheduling produces candidates concentrated in regions of better fitness for several operators. These results indicate that allocating candidate-generation calls based on population state and evolutionary feedback improves the use of a limited number of calls. Joint scheduling further considers population quality, candidate feedback, and parent mechanisms to select better-matched operator–parent combinations. This helps reduce the generation of low-quality candidates and improves the use of the evolutionary budget.

7 Conclusion

We present MAPF-Evo, which combines an LLM with solver feedback to automatically generate and evolve conflict elimination heuristics for MAPF-LNS2. It uses conflict structure and search progress to guide candidate generation, and evaluates candidates based on residual conflicts and solving efficiency. Experiments across maps, agent counts, and scheduling settings consistently show that adding the evolved heuristic to the original ALNS pool improves conflict elimination under a limited budget. Dynamic scheduling of operators and parents further improves the quality of the evolved heuristics. The current MAPF-Evo pipeline retains only the single heuristic with the lowest fitness. It therefore does not fully exploit the complementarity among multiple candidates. Future work will evolve and dynamically deploy a complementary set of heuristics to improve complete conflict elimination in highly congested scenarios.

AI USE STATEMENT

We used generative AI tools for English language polishing and grammar checking. In addition, MAPF-Evo incorporates an LLM into its heuristic evolution framework to generate and refine candidate heuristics, schedule operators and parents, and produce evolutionary reflections. These uses are integral to the proposed algorithm and constitute the integration of generative AI into our research method. The method section details these mechanisms and their interactions with the solver. Experimental data and performance results were obtained from actual solver runs and statistical analysis. Generative AI tools were not used to fabricate or falsify them. These tools did not replace the authors' judgment in algorithm design, experimental setup, or scientific conclusions. All AI-assisted content was reviewed, revised, and verified by the authors. The authors take full responsibility for the final text, methods, data, results, and conclusions of this paper.

REPRODUCIBILITY STATEMENT

The method section describes MAPF-Evo's heuristic representation, evolutionary process, solver evaluation, and fitness function. It also details candidate generation, feedback use, population updates, and final heuristic selection. The experimental setup specifies the Moving AI MAPF benchmark, map and scenario splits, initial path generation, random seeds, time budgets, evaluation metrics, and baseline adaptations. Training scenarios are used for heuristic evolution and candidate evaluation, while disjoint test scenarios are used only for final performance evaluation. All methods start from identical initial paths and are compared under the same time budgets to reduce the impact of initialization differences. The appendix provides complete parameter settings, results under additional time budgets, and per-map results. The supplementary materials will include anonymized source code to support reproduction of the algorithm and experimental results.

REFERENCES

Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23126–23134, 2025.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025.

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*, volume 36, pp. 9368–9376, 2022.

Rishabh Jain, Keisuke Okumura, Michael Amir, and Amanda Prorok. Graph attention-guided search for dense multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29504–29512, 2026.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. Anytime multi-agent path finding via large neighborhood search. In *International joint conference on artificial intelligence 2021*, pp. 4127–4135. Association for the Advancement of Artificial Intelligence (AAAI), 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.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

Thomy Phan, Taoan Huang, Bistra Dilkina, and Sven Koenig. Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17514–17522, 2024.

Thomy Phan, Shao-Hung Chan, and Sven Koenig. Truncated counterfactual learning for anytime multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29633–29641, 2026.

Mike Phillips and Maxim Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE international conference on robotics and automation*, pp. 5628–5635. IEEE, 2011.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

Oren Salzman and Roni Stern. Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proceedings of the 19th international conference on autonomous agents and multiagent systems*, pp. 1711–1715, 2020.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.

David Silver. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, pp. 117–122, 2005.

Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, Thirunarayanapuram Krishnamachari Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the international symposium on combinatorial search*, volume 10, pp. 151–158, 2019.

Pavel Surynek. Problem compilation for multi-agent path finding: a survey. In *IJCAI*, pp. 5615–5622, 2022.

Hui Wang, Xufeng Zhang, and Chaoxu Mu. Planning of heuristics: Strategic planning on large language models with monte carlo tree search for automating heuristic optimization. *arXiv preprint arXiv:2502.11422*, 2025a.

Yutong Wang, Tanishq Duhan, Jiaoyang Li, and Guillaume Sartoretti. Lns2+ rl: Combining multi-agent reinforcement learning with large neighborhood search in multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23343–23350, 2025b.

Zhongxia Yan and Cathy Wu. Neural neighborhood search for multi-agent path finding. In *International Conference on Learning Representations*, volume 2024, pp. 48473–48494, 2024.

Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27144–27152, 2025.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Jingjin Yu and Steven LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pp. 1443–1449, 2013.

APPENDIX

CONTENTS

A Implementation Details of MAPF-Evo **12**

B Case Study of an Evolved Heuristic **14**

C Complete Conflict Elimination Results. **15**

A IMPLEMENTATION DETAILS OF MAPF-EVO

A.1 HEURISTIC INTERFACE AND RUNTIME STATE

MAPF-Evo provides candidate heuristics with the current solver state through a common interface. Each heuristic uses this information to select agents for joint replanning. MAPF-LNS2 then replans their paths and updates the solution. The interface provides five categories of information.

(1) Map and Current Path Information. This includes map topology, traversable locations, and each agent's current path. These inputs describe spatial constraints, local connectivity, and spatiotemporal relationships between paths. Heuristics can use narrow passages, congested regions, and path intersections to identify agents that may require coordinated replanning.

(2) Conflict Structure Information. This includes colliding agent pairs, each agent's number of conflict partners, and the adjacency relations and connected components of the conflict graph. It also provides the time, type, and location of the first detected conflict for each colliding pair. These inputs help identify agents involved in many conflicts and groups linked by conflict relationships. The interface also provides information about conflict-free agents near conflict regions. These agents can be considered as potential participants in joint replanning.

(3) Path Cost and Spatiotemporal Occupancy Information. This includes current path lengths, single-agent shortest-path lengths, their differences, and occupancy at different locations and timesteps. These inputs help assess detours and local congestion. Excess path length does not directly indicate the number of available alternative paths. This information supports neighborhood selection without changing the primary objective of conflict elimination.

(4) Search Progress Information. This includes the current iteration count, available budget, the lowest number of colliding pairs observed so far, and search stagnation. These inputs describe the remaining resources and recent progress in conflict elimination. A candidate may retain selection rules that have recently worked well or try different agent combinations when progress stalls. The heuristic itself determines how to make these adjustments.

(5) Selection History and Repair Feedback. This includes recently selected agent sets, whether replanning succeeded, whether an update was accepted, and changes in colliding pairs and path cost. The interface also supports custom history within the current solver run. Candidates can use past selections and their outcomes to guide subsequent decisions. Feedback distinguishes failure to generate a complete candidate, candidate rejection, and update acceptance. Each outcome applies to the entire replanning set and cannot be directly attributed to an individual agent.

A.2 Candidate Generation Operators and Their Roles

MAPF-Evo uses four candidate generation operators to guide the LLM in improving neighborhood construction. Complementary Crossover (e_1) and Commonality Enhancement (e_2) each take two parents. They focus on recombining complementary mechanisms and strengthening shared mechanisms, respectively. Mechanism Refinement (m_1) and Strategy Restructuring (m_2) each take one parent. They focus on improving an existing selection strategy and revising its core selection basis, respectively. Each operator uses parent programs, mechanism descriptions, and solver evaluation results to generate a new candidate. Table 4 summarizes their inputs and roles.

Table 4: **Inputs and roles of MAPF-Evo's candidate generation operators.** The parent count specifies the number of parent heuristics used in each call.

Operator	Parents	Role
Complementary Crossover (e_1)	2	Identify complementary mechanisms in agent selection, ranking, and neighborhood expansion across two parents. Use evaluation results to assess when these mechanisms are useful. Reorganize them and their interactions into a new heuristic with coherent decision logic.
Commonality Enhancement (e_2)	2	Extract a shared conflict elimination idea from the parents' program logic and evaluation results. Analyze when it works, its limitations, and its failure cases. Make targeted improvements while preserving the shared core mechanism.
Mechanism Refinement (m_1)	1	Retain the parent's main agent-selection approach. Use evaluation feedback to improve key steps, such as seed selection, candidate ranking and filtering, neighborhood expansion, and input use. Refine the mechanism around one clear improvement.
Strategy Restructuring (m_2)	1	Revisit the parent's core assumption about which agents should be jointly replanned. Identify conflict or path relationships that the original strategy may overlook. Develop a different primary selection basis and rebuild the neighborhood construction process around it.

Figure 5 presents prompt examples for the four operators. These examples summarize the shared inputs, operator-specific generation instructions, and common output format. The complete prompts incorporate the specific task, parent information, and reflection content.

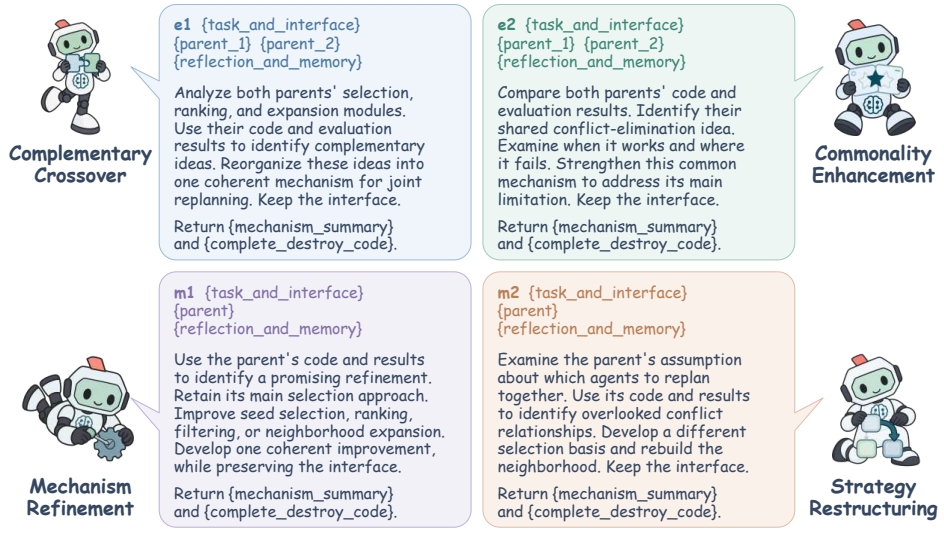


Figure 5: **Prompt examples for the four candidate generation operators.** Each prompt combines task and interface information, parent information, and reflection and memory. The operators generate a mechanism summary and a complete heuristic program while preserving the common interface.

B Case Study of an Evolved Heuristic

This section examines a best heuristic generated and evolved by GPT-5.6 terra within MAPF-Evo to illustrate its interpretable neighborhood construction mechanism. The candidate is an executable program whose state inputs, scoring rules, and selection conditions can be directly inspected. We describe the information it uses and compare it with Target, CollisionGraph, and Random to explain how it complements the original neighborhood construction strategies. Algorithm 1 summarizes its main procedure. The complete source code of this heuristic is provided in the supplementary materials.

(1) Conflict structure and repair history. The heuristic uses conflict graph adjacency, conflict partner counts, stagnation signals, nearby conflict-free agents, and repair feedback. Repair outcomes update per-agent fatigue values, lowering priority after ineffective repairs and raising it after accepted conflict reductions. Fatigue decays with new feedback. ALNS adjusts selection probabilities across heuristics, while best uses feedback to adjust agent selection within each neighborhood.

(2) Complementing Target. Target focuses on blocking relationships associated with start and goal locations. Best instead expands from a conflicting pair using links to the selected set, favoring candidates connected to several selected agents. Under stagnation, nearby conflict-free agents can also compete for a slot. This supplements Target's endpoint-based neighborhoods with choices guided by group connectivity and local coordination.

(3) Complementing CollisionGraph. CollisionGraph uses connected components and random walks. Best dynamically scores candidates using links to the selected set, external conflict links, and repair history. For the final two slots, it compares connected candidate pairs against a baseline that considers sequential expansion and a possible helper. This adds explicit combination assessment to the exploration provided by random walks, producing alternative neighborhoods from the same conflict graph.

(4) Complementing Random. Random samples agents with weights equal to their conflict partner counts plus one, preserving broad exploration. Best uses fatigue to discourage repeated ineffective choices and stagnation to activate helper selection. In its main expansion branch, a helper must be near at least two selected conflicting agents and compete with direct conflict candidates. These rules provide targeted adjustments based on repair history and local structure, complementing Random's broader sampling within the ALNS pool.

Algorithm 1 Neighborhood construction by the evolved best heuristic

Input: $G, n, b, \sigma, \mathcal{H}, M, f$
Output: Selected agent set S

$k \leftarrow \min(b, n)$; $S \leftarrow \emptyset$
if $k \in \{0, n\}$: **return** $\{a : 0 \le a < k\}$
UpdateFatigue(M, f)
$A \leftarrow \{a : \deg_G(a) > 0\}$
$e \leftarrow (k \ge 4 \wedge \sigma \ge 8)$; $u \leftarrow 0$
if $|A| \le k$ **then**
  $S \leftarrow A$; TryHelper($S, e, \mathcal{H}, M, k$)
  return SortFill(S, k)
$S \leftarrow$ SeedPair(G, M, k); refresh links
while $|S| < k$ **do**
  $F \leftarrow$ Frontier(G, S); score F
  $h \leftarrow$ Helper($\mathcal{H}, S, e, u, M$)
  if $k - |S| = 2$ **and** $|F| \ge 3$ **then**
    $P \leftarrow$ PairLookahead(F, h, S, M)
    if $P \ne \emptyset$: Add(P); **continue**
  if Wins(h, F): $a \leftarrow h$; $u \leftarrow 1$
  else if $F \ne \emptyset$: $a \leftarrow$ Best(F)
  else: $a \leftarrow$ Restart($A \setminus S, M$)
  if $a = \perp$: **break**; **else**: Add($\{a\}$)
return SortFill(S, k)

Notation. G: conflict graph; n: agent count; b: requested size; σ: stagnation count; $\mathcal{H}$: nearby agents; M: memory; f: repair feedback. e enables helper selection, and u records whether its slot has been used. All random choices use the supplied RNG.

Operations. UpdateFatigue processes each feedback ID once. SeedPair respects capacity k. TryHelper adds at most one helper if enabled and space remains. Helper returns an eligible helper only when its slot is unused. PairLookahead returns a pair only if its score exceeds the baseline plus a margin. Add updates S and its links. Restart returns $\perp$ if no candidate remains. SortFill randomly fills unused slots and sorts S.

C Complete Conflict Elimination Results

C.1 Experimental Hardware

The experimental hardware is summarized in Table 5.

Table 5: **Experimental hardware configuration.**

Component	Configuration
CPU	Intel Core i9-14900KF; 24 physical cores, 32 logical processors
Memory	64 GB (2×32 GB), KingBank
GPU	NVIDIA GeForce RTX 4090 D

C.2 Baseline Adaptation

NNS, BALANCE, and TACKLE are adapted to conflict elimination, primarily by replacing SOC reduction with a reduction in colliding agent pairs. BALANCE directly reuses the conflict elimination branch available in its official implementation. All three methods start from identical initial paths, and subsequent SOC optimization is disabled. The adaptations are described below, with key settings summarized in Table 6.

NNS. The Multi-Subset encoder, candidate ranking framework, and pairwise SVM ranking loss are retained. Training labels are changed from path-cost differences to $C(P') - C(P)$, where $C(P)$ denotes the number of colliding agent pairs in path set P. Lower values indicate greater improvement. Each iteration scores $J = 100$ candidate neighborhoods of eight agents. Only the predicted best neighborhood undergoes PP+SIPP repair. Since directly usable pretrained weights were not available from the public repository, a separate model is trained for each map. Training retains the original architecture and main optimization settings, with a budget of 10^5 updates. Figure 6 shows the training loss curves.

BALANCE. The official conflict elimination branch is reused. Its two-level Thompson sampling scheme, based on Normal-Gamma posteriors, selects a destroy heuristic and a neighborhood size. Both levels receive the accepted reduction in colliding agent pairs as the reward, or zero when no improvement is accepted. The four prior parameters retain their original values. The neighborhood size set is $\{2, 4, 8, 16, 32\}$.

TACKLE. Intention-conditioned Thompson sampling, Top-K truncation, and the reset mechanism are retained. Conflict partner counts replace agent delays in intention sampling and ranking. The selected seed guides conflict-graph neighborhood construction, followed by PP+SIPP repair. An online learning trial is successful when repair produces a complete candidate that reduces the number of colliding agent pairs. The corresponding Beta distribution receives a success or failure update. No offline training is required.

Table 6: **Key parameters and adaptation settings for the baseline methods.**

Method	Item	Settings and adaptations
NNS	Network and loss	Retain Multi-Subset and pairwise SVM. Labels are changed to differences in colliding pair counts.
NNS	Architecture	Backbone channels: 16, 16, 32, 32, 64, 64, 128, 128. Subset attention: 3 layers, 128 dimensions, 2 heads.
NNS	Candidates	$J = 100$ candidates per iteration; neighborhood size: 8.
NNS	Training	Adam; initial learning rate 10^{-4}; batch size of 16 candidate groups; cosine learning-rate decay; 10^5 updates. Models are trained per map. Checkpoints are selected by candidate selection gains on the validation set.
BALANCE	Two-level selection	Retain the official InitLNS branch. The upper level selects Target, Collision, or Random; the lower level selects the neighborhood size.
BALANCE	Prior parameters	Retain $\mu_0 = 0$, $\lambda_0 = 0.01$, $\alpha_0 = 1$, and $\beta_0 = 100$.
BALANCE	Neighborhood sizes	$\{2, 4, 8, 16, 32\}$.
TACKLE	Intention and truncation	`tackle-roulette`, $K = 32$. Conflict partner counts replace agent delays.
TACKLE	Neighborhood construction	Seed-guided InitLNS conflict-graph neighborhoods, with at most 8 agents.
TACKLE	Learning and reset	Retain the $\mathrm{Beta}(1, 1)$ prior. Reset when Top-K membership or ordering changes. Success is defined as a complete candidate that reduces the number of colliding agent pairs.

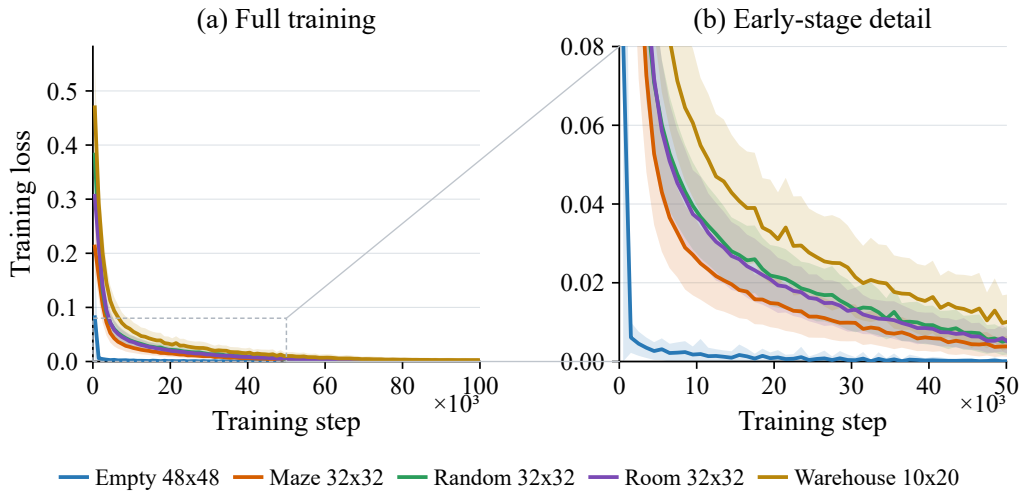


Figure 6: **NNS training loss curves for conflict elimination.** A separate model is trained for each map using the pairwise SVM ranking loss with colliding-pair differences as labels.

C.3 Results under Different Time Budgets

Table 7 reports the overall results under 100- and 200-second conflict elimination budgets. Longer budgets improve success rates and reduce remaining conflicts for all methods. The success rate of LNS2+RL increases from 55.00% to 80.00%, the highest at 200 seconds. However, its mean remaining conflict count and resolution time are 277.58 and 103.60 seconds, compared with 41.98 and 72.88 seconds for MAPF-Evo (best-ALNS). Figure 7 shows how remaining conflicts change over time on the five maps. MAPF-Evo (best-ALNS) maintains fewer remaining conflicts on Maze and eliminates conflicts quickly on Warehouse. LNS2+RL shows delayed conflict reduction on some maps. These results highlight the need to assess both the speed and extent of conflict elimination alongside success rate. Timing breakdowns are needed to determine how much computational overhead learning-based strategies introduce. Across both budgets, MAPF-Evo (best-ALNS) achieves the lowest mean remaining conflict count and the shortest mean resolution time. Its reported SOC also remains comparable to that of most baselines. These findings encourage further research on efficient conflict elimination for large-scale MAPF.

Table 7: **Overall performance under 100- and 200-second budgets.** best-only uses the best evolved heuristic alone; best-ALNS adds it to the original ALNS heuristic pool. Higher success rates and lower values of the other metrics are better. Bold indicates the best value for each metric within each budget.

Method	Success rate (%)	Conflicting pairs	Resolution time (s)	SOC
100-second budget				
MAPF-LNS2	62.50	154.85	48.44	58722.75
BALANCE	55.00	158.25	59.09	59592.58
NNS	32.50	279.50	79.83	**57841.53**
LNS2+RL	55.00	570.88	72.24	59124.33
TACKLE	47.50	164.65	62.95	58137.18
MAPF-Evo (best-only)	60.00	104.00	46.19	58442.35
MAPF-Evo (best-ALNS)	**65.00**	**85.15**	**42.80**	58603.75
200-second budget				
MAPF-LNS2	70.00	75.00	82.09	59022.98
BALANCE	57.50	100.30	102.31	59849.25
NNS	42.50	191.35	140.93	**58351.83**
LNS2+RL	**80.00**	277.58	103.60	63003.90
TACKLE	55.00	105.10	111.59	58397.90
MAPF-Evo (best-only)	65.00	65.93	85.39	58602.00
MAPF-Evo (best-ALNS)	72.50	**41.98**	**72.88**	58714.20

C.4 Generalization to Unseen Maps

To examine the generalization of the evolved best heuristic, we select three maps that are unseen during both evolution and test evaluation: `random-64-64-10`, `maze-128-128-2`, and `den312d`. For each map, we randomly select five scenarios. MAPF-LNS2 and MAPF-Evo (best-ALNS) use the same initial paths and solver protocol with a 100-second repair budget. We report the mean initial number of conflicting pairs, the number of solved scenes, the mean final number of conflicting pairs, and the mean repair time.

Both methods solve all five scenarios on `random-64-64-10` without remaining conflicts. On the more challenging `maze-128-128-2` and `den312d`, the two methods solve the same number of scenarios, while MAPF-Evo (best-ALNS) leaves fewer final conflicting pairs.The results are sum-

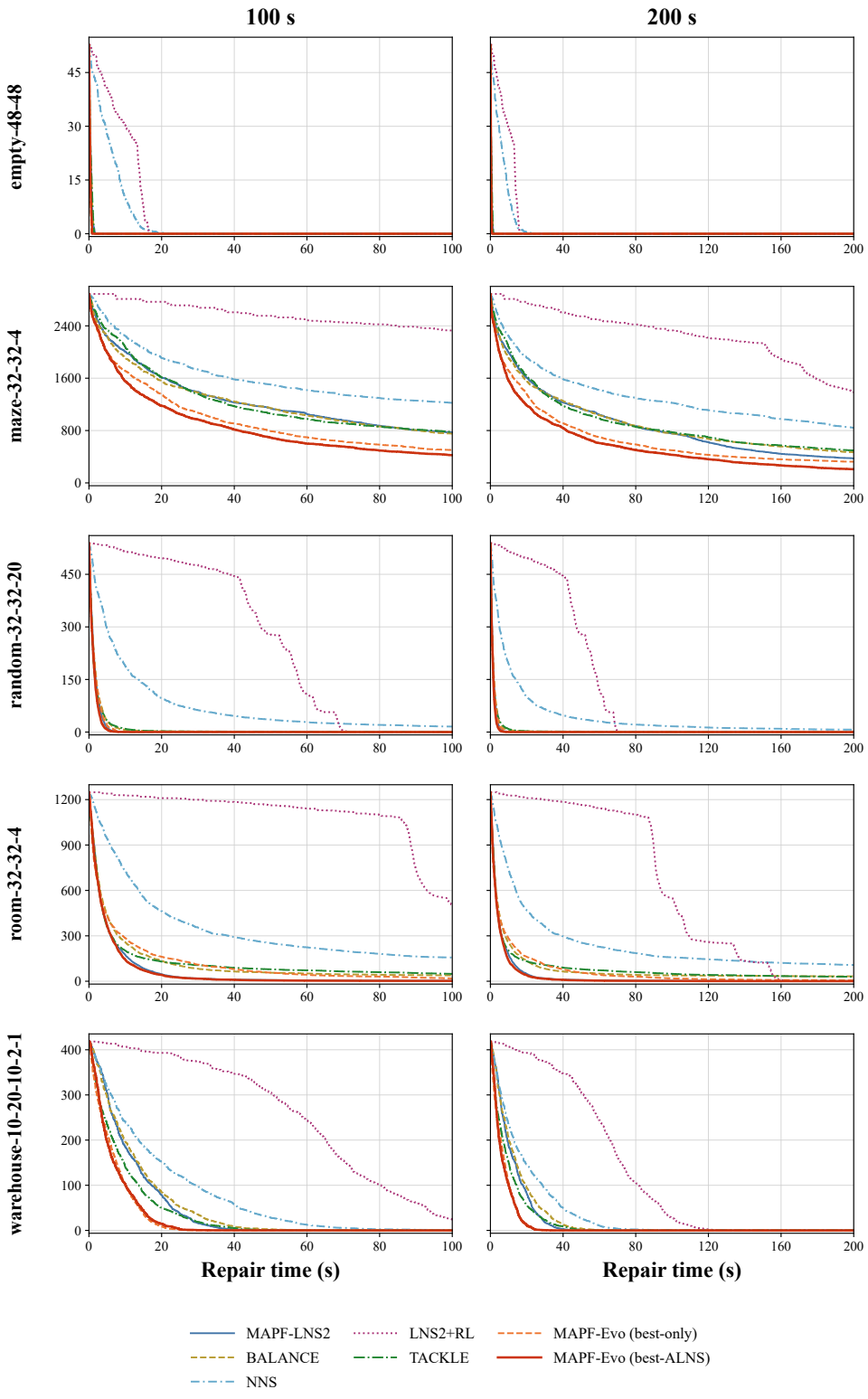


Figure 7: **Conflict elimination over time under different budgets.** The left and right columns show the 100- and 200-second budgets, respectively. Each row corresponds to one map. Curves show the mean number of remaining colliding agent pairs over repair time.

Table 8: **Generalization to unseen maps under a 100-second budget.** MAPF-Evo (best-ALNS) is compared with MAPF-LNS2 on five randomly selected scenarios from each unseen map.

Map	Method	Initial conflict pairs	Success scenes	Final conflict pairs	Repair time (s)
random-64-64-10	MAPF-LNS2	7.60	5/5	0.00	0.16
	MAPF-Evo (best-ALNS)	7.60	5/5	0.00	0.25
maze-128-128-2	MAPF-LNS2	1010.40	0/5	744.20	100.92
	MAPF-Evo (best-ALNS)	1010.40	0/5	603.20	100.52
den312d	MAPF-LNS2	3122.80	1/5	1001.00	95.04
	MAPF-Evo (best-ALNS)	3122.80	1/5	596.80	89.45

marized in Table 8. It reduces the mean final count from 744.20 to 603.20 on `maze-128-128-2` and from 1001.00 to 596.80 on `den312d`. It also slightly reduces the mean repair time on these two maps. Although it is slower on the simple random map, the overall results provide preliminary evidence that the evolved heuristic can transfer to unseen maps and retain effective conflict elimination on more complex instances. Broader evaluation across map structures and scenarios is still needed.