Scalable Lifelong Multi-Agent Planning with Multi-Tick Actions and Execution Delays

Egor Yukhnevich, Yerasyl Kenesbek, and Yersain Kenesbek
No Man's Sky Team — League of Robot Runners 2026
https://www.leagueofrobotrunners.org/

Abstract. We describe a high-throughput planning and task-allocation system for lifelong multi-agent path finding under multi-tick actions, stochastic execution delays, sparse state exchange, and strict per-tick computation limits. The system maintains a simulator-faithful world model and overlaps planning with execution through three state-freshness versus compute-window modes. Its planner, EPIBTX, extends priority inheritance and backtracking from one-step moves to short operation sequences and combines deterministic construction with parallel adaptive Large Neighborhood Search. A persistent background scheduler performs sparse top-K linear assignment, while directed guidance graphs, orientation-aware exact distance stores, dynamic congestion distances, and static priority fields provide map-specific inductive bias. The final configuration was selected through 87 competition submissions and completed 310,466 tasks with an official combined score of 10.628. The largest measured component gains were +152% from lane guidance on `orz`, +91% from lead-time planning on `fulfill-C`, and +66% from exact GoalRows heuristics on `iron`. The results support three practical conclusions: prediction fidelity is a first-class planning concern, exact map-aware heuristics can dominate generic tuning, and the computation budget must be modeled explicitly inside every search loop.

Keywords: lifelong multi-agent path finding, multi-agent pickup and delivery, priority inheritance, large neighborhood search, execution delays, task assignment

1 Introduction

Lifelong multi-agent path finding (LMAPF) couples collision-free motion with a continuing stream of goals, while multi-agent pickup and delivery (MAPD) additionally requires online task assignment [7, 10]. At warehouse scale, the planner must produce useful actions in predictable time even when agents do not execute those actions perfectly. The League of Robot Runners 2026 makes these concerns explicit: agents receive multi-waypoint tasks, primitive actions occupy multiple simulator ticks, execution can be delayed, and every live command call has a strict computation limit [3].

This combination creates a timing problem that is absent from standard one-shot MAPF. A plan computed asynchronously is evaluated on a future boundary, not on the state from which computation started. Long solve windows improve search quality but require longer prediction horizons; short horizons provide fresher state but less computation. Moreover, an agent may be partway through an uninterruptible primitive when planning begins. Task progress exposed by the external planning interface can also lag behind the physical state observed by the executor.

The resulting system cannot be separated cleanly into an offline planner and a passive executor. Prediction, assignment, search, and budget management have to be designed as one online pipeline. We therefore treat the computation schedule and the simulator semantics as part of the planning algorithm.

This paper describes the final No Man's Sky system. Its main contributions are:

1. a unified world model that replays the simulator forward, tracks task progress from physical observations, and supports three planning modes spanning the state-freshness versus computation-window trade-off;
2. EPIBTX, a short-horizon extension of Priority Inheritance with Backtracking (PIBT) for multi-tick operations, coupled with single-threaded construction and parallel adaptive Large Neighborhood Search (ALNS);
3. a persistent background scheduler using sparse top-K Jonker-Volgenant assignment over exact directed distances;
4. map-aware guidance through directed edge costs, orientation-aware exact distance stores, dynamic congestion distances, and spatial priority fields; and
5. an empirical account of 87 competition submissions, including regressions, controlled component comparisons, and the final per-map configuration.

The final system completed 310,466 tasks with an official combined score of 10.628. Its largest measured improvements were strongly map-specific: lane guidance on `orz` yielded +152%, lead-time asynchronous planning on `fulfill-C` yielded +91%, and exact GoalRows distances on `iron` yielded +66%. These results motivate a broader engineering conclusion: in large online multi-agent systems, an exact model of execution and map structure can matter more than extensive tuning of generic coefficients.

2 Problem Setting and Related Work

2.1 Online planning model

Let $G = (V, E)$ be a grid map. Because rotation consumes time, planning uses an orientation-expanded graph with nodes

$$\hat{V} = V \times \{N, E, S, W\}.$$

There are n agents. Agent i has a cell, orientation, current primitive, progress within that primitive, a staged action queue, and an optional task. A task is an ordered waypoint sequence

$$\tau = (v^{(1)}, v^{(2)}, \ldots, v^{(k)}),$$

typically beginning at pickup and ending at delivery. Tasks that have passed their first waypoint are hard-committed to their current owners.

The action alphabet is

$$\mathcal{A} = \{\text{FORWARD}, \text{ROTATE}, \text{CROTATE}, \text{WAIT}\}.$$

Each primitive lasts A simulator ticks, denoted **ActionCost**. A primitive already in progress cannot be cancelled. The simulator may delay an agent, pausing the primitive for an unknown duration drawn from a known bracket. Every tick invokes an executor, but state exchange with the external planner occurs only once per communication period, which can span several ActionCost windows.

A **boundary** is the tick at which a newly staged primitive sequence takes effect. At each boundary the system must produce collision-free staged queues before the live-call deadline. Safety requires unique physical occupancy, no opposite directed traversal, and no simultaneous multi-tick FORWARD starts whose dependency cycle would make both primitives unfinishable.

The competition objective is throughput: maximize completed tasks under fixed simulation and compute limits. The reported combined score is the official normalized competition metric; raw completed-task counts are used for component comparisons. We do not claim optimality or completeness for the full delayed, lifelong setting.

2.2 Related work

Lifelong MAPF and MAPD. MAPF conventions and benchmark variants are summarized by Stern et al. [10]. MAPD extends the setting with online pickup-and-delivery assignment [7]. Rolling-Horizon Collision Resolution decomposes lifelong warehouse planning into bounded-horizon MAPF instances and has demonstrated operation with up to 1,000 agents [4]. Our system is also horizon-based, but must additionally account for multi-tick primitive locks, sparse external state exchange, and planning that overlaps execution.

Priority inheritance. PIBT assigns dynamic priorities and resolves local conflicts by priority inheritance with recursive backtracking [8]. It is attractive in large online settings because it produces actions quickly and predictably. Earlier No Man's Sky work extended PIBT with multi-action operations and combined it with graph guidance and LNS [11]. EPIBTX retains this push-chain structure but adds multi-tick edge reservations, previous-boundary warm starts, delayed-execution handling, weighted progress, and a deadline-aware parallel improvement phase.

Large Neighborhood Search. LNS-based MAPF methods repeatedly destroy and repair subsets of a solution. MAPF-LNS2 uses fast local repair to remove collisions [5], while BALANCE adapts destroy choices and neighborhood sizes online [9]. Our ALNS phase begins from a feasible EPIBTX solution, adapts destroy-operator selection by committed improvement per unit time, and parallelizes repair through optimistic overlays on one shared reservation state.

Guidance and congestion. Guidance graphs encode directed traffic preferences as edge weights and can raise lifelong throughput across several planners [13]. We use the same general abstraction but combine programmatic, manually marked, and heatmap-derived guidance. A separate dynamic heuristic recomputes goal distances under current congestion, while a lighter presence penalty modifies operation scoring without rebuilding distances.

Execution delays. Delay-probability models and robust execution policies were introduced for MAPF by Ma et al. [6]. Other work repairs plans by introducing additional delays [2], while concurrent planning-and-execution frameworks explicitly combine replanning with delayed execution [12]. Our approach is complementary: rather than provide a robustness guarantee, it uses simulator-faithful prediction and map-specific delay strategies to maximize throughput under a hard online budget.

3 Asynchronous System Architecture

Figure 1 shows the three long-lived roles. The executor runs on every live simulator tick and performs only state capture, worker launch, tick pacing, result harvest, safety validation, and commit. A persistent scheduler continuously solves task assignment on coalesced snapshots. One planner worker owns the candidate solution for a boundary. Component-specific pools for world prediction, heuristic row construction, and LNS repair are children of the planner worker, so parallel resources are activated only during the relevant phase.

3.1 Unified world model

`World` is the only state type consumed by planner and scheduler. It contains the snapshot tick, the delay bracket, all agents, all assigned and free tasks, and task ownership. Each agent stores both physical state and solver state: location, orientation, primitive phase, delay status, staged actions, current goal sequence, priority, and task-commit status.

The external planning snapshot can be stale between communication periods. A lightweight `WorldTracker` therefore observes physical positions every live tick and detects waypoint crossings, pickup commitment, and task completion. This fresh state is used to prevent planning toward an already crossed waypoint and to keep committed tasks pinned.

3.2 Simulator-faithful prediction

For an asynchronous job launched h ticks before its target boundary, `World::predict(h)` replays the simulator tick by tick. A prediction step:

1. continues a mid-primitive action and pauses it if the same ActionModel oracle used by the simulator rejects progress;
2. applies the head-conflict guard;
3. applies the selected delay policy;
4. executes the front staged action for agents not already mid-primitive, including stale-action flushing; and
5. updates delay age, waypoint progress, hard commitment, and completion, then validates the complete world.

Validation checks unique cell occupancy, fractional box intersections during FORWARD, head conflicts, and symmetric agent-task ownership. Any predictor divergence becomes a reproducible failure rather than a silent planning error.

Three delay policies determine how a delayed agent is projected. SMART estimates the remaining delay from its bracket and observed age. PESSIMISTIC holds the agent for the complete forecast horizon. OPTIMISTIC uses a minimal remainder and was not selected in the final system. PESSIMISTIC is used on `fulfill-A/B/C` and `rand-A`, where SMART produced phantom wake-ups; SMART is retained on `iron`, `orz`, and `bos`.

3.3 Planning modes

Figure 2 summarizes the planning modes.

- **SYNC_STEP** solves on the boundary tick from actual state. It eliminates prediction error but receives only one live-call budget. It is selected on `maze-A/B` and

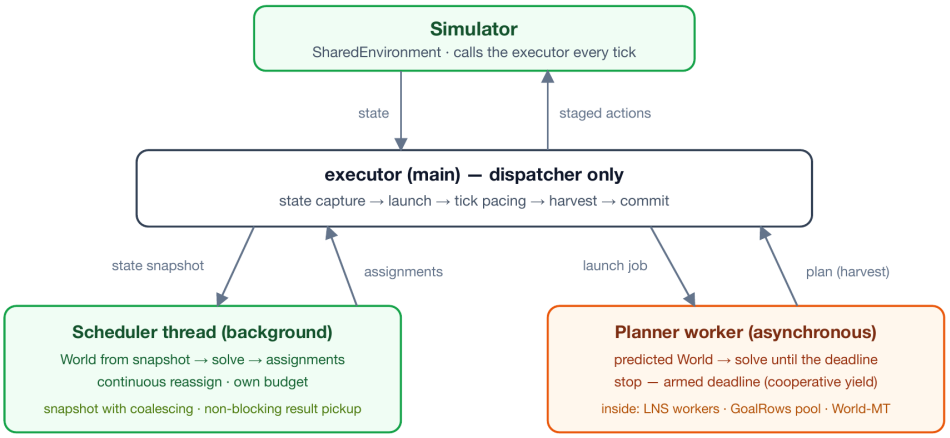


Figure 1: Top-level execution architecture. The main thread dispatches work; scheduler and planner operate on independent world snapshots.

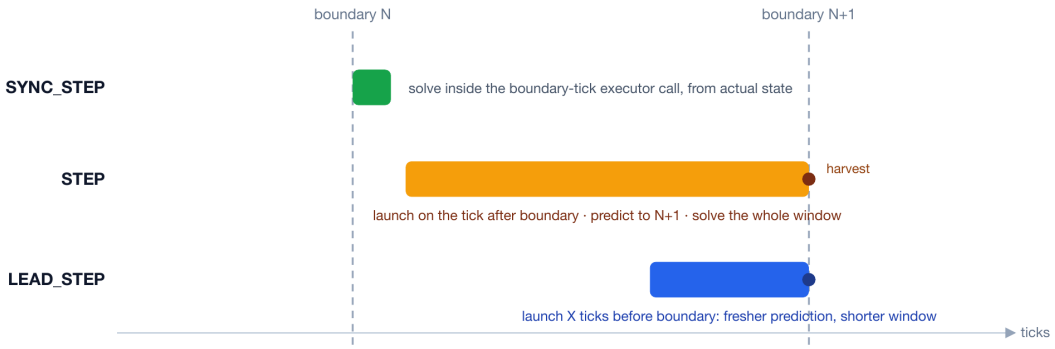


Figure 2: Planning modes trade prediction horizon for available search time.

`room-A/B`.

- **STEP** launches immediately after the previous boundary, predicts one ActionCost window, and overlaps planning with execution. It gives the solver the longest window and is used where compute dominates prediction error.
- **LEAD_STEP** launches X ticks before the boundary. It was introduced for `fulfill-C`, where STEP predicted too far ahead under harsh delays but SYNC did not leave enough time for LNS. The final setting is $X = 4$.

3.4 Safety and budget invariants

The simulator can enter a permanent dependency cycle if two agents start FORWARD into the same destination or into mutually occupied targets. The executor therefore replaces the unsafe second start with WAIT. The same guard is applied during prediction, so the state used by asynchronous planning obeys the execution invariant.

At a boundary, the executor arms a cooperative deadline:

$$t_{\text{deadline}} = t_{\text{start}} + L_{\text{move}} - m_{\text{safety}} - r_{\text{handover}}.$$

Every construct, retry, LNS, and heuristic loop checks this deadline and returns its best incumbent. The executor waits only until $L_{\text{move}} - m_{\text{safety}}$; the handover reserve covers validation and staged-queue commit. Between boundaries the deadline is disarmed, allowing search to consume the overlapping execution window.

4 EPIBTX and Parallel ALNS

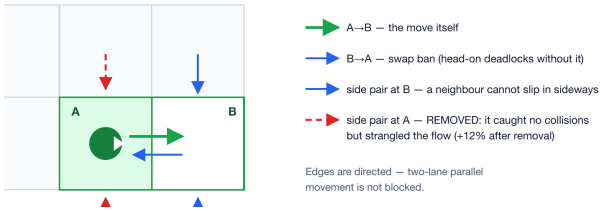


Figure 3: Directed resources reserved by a FORWARD operation.

4.1 Short operation sequences

EPIBTX plans an **operation** for each agent: a sequence of d primitives from $\mathcal{A}$, with $d = 3$ by default and $d = 4$ on `rand`, `simple`, `tunnel`, and `room`. The fixed operation pool contains sequences such as `FFF`, `RFF`, and `CWF`; operation zero is all-WAIT.

For every agent r and operation o, initialization computes the orientation-aware path at every depth, filters operations that violate primitive locks, and sorts the remaining candidates by `smart_dist`. A mid-primitive agent is forced to finish that primitive. A delayed agent begins with WAIT. An agent delayed mid-FORWARD reserves the unfinished transition at every depth so that neighbors cannot enter its swept region.

4.2 Multi-tick reservations

Two single-owner tables represent the current solution:

$$R_V[v,k] \quad \text{and} \quad R_E[e,k],$$

for vertex and directed-edge occupancy at primitive depth k. Adding or removing an operation updates every reserved slot and the objective telescopically, which makes recursive rollback exact.

Figure 3 shows the FORWARD reservation model. A move $A \to B$ reserves the destination, the forward edge, the opposite $B \to A$ edge, and side-entry edges at B. The side pair at source A was removed after it blocked parallel flow without preventing observed collisions; this change improved the maze/room class by 12%. Rotation and WAIT reserve only their vertex.

4.3 Operation objective

The base score of operation o is the directed heuristic distance from its endpoint to the active goal. Where enabled, dynamic congestion distance replaces the base heuristic. Early goal arrival sets the distance to the reached depth, and an agent already on its goal is scored toward the next waypoint. On `rand-A`, a $\log(1+x)$ transform compresses far-field distance differences.

Let $d_r(o)$ be this distance after map-specific transforms, $b_r(o)$ the prefix match with the shifted previous-boundary plan, and $a_r(o)$ an indicator that the operation actually reaches its goal. The effective score is

$$s_r(o) = d_r(o) - 24\, b_r(o) - 0.5\, a_r(o),$$

with arrive bonus disabled on `rand-A` and `orz`. EPIBTX maximizes weighted progress:

$$J(\mathbf{o}) = \sum_r w_r \left[s_r(\mathrm{WAIT}^d) - s_r(o_r) \right].$$

Priority rank determines w_r; PriorityField further multiplies it at marked intersections. `rand-A` instead uses uniform weight to prevent a sacrificial-lock pattern in which low-priority agents repeatedly remain stationary.

The prefix term provides **plan inertia**. The first warm-start pass favors the validated suffix of the preceding plan. The objective is then recomputed without inheritance bias before later passes. This retains temporal consistency without permanently changing the optimization target.

4.4 Priority inheritance and backtracking

Agents are processed in priority order. Algorithm 1 summarizes the recursive build. An operation with no blocker is accepted. With one blocker, the blocker is evicted and recursively repaired, provided it is weaker than the initiating agent and has not already entered the current chain. Failure restores the prior reservation state and tries the next operation. Operations with multiple blockers are skipped in the flagship configuration.

One solve performs an inheritance-biased pass, removes the bias, and runs up to four full passes while the objective improves. Agents left on all-WAIT receive up to ten tail-first retry passes. If progress stalls, force-top-inheritance temporarily lets a stuck agent push any blocker. Every recursion and retry is deadline-aware.

4.5 Parallel adaptive LNS

Figure 4 shows the two-phase flagship planner. Single-threaded `solve_multipass` constructs a high-quality feasible solution. A K-worker ALNS phase then improves it until the deadline. Parallel construction was tested but underperformed: moving all workers to LNS improved `fulfill-A` by 8.2%.

Algorithm 1 Recursive EPIBTX build

```
function BUILD(r, ρ)
    remove the current operation of r
    for all admissible o in increasing s_r(o) do
        B ← owners of resources required by o
        if B = ∅ then
            place o; return success
        else if B = {q} and q is weaker than ρ then
            remove q; place o
            if BUILD(q, ρ) then return success
            end if
            rollback o and restore q
        end if
    end for
    place all-WAIT for r; return failure
end function
```

Each LNS worker explores a repair in a thread-local `ChainOverlay`. On first access to a reservation slot, the overlay records the shared base value; subsequent writes remain local. Agents are claimed with non-blocking compare-and-swap, so workers never wait on one another. A repaired chain is admissible only if $\Delta J \geq 0$. Commit takes one mutex and validates that every recorded base still matches shared state. Any mismatch discards the complete chain as stale; otherwise all slot and score changes are published atomically.

Five destroy operators target different failure modes: random replacement, the largest weighted gap, a stuck WAIT agent with blockers of its best operations, a congestion hotspot, and a spatial cluster. Operator selection uses committed improvement divided by elapsed time, with warm-up and epsilon-greedy exploration. Multi-agent operators try several reinsertion orders and reject any variant that introduces a new WAIT-first agent. Adaptive selection added approximately 1.5% over flat LNS on `maze`, `room`, and `rand`.

5 Assignment and Map-Aware Guidance

5.1 Persistent background assignment

For free agent r and task τ, assignment cost is

$$c(r,\tau) = 5\, H(x_r, \mathrm{pickup}(\tau)) + H_{\mathrm{route}}(\tau),$$

where H is the exact directed, orientation-aware distance and H_{route} is the remaining task-route length.

Each free agent retains its nearest $K = 100$ tasks. When fewer than 2,000 agents are free, the scheduler runs a sparse Jonker-Volgenant assignment [1] over the union of these task columns, then fills unmatched pairs greedily. At larger scale it uses a greedy heap with a final fallback. Replacing pure greedy matching improved total throughput by 11%.

The scheduler owns a persistent thread and processes the newest coalesced `World` snapshot whenever free. Results are harvested non-blockingly. This separation is essential for reassignment: before an agent reaches or approaches the first waypoint, a task may move to a better owner. Tasks are pinned within Manhattan distance 6 of pickup; only a capped set of the worst assignments is reconsidered; and warehouse maps require at least a 5% distance improvement. Background

Figure 4: Single-threaded construction followed by optimistic parallel ALNS repair.

scheduling plus reassignment improved `fulfill-A` by 63%, `fulfill-B` by 28%, and the maze/room/rand group by 17%.

A persistent cache stores sorted agent-to-task distance rows across ticks. Rebuild work starts from the oldest rows and is capped, preventing the cache refresh itself from consuming the complete scheduling budget.

5.2 Guidance Graph

The Guidance Graph (GG) assigns an action cost to each pair of oriented cell and primitive. These directed weights enter the shortest-path relaxation, so lane structure affects every downstream heuristic before conflict resolution. An edge may also be hard-blocked.

Different map families use different sources of guidance. Warehouse maps use programmatic alternating one-way corridors; `orz` uses manually marked two-lane ribbons and blocked danger zones; `bos` uses lanes derived from replay heatmaps; `maze` is generated programmatically; and `iron` is almost neutral except at its narrow bottleneck. The lane graph on `orz` was the largest individual improvement in the system, raising completed tasks by 152%.

5.3 Orientation-aware exact distances

HeuristicMatrix exposes $H(x, g)$, the shortest cost from oriented state $x \in \hat{V}$ to goal cell g, including rotation and GG weights. Most maps use a full backward-Dijkstra matrix. Two memory reductions are available:

- **Half-node storage** retains only states whose cell has even grid parity. Since FORWARD flips parity, missing values are reconstructed exactly through an adjacent stored state.
- **GoalRows** stores only active-goal rows under a cap and builds them in a background pool. Rows are atomically published and retired only after a boundary grace period. Until publication, the query falls back to Manhattan distance.

The full matrix did not fit on `iron`; replacing a stitched pattern-database estimate with exact GoalRows raised throughput from 79.9k to 132.4k tasks, a 66% gain. After the evaluation memory limit increased, the final submission used full stores where possible while retaining both compact modes.

5.4 Dynamic congestion and spatial priority

DynamicHeuristicMatrix (DHM) periodically rebuilds a goal row with an added entry cost:

$$c_{\mathrm{DHM}}(u, v) = c_{\mathrm{GG}}(u, v) + \lambda \ \mathrm{stallHeat}(v).$$

This makes a route around a queue cheaper in the dynamic metric. Missing rows fall back to the base matrix. DHM is enabled only where alternate routes exist: `fulfill-A/B/C`, `rand-A`, and `bos`. A lighter TrafficFlow term adds exponentially decayed current presence directly to operation scoring on `maze-A` and `bos`.

PriorityField addresses a different failure mode. On warehouse intersections, a following convoy could push a turning agent away from its task. Marked intersection and nearby T-junction cells multiply the agent's construction weight by 100, so the turn is processed earlier and survives push conflicts. The effect was +7.7% with the flagship planner on `fulfill-A`, and larger with weaker planners.

6 Experimental Methodology

6.1 Evaluation domains

The official evaluation contains nine named map families with distinct topology, delay behavior, and scale: `fulfill-A/B/C`, `iron`, `orz`, `bos`, `rand-A`, `maze-A/B`, and `room-A/B`. The same global code path is used on every map, but configuration selects the planning mode, delay predictor, heuristic backend, and guidance features shown in Table 1.

6.2 Metrics and evidence sources

The primary metric is the number of tasks completed by the end of a run. The competition also reports a normalized per-map score and a combined score relative to other submissions. The final result is reported with both raw total tasks and the official combined score.

Evidence comes from three sources:

1. **controlled component comparisons** on a fixed development or evaluation configuration, reported as the relative change in completed tasks;
2. **official submissions**, which capture the complete integrated system under the competition server; and
3. **replay diagnostics**, including action shares, delay and stuck counts, prediction misses, assignment queues, planner time, and per-cell visit/wait heatmaps.

The 87 submissions constitute a sequential system-development trace rather than a full factorial benchmark. Approximately one third were regressions or isolation runs. Component percentages are therefore local effects under the stated map and configuration, are not additive, and should not be interpreted as confidence intervals. The official normalized score also changes when competing teams improve; task counts and same-configuration A/B comparisons are more stable for attribution.

Map	Planning mode	Delay model	Distance backend	Selected map-specific features
`fulfill-A`	STEP	PESSIMISTIC	Precomputed + DHM $\lambda = 6$	v2 guidance, PriorityField
`fulfill-B`	STEP	PESSIMISTIC	Precomputed + DHM $\lambda = 6$	v9 guidance, PriorityField
`fulfill-C`	LEAD, $X = 4$	PESSIMISTIC	Precomputed + DHM $\lambda = 6$	v2 guidance, PriorityField
`iron`	STEP	SMART	GoalRows + builder pool	near-neutral guidance; DHM disabled
`orz`	STEP	SMART	Precomputed	lane guidance; arrive bonus disabled
`bos`	STEP	SMART	Precomputed + DHM $\lambda = 20$	manual guidance; presence $\lambda = 1$
`rand-A`	STEP, $d = 4$	PESSIMISTIC	Precomputed + DHM $\lambda = 6$	uniform weights; heat priority; $\log(1 + x)$ compression
`maze-A/B`	SYNC	actual state	Precomputed	programmatic guidance; presence on A
`room-A/B`	SYNC, $d = 4$	actual state	Precomputed	file-based guidance

Table 1: Final per-map configuration.

6.3 Selection procedure

Development followed a one-change-at-a-time policy whenever possible. Candidate modifications were first tested on the affected map family, then submitted in isolation. Large regressions were rolled back to the last trusted base, and only validated changes were ported into the final branch. The per-map registry prevents a gain on one topology from becoming an unverified global default.

The most important counterexample occurred when SMART prediction was enabled globally. `rand-A` lost 75% because the asynchronous worker effectively froze. Adding the cooperative deadline fixed the freeze, but also shortened the LEAD window on `fulfill-C`. The final system therefore restores PESSIMISTIC prediction on fulfill/rand and treats delay policy as part of map configuration.

7 Results

7.1 Final performance

The final submission completed **310,466 tasks** and achieved an official combined score of **10.628**. Figure 5 compares its per-map score with the two nearest competitors. Performance is not concentrated in one domain: the system remains competitive across warehouse, open random, corridor, room, and large-memory maps, although the mechanism responsible for each result differs.

7.2 Measured component effects

Table 2 lists the largest attributed gains. The top three are all map-specific: directed lane structure, the planning launch horizon, and exact distance representation. General mechanisms such as background assignment and plan inheritance remain important, but their effect also depends strongly on topology.

7.3 Search trajectory

Figure 6 shows five phases.

Core search, submissions 1–16. The planner family and scheduling family were explored separately. Top-K Hungarian assignment emerged as the scheduler base.

Large maps, 17–30. `iron` and `orz` were enabled; SYNC planning and the first pattern-database distances appeared.

Infrastructure, 31–52. STEP EPIBTX, the background scheduler, reassignment, DHM, plan inheritance, the F-edge model, and per-map configuration established the final architecture.

Flagship leap, 54–68. Parallel ALNS and PriorityField became defaults. Manual lane guidance raised `orz` from 8.3k to 20.9k tasks. Exact GoalRows raised `iron` from 79.9k to 132.4k.

Isolation and final assembly, 69–87. A global SYNC experiment exposed the value of fresher state on `fulfill-C`, leading to LEAD. Three bundled regressions motivated stricter isolation. Deadline-aware solving repaired a catastrophic asynchronous freeze. The final branch started from the last trusted base and ported only LEAD, memory-safe distance stores, and background GoalRows construction.

7.4 Negative results

Several rejected ideas were informative. Parallelizing initial construction reduced solution quality relative to one-thread multipass construction. A construction-time no-worsening gate nearly eliminated corridor throughput because temporary sacrifices were required to form push chains. Global SMART prediction failed on `rand-A`. A component-based scheduler improved development maps but did not transfer to official evaluation. A lazy operations cache caused a decisive memory failure on `iron`. These cases reinforce the need to evaluate algorithmic quality, memory, and deadline behavior together.

8 Discussion and Limitations

8.1 What drove throughput

Exact map structure dominated generic tuning. The largest gains were not coefficient sweeps. They came from encoding directed traffic on `orz`, choosing the correct launch horizon on `fulfill-C`, and representing long-range distance exactly on `iron`. The local planner was most effective when its heuristic already expressed the global topology.

Prediction horizon is an algorithmic parameter. Asynchronous planning is not unconditionally better than synchronous planning. STEP wins when extra search time outweighs prediction error; SYNC wins when local solving is already fast; LEAD occupies the middle regime. This suggests that adaptive launch timing, based on observed delay rate and recent solve improvement, is a promising extension.

Temporal consistency is valuable. Operation inheritance was a large warehouse gain at negligible runtime. The previous plan is not simply reused: it biases the first pass, then the unbiased objective resumes from a temporally coherent incumbent. This is cheaper than repeatedly rediscovering the same coordinated flow.

Budget behavior belongs inside the solver. Stopping only at the executor boundary is too late. Cooperative checks in recursive build, retries, ALNS, and heuristic construction are what prevent a high-quality plan from causing an empty execution tick.

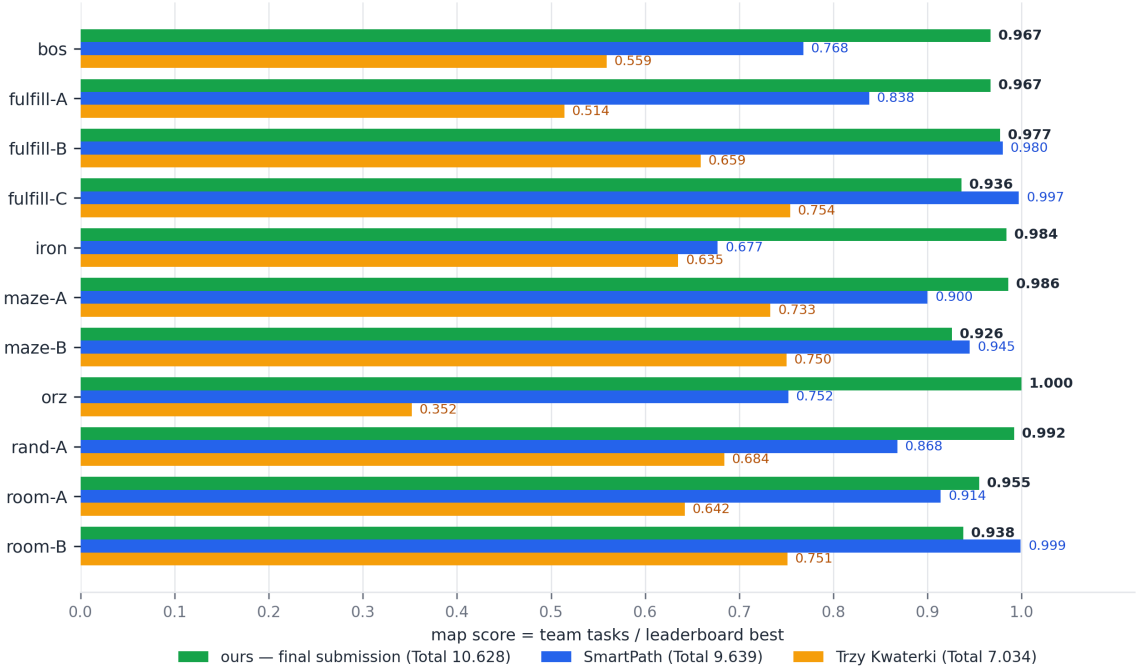


Figure 5: Final per-map score relative to the two closest competitors in the official evaluation.

8.2 Limitations

First, the evidence is a competition development trace, not a randomized benchmark study. Most reported gains are single-configuration A/B effects, and interactions between components are substantial. Repeated seeds, variance estimates, and an independent test set would be required for stronger causal claims.

Second, the per-map registry deliberately uses domain knowledge. Manual guidance on `orz` and heatmap-guided markup on `bos` do not transfer automatically to unseen layouts. Guidance Graph Optimization and online guidance methods offer a route toward automation [13], but were outside this implementation.

Third, EPIBTX is an incomplete, bounded-horizon planner for the full delayed MAPD problem. The safety guard prevents observed permanent primitive cycles, but the system offers no formal liveness or optimality guarantee under arbitrary delays.

Fourth, exact distance storage trades time for memory. GoalRows and half-node parity compression reduce the footprint, but configuration still depends on the evaluation memory limit and active-goal distribution.

Finally, the reported official score is relative to the competition field and may change as other submissions improve. The raw final task count and fixed-configuration comparisons are therefore the more reproducible quantities. A release intended for archival reproducibility should include source code, exact competition kit revision, configuration files, and replay logs for the cited runs.

9 Conclusion

We presented an online planning system for lifelong multi-agent pickup and delivery with multi-tick actions, stochastic delays, sparse state exchange, and hard computation limits. The system combines simulator-faithful prediction, asynchronous launch modes, short-horizon priority-inheritance planning, parallel adaptive LNS, background task assignment, and map-aware exact and congestion-sensitive heuristics.

The final competition result was 310,466 completed tasks and a combined score of 10.628. More importantly, the development trace identifies the mechanisms behind that result. High throughput required an honest model of when a plan becomes active, exact representation of map-specific distance and traffic structure, temporal consistency across boundaries, and cooperative budget control throughout the solver. Future work should replace the fixed per-map registry with online selection of prediction horizon, delay policy, and guidance while retaining the same explicit safety and deadline invariants.

Component change	Measured on	Gain	Interpretation
manual lane Guidance Graph	`orz`	+152%	separated opposing flows before local conflict resolution
LEAD_STEP, $X = 4$	`fulfill-C`	+91%	balanced prediction freshness against time available for LNS
exact GoalRows-HM	`iron`	+66%	removed systematic errors from stitched pattern-database estimates
BACKGROUND scheduler + reassignment	`fulfill-A`	+63%	removed assignment from the critical path and corrected stale ownership
BACKGROUND scheduler + reassignment	`fulfill-B`	+28%	same mechanism under a different warehouse flow pattern
DynamicHeuristicMatrix	`fulfill-B`	+24%	routed traffic around queues where alternate paths exist
$\log(1 + x)$ distance compression	`rand-A`	+18%	reduced over-prioritization of far-field distance differences
operation inheritance + v2 guidance	`fulfill-A`	+15.2%	stabilized consecutive plans and reduced oscillation
F-edge reservation revision	maze/room	+12%	restored parallel flow while retaining destination-side safety
top-K assignment vs. greedy	aggregate	+11%	minimized fleet approach distance rather than local first-come choices

Table 2: Largest measured component effects. Percentages are local comparisons and are not additive.

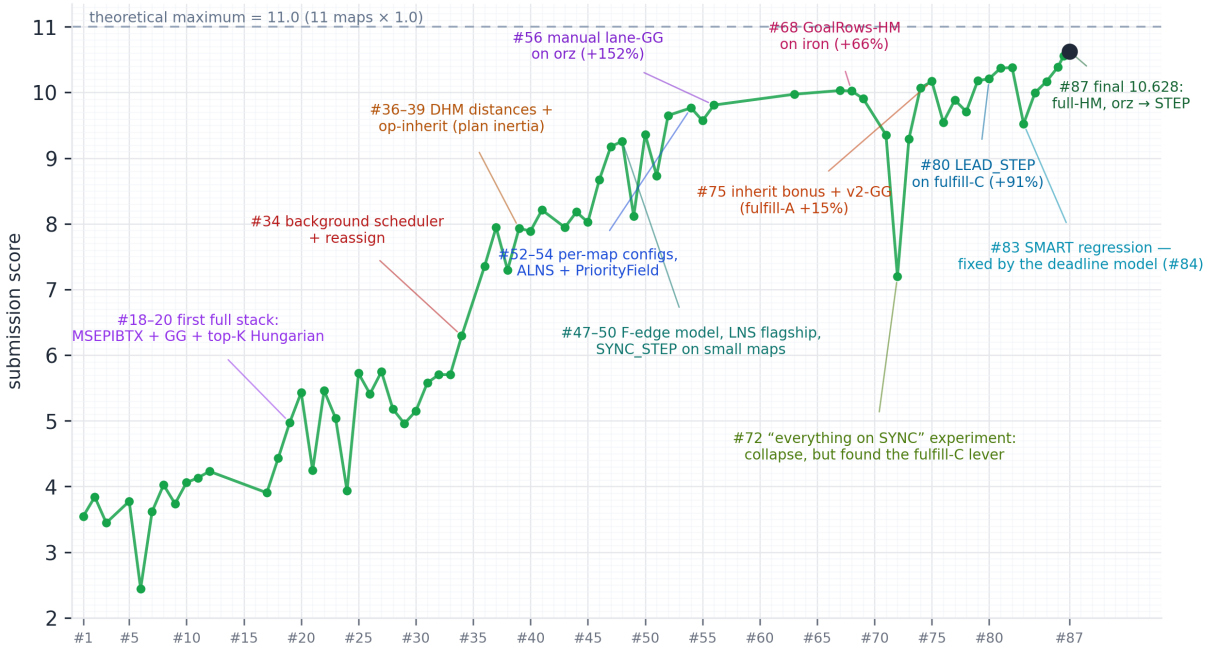


Figure 6: Combined score across the submission history. Values before submission 46 were recomputed later; subsequent values are the score visible at submission time.

A Additional Implementation Detail

A.1 World state

Table 3 lists the state required to reproduce simulator semantics and expose a single interface to planner and scheduler.

Population and prediction loops use a dedicated worker pool only on the launch path, when the planner worker itself is idle. Outside launch, the same functions execute serially. This avoids nested parallelism during search.

A.2 ALNS operators

Joint neighborhoods use multi-order repair. The first order follows regret, inserting the agent with the highest loss if deprived of its best operation. Later orders are random. The best total ΔJ is replayed into the overlay and committed optimistically. A cascade queue places agents evicted by a just-committed chain at the front of the same worker's next attempt.

A.3 Distance backends

The half-node representation is exact on the bipartite grid because every FORWARD changes cell parity. Debug builds compare compact queries against a full store. GoalRows uses a publication grace period of at least one boundary before old rows are reclaimed, preventing readers in the current solve window from observing freed memory.

B Markup and Replay Diagnostics

The planning system is paired with two browser tools. The markup editor overlays the live map and writes production GG, PriorityField, and component files. Direction painting assigns E/S/W/N, neutral, and danger states; the same converter used in production derives along-flow, cross-flow, turn, and blocked-edge costs. A live distance heatmap rebuilds after each edit.

Every run writes a single replay containing initial state, actual per-tick actions, planned operations, task events, delay intervals, and subsystem metrics. The viewer reconstructs all agent positions and provides playback, single-step control, plan overlays, task goals, trajectories, density, wait, stuck, and delay layers. Aggregate visit and wait heatmaps were used to derive manual guidance on `bos`.

The same replay stores run-level aggregates: fleet action shares, EPIBTX operation acceptance, LNS commit quality, ALNS operator yield, scheduler reassignment, prediction misses, and executor timing. These diagnostics were essential for distinguishing algorithmic stalls from deadline, memory, and prediction failures.

Field group	Contents
physical agent	cell, orientation, active primitive, progress counter, delay flag and age, staged queue
planner agent	orientation-expanded node, current and remaining goals, priority, primitive lock, predicted WAIT prefix
task	waypoint sequence, crossed index, hard-commit flag, completion flag, owner
world	snapshot tick, delay bracket, prediction policy, agents, assigned tasks, free task pool

Table 3: Core `World` state.

Operator	Destroyed neighborhood
`replace`	one random agent not using its best operation
`gap-target`	agent with the largest weighted difference between current and best operation
`stuck-chain`	a WAIT anchor and owners of resources required by its top three operations
`hotspot`	agents near a high congestion-EMA cell; falls back to a window around a WAIT agent
`cluster`	owners of reserved vertices in a Chebyshev window around a gap anchor

Table 4: ALNS destroy operators.

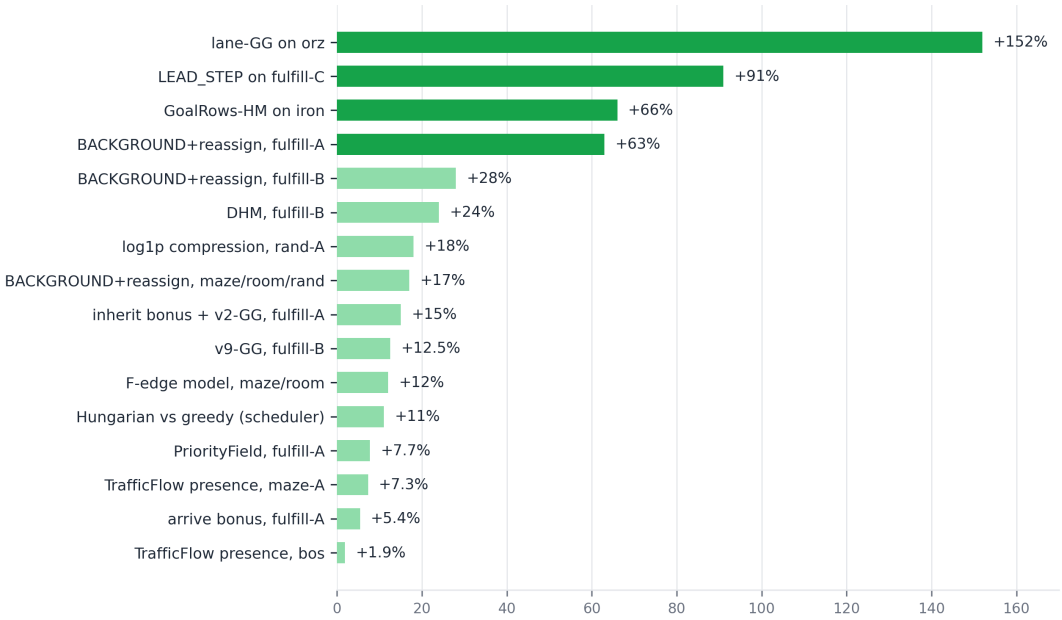
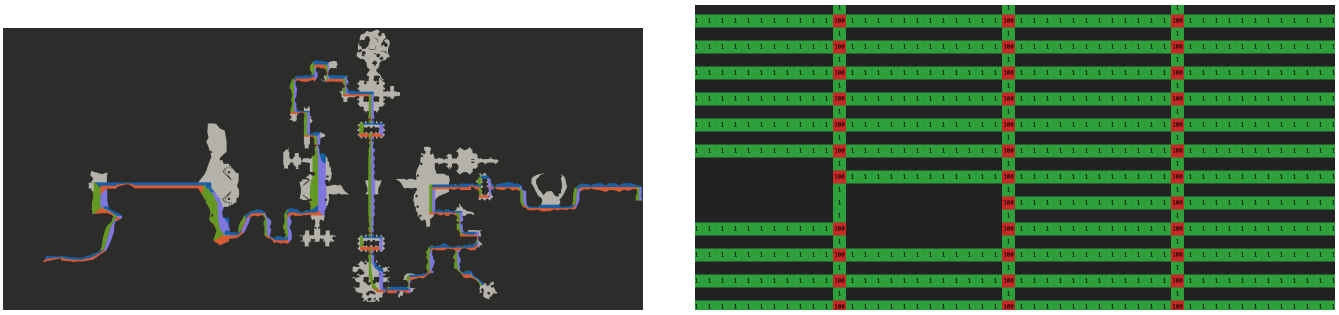


Figure 7: Largest measured component effects across the development trace. Values are local comparisons under different maps and configurations.



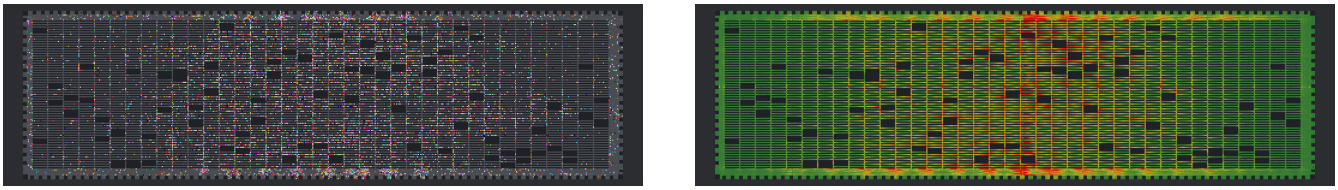
(a) Directional lane markup on `orz`.

(b) Warehouse PriorityField: corridor weight 1 and intersection weight 100.

Figure 8: Production map annotations used by the planner.

Backend	Maps	Mechanics
Precomputed	most maps	full orientation-node to goal matrix from backward Dijkstra
half nodes	`orz` option	store one cell parity and reconstruct the other exactly
GoalRows	`iron`	lazy active-goal rows, capped and evicted; background build and atomic publication
Block/Cluster PDB	legacy `iron`	stitched lower bound; compact but systematically underestimated long detours

Table 5: HeuristicMatrix storage backends.



(a) Full `fulfill-A` replay with agent distribution.

(b) Aggregate wait heatmap used for congestion diagnosis.

Figure 9: Spatial replay diagnostics.

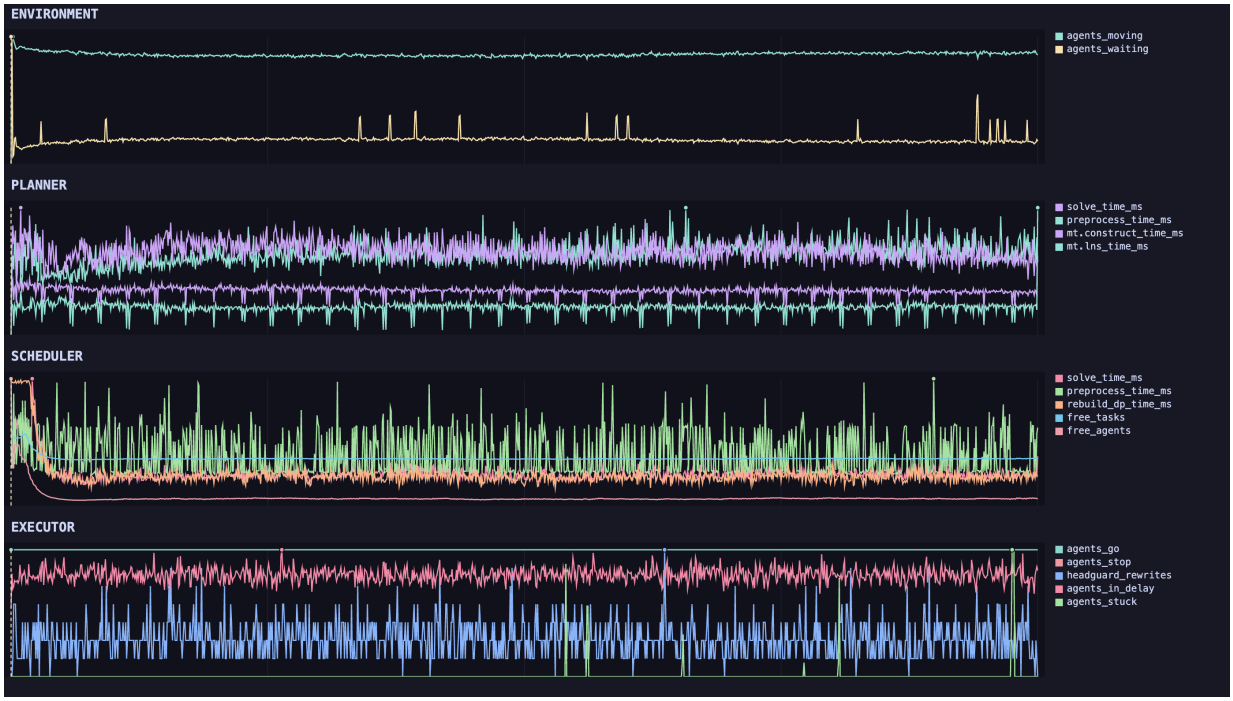


Figure 10: Synchronized per-tick environment, planner, scheduler, and executor series. Curves are normalized to their own maxima in the viewer.

References

[1] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4): 325–340, 1987. doi: 10.1007/BF02278710.

[2] Justin Kottinger, Tzvika Geft, Shaull Almagor, Oren Salzman, and Morteza Lahijanian. Introducing delays in multi-agent path finding. In *Proceedings of the 17th International Symposium on Combinatorial Search*, 2024. URL https://arxiv.org/abs/2307.11252.

[3] League of Robot Runners. League of robot runners 2026: Problem and evaluation. https://www.leagueofrobotrunners.org/problem, 2026. Accessed 2026-07-24.

[4] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11272–11281, 2021. doi: 10.1609/aaai.v35i13.17344.

[5] 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, pages 10256–10265, 2022. doi: 10.1609/aaai.v36i9.21266.

[6] Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017. URL https://arxiv.org/abs/1612.05309.

[7] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pages 837–845, 2017. URL https://arxiv.org/abs/1705.10868.

[8] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 535–542, 2019. doi: 10.24963/ijcai.2019/76.

[9] 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, pages 17514–17522, 2024. URL https://arxiv.org/abs/2312.16767.

[10] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Barták. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th Annual Symposium on Combinatorial Search*, pages 151–159, 2019. URL https://arxiv.org/abs/1906.08291.

[11] Egor Yukhnevich and Anton Andreychuk. Enhancing PIBT via multi-action operations. *arXiv preprint arXiv:2511.09193*, 2025. doi: 10.48550/arXiv.2511.09193. URL https://arxiv.org/abs/2511.09193.

[12] Yue Zhang, Zhe Chen, Daniel Harabor, Pierre Le Bodic, and Peter J. Stuckey. Concurrent planning and execution in lifelong multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23387–23394, 2025. doi: 10.1609/aaai.v39i22.34506.

[13] Yulun Zhang, He Jiang, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. Guidance graph optimization for lifelong multi-agent path finding. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*, pages 311–320, 2024. doi: 10.24963/ijcai.2024/35.