Flatland Challenge

Claire Sun*

ABSTRACT

This report documents my solutions to the three stages of the Flatland Challenge, a multi-agent pathfinding benchmark that progressively increases in complexity. Question 1 addressed single-agent pathfinding without collisions or time constraints. Using A* with a Manhattan heuristic, I achieved optimal paths and a perfect score, establishing a baseline. Question 2 introduced temporal dynamics and sequential planning with dynamic obstacles. I first implemented a naïve time-aware A*, then refined it into a Safe Interval Path Planning (SIPP)-style algorithm, which scaled efficiently and achieved a F-score above 100%. Question 3 extended the challenge to full-scale multi-agent scheduling with deadlines, malfunctions, and replanning. Across four iterative approaches, I combined SIPP-heading, Prioritised Planning, Large Neighbourhood Search, and targeted conflict prevention, culminating in a memory-optimised framework that achieved a 58% F-score. The results highlight the interplay of algorithmic design and systems engineering in achieving scalable, real-world multi-agent planning.

Keywords: Multi-Agent Pathfinding, A*, Safe Interval Path Planning (SIPP), Prioritised Planning, Large Neighbourhood Search (LNS), Deadlock Prevention, Flatland Challenge

I. INTRODUCTION

The Flatland Challenge provides a progressively more complex sequence of multi-agent pathfinding tasks, designed to test planning algorithms under increasing levels of realism and difficulty. Each stage builds upon the previous one, moving from basic single-agent routing to large-scale, multi-agent coordination with dynamic disruptions.

The first task (Q1) serves as a warm-up exercise, focusing on single-agent pathfinding. Each instance involves a single train agent with a fixed start and target position, without collisions or temporal constraints. This reduces the problem to a static shortest-path search, where the primary objective is to generate a valid and cost-efficient route while minimising the Sum of Individual Costs (SIC). The task provides an opportunity to establish a strong baseline and explore the behaviour of classical search algorithms such as A*.

The second task (Q2) significantly increases the complexity by introducing a temporal dimension and requiring sequential planning for multiple agents. Unlike Q1, where agents were isolated, here each new agent must be routed while avoiding conflicts with the trajectories of already planned agents. This transforms the problem into a time-expanded pathfinding challenge, where agents become dynamic obstacles. To address this, I experimented with two approaches: an initial time-aware A* search, and a more effective Safe Interval Path Planning (SIPP)-inspired algorithm that balances feasibility with computational efficiency.

The third task (Q3) extends the challenge further into full-scale multi-agent scheduling with deadlines, malfunctions, and dynamic replanning. Here, the planner must generate collision-free schedules for all trains simultaneously while managing unexpected disruptions during execution. My design combined a robust SIPP-heading planner for single agents with higher-level strategies including Prioritised Planning (PP), Large Neighbourhood Search (LNS), and targeted replanning. Across four iterations, I addressed practical challenges such as deadlocks, corridor conflicts, memory usage, and replan time. The final system achieved a 58% F-score.

II. Question 1 Implementation

A. Methodology

I implemented the A search algorithm* with a Manhattan distance heuristic to compute the shortest path from the starting position to the goal. A* was chosen because it balances exploration and efficiency, guaranteeing both completeness and optimality when an admissible heuristic is used.

The **Manhattan distance** was selected as the heuristic because the environment is grid-based and allows four cardinal directions (N, E, S, W). This heuristic is admissible (never overestimates true cost) and consistent, ensuring optimality.

B. Theoretical Analysis

Completeness: A* is complete if the search space is finite and step costs are positive. In Flatland, both conditions hold, so a valid path will always be found if one exists.

Optimality: Since Manhattan distance is admissible, the returned path is guaranteed to be optimal in terms of path length.

Time Complexity: In the worst case, A* explores O(b^d) nodes, where b is branching factor and d is the depth of the shallowest solution. With a 4-directional grid, b=4, though pruning via the heuristic significantly reduces exploration.

Space Complexity: O(b^d) due to storage of open/closed sets. This is acceptable for Q1 since instances involve only one agent.

C. Experimental Results

The solution was tested against 40 hidden evaluation cases provided by the server. The results are summarised below:

Total Agents	Agents Done	Dead-lines Met	SIC	Make-span	Pen-alty	Final SIC	F Score
40	40	40	1681	1681	0	1681	100%

The implementation achieved a perfect F-score of 100%, matching the staff's optimal benchmark. All 40 agents reached their targets with zero penalties. The SIC and makespan values confirmed that the algorithm produced shortest paths without redundancy.

D. Critical Discussion

The A* algorithm proved highly effective for this simplified problem. Its strengths are clear:

- Optimality: Guaranteed shortest paths, directly reflected in the perfect F-score.
- Efficiency: Low computation time (near-instantaneous plan generation).
- Simplicity: Straightforward implementation with a clear heuristic.

However, there are limitations:

- Scalability: Memory and time costs could increase drastically with larger grids or when extended to multi-agent contexts (Q2/Q3).
- Domain awareness: A* does not exploit domain-specific constraints beyond grid geometry; more advanced heuristics (e.g., rail topology-based heuristics) could improve efficiency further.
- Single-agent assumption: The absence of time and collisions makes Q1 artificially easy; the method does not generalise directly to later problems.

III. Question 2 Implementation

A. Methodology

1) Initial Approach: Time-Aware A*

My first attempt extended the basic A* algorithm to incorporate time. Each state was represented by a tuple (x, y, direction, t), where t was the timestep. At each expansion, the algorithm checked for:

- Vertex conflicts: whether another agent already occupied the same cell at the same timestep.
- Edge conflicts (swaps): whether two agents attempted to traverse the same edge in opposite directions simultaneously.
- Wait actions: an agent could stay at the same cell if it was unoccupied at the next timestep.

The heuristic used was Manhattan distance, as in Q1, ignoring time. This ensured admissibility since each move or wait cost one timestep.

Although straightforward, this approach treated every possible timestep as a new state, leading to excessive branching and state space explosion, particularly in

congested scenarios. While it successfully avoided collisions, it failed to plan efficient routes for many agents within the allowed search horizon.

2) Refined Approach: Time-Expanded A* (SIPP-style)

To address these limitations, I implemented a time-expanded A based on the Safe Interval Path Planning (SIPP) framework*. Instead of treating each cell-timestep pair as an independent state, this method represents states as safe intervals during which a cell remains unoccupied.

Key design elements included:

- Reservation tables: I pre-computed position and edge reservations from existing agent paths.
  - pos_reserved[(x,y)] = timesteps when (x,y) is occupied.
  - edge_reserved[(a,b,t)] = an edge from a →b used at timestep t.
- Successor generation: At each step, the algorithm only expanded actions (move or wait) that respected reservations.
- Heuristic: Manhattan distance was retained, which remained admissible as each action cost one timestep.
- Path reconstruction: Once the goal was reached, the path was reconstructed backwards from the parent map.

This method reduces redundant expansions, as it avoids exploring "unsafe" states preemptively, instead restricting search to feasible intervals.

B. **Theoretical Analysis**

Completeness: Both approaches are complete under the assumption of bounded timesteps (max_timestep) and positive costs. However, SIPP improves completeness in practice by pruning unsafe states early.

Optimality: With an admissible heuristic, A* and SIPP guarantee optimal solutions (shortest time paths) relative to the reservation table. However, sequential planning introduces suboptimality globally, as earlier agents' paths constrain later ones.

Time Complexity:

- Time-aware A*: O(b·T), where b is branching factor and T the planning horizon, often exploding in dense traffic.
- SIPP-style: Reduces branching by grouping unsafe times, yielding practical improvements though still exponential in the worst case.

Space Complexity: Both algorithms require storing visited states, O(b·T). SIPP reduces redundant entries, improving memory efficiency.

C. **Experimental Results**

The two methods were evaluated on the hidden benchmark set of 2832 agents. Results are summarised below:

Algorithm	Total Agents	Agents Done	SIC	Makespan	Plan Time	F Score
Time-aware A*	2832	2343	1,262,139	12,213	72.44	40%
SIPP-style (final)	2832	2799	490,861	11,689	1173.83	>100%

The initial time-aware A* struggled to scale: it successfully routed only 2343 agents (82%), leaving nearly 500 unplanned. Its SIC was significantly higher (≈2.5× that of the staff baseline), yielding an F-score of 40%.

In contrast, the SIPP-style algorithm routed 2799 agents (98.8%), with a drastically lower SIC and shorter makespan. The longer plan time reflects additional overhead in reservation table management but was still acceptable given the 2-hour evaluation window. Crucially, the solution achieved an F-score greater than 100%, outperforming the staff baseline.

D. **Critical Discussion**

The contrast between the two approaches highlights the importance of time-efficient state representation in multi-agent pathfinding:

- Strengths of Time-aware A*:
  - Simple to implement, directly extending Q1's A*.
  - Conceptually clear, ensuring correctness with explicit timestep tracking.
- Weaknesses of Time-aware A*:
  - Severe state space explosion due to expanding every timestep.
  - Incomplete coverage of all agents under evaluation constraints.

- Poor F-score, reflecting inefficiency in congested maps.

- Strengths of SIPP-style planning:
  - Reservation-based pruning focuses search on feasible safe intervals.
  - Significantly reduces unnecessary expansions, improving both scalability and success rate.
  - Achieved near-optimal SIC and >100% F-score.

- Weaknesses of SIPP-style planning:
  - Sequential planning still imposes a "first-mover advantage," leading to suboptimal global schedules.
  - Planning time increased compared to naive A*, due to more sophisticated conflict checking.
  - Implementation complexity was higher, requiring careful reservation table design.

IV. Question 3 Implementation

A. Approach 1 — Prioritised Planning + Safe Interval Path Planning + Large Neighbourhood Search + Deadlock Detection

Motivation & method: For initial path planning, build a conflict-free baseline by planning agents sequentially (PP with SIPP-heading) and then run LNS to iteratively improve solution quality. Replan the affected agents (malfunction + failed execution), detect deadlocks via a wait-for graph and resolve cycles with targeted backoffs.

Strengths: Conceptually straightforward, guaranteed conflict-free initial solution, and LNS enables local optimisations that can fix priority-induced poor allocations.

Weaknesses & outcome: Susceptible to cascading replans when malfunctions occur; lazy planning introduced late failures; runtime and replan cost grew quickly with agent count. Achieved modest performance ($F \approx 43\%$) and failed to finish full-scale problems within the 2-hour evaluation window.

Lessons learned: PP+LNS is a solid baseline but needs stronger conflict-avoidance hooks and efficient reservation structures for large instances.

B. Approach 2 — Proactive Conflict Avoidance

Motivation & method: Observing many deadlocks in visualisations, I pivoted toward proactive prevention instead of reactive resolution. I removed lazy planning (plan all agents up-front), added deadlock prevention steps in the replan function where critical areas were reserved ahead of time, and used SIPP with relaxed reservation windows as fallback repairs.

Strengths: Reduced number of replans and improved the number of agents finishing before deadlines in isolated tests.

Weaknesses & outcome: Proactive checks heavily increased computation time; the solver still hit many corridor head-to-head deadlocks and could not finish high-difficulty tests within time limits. This highlighted that prevention must be targeted and lightweight to scale.

Lessons learned: Prevention helps, but indiscriminate reservation and expensive checks harm throughput.

C. Approach 3 — Corridor Direction Reservation

Motivation & method: Visual inspection revealed corridor head-to-head conflicts were the most damaging. I introduced corridor detection and directional reservations: when an agent enters a corridor segment, it reserves that segment in its travelling direction until it exits; opposite-direction entries are blocked until the corridor is clear.

Strengths: This solved the most frequent unrecoverable deadlock: stuck pairs in long, unbranching passages. With corridor-aware planning, the success rate rose substantially (reached $F \approx 49\%$), fewer costly replans were necessary, and the evaluation speed significantly increased.

Weaknesses & outcome: The corridor reservation data structures created additional memory overhead. On large maps the evaluation process ran out of memory and was killed for some tests. Reduced complexity LNS and priority simplifications partially compensated but did not eliminate memory issues.

Lessons learned: Structural, domain-specific constraints (corridors) are powerful, but they must be implemented with memory-efficient data structures.

D. Approach 4 — Unified Reservations and Replan Optimisation (Final)

Motivation & method: The final iteration combined the corridor idea with careful systems engineering.

Key design elements:

- Unified Reservation system: a single, compact structure for vertex, edge, and corridor reservations avoiding duplication and double-booking. Uses sparse maps keyed by timestep to minimize memory.
- Corridor map caching: compute corridor segments once and reuse globally to avoid repeated scans and copies.
- Horizon-limited reservations for unaffected agents: for instances with large number of agents, only a limited future window (e.g., 60 timesteps) is reserved so replanners have more flexibility and reservation tables remain bounded.
- Early termination & replan budget guards: prevents catastrophic repeated replans by returning conservative fallbacks when too many replans occur.
- Garbage collection & reservation clearing: free memory between LNS iterations and replans.
- Simplified conflict resolution: trade off complex cycle-handling for faster heuristics (slack-based victim selection).

Strengths: Balanced conflict prevention (corridor reservations) with efficient memory use. Completed all tests within time and improved robustness to malfunctions and deadlocks.

Outcome: Final aggregated performance: 58% F-score, passing the staff baseline. All test instances finished within the evaluator's time limits.

Total agents	Agents done	DDLs met	Plan Time (s)	SIC	Makespan	Penalty	Final SIC	F Score
2832	2164	1966	3614.37	1,793,958	39,686	562,407	2,356,365	58%

Lessons learned: Algorithmic ideas must be matched with effective engineering: compact reservation representations, caching, and horizon-limited reasoning are as important as novel heuristics.

E. Comparative discussion

The four approaches show a clear evolution:

- Start with a good algorithmic backbone (SIPP-heading + PP), but naive system design fails at scale (Approach 1).
- Add proactive conflict avoidance to reduce replans (Approach 2) — good conceptually, but heavy.
- Introduce domain-specific structure (corridor reservations) to solve the dominant failure mode (Approach 3) — powerful, but memory-sensitive.
- Finally, combine domain insights with unified, memory-efficient structures and pragmatic limits (Approach 4) — the approach that completed all tests and passed the baseline.

Two cross-cutting insights stand out: (1) conflict avoidance outperforms post-hoc resolution when the avoidance is targeted (e.g., corridors); (2) scalability is chiefly a systems problem, memory and reservation representation determine whether algorithmic gains can be realised.

V. Conclusion

Across the three stages of the Flatland Challenge, I progressively developed solutions that evolved from simple single-agent pathfinding to scalable multi-agent scheduling under uncertainty. Each stage provided not only technical results but also valuable insights into the strengths and limitations of different planning paradigms.

In Question 1, the A* algorithm with a Manhattan heuristic produced optimal single-agent paths, achieving a perfect score. This validated the correctness of classical heuristic search in a simplified setting and offered a strong baseline. However, it also underscored the limitations of such methods when scaling to more complex, multi-agent environments where collisions and time dimensions cannot be ignored.

In Question 2, the introduction of temporal dynamics and sequential planning revealed the inadequacy of naïve time-aware A*, which quickly became computationally expensive and incomplete. Transitioning to a SIPP-style time-expanded A* demonstrated how careful algorithmic refinement could achieve both efficiency and near-optimal performance. The >100% F-score confirmed theoretical expectations and highlighted the critical role of interval reasoning in handling dynamic obstacles. Still, the sequential framework was inherently limited, motivating the more ambitious multi-agent strategies required in Question 3.

Question 3 represented the culmination of these ideas, combining SIPP-heading for single-agent robustness with higher-level frameworks: Prioritised Planning, Large Neighbourhood Search, and targeted replanning.

Early approaches introduced effective concepts such as deadlock detection and corridor reservations but collapsed under scale due to memory usage and replanning overheads. Later iterations shifted focus toward system engineering: unified reservations, caching, bounded horizons, and lightweight replanning. These structural improvements proved decisive, enabling the final approach to achieve a 58% F-score.

Overall, this project illustrates that success in multi-agent pathfinding requires both algorithmic innovation and practical engineering trade-offs. The final planner balances correctness, runtime, and memory efficiency, and the iterative process demonstrates how theory, experimentation, and system optimisation together drive progress. Beyond Flatland, these lessons extend to broader applications in robotics, logistics, and autonomous transport, where scalable coordination under uncertainty remains a central challenge.

VI. Limitation and Future Work

While the final system achieved competitive performance, several avenues remain for improvement.

First, the framework could be extended to replan agents that fail to find a feasible path in the initial planning stage. At present, these agents are ignored completely and returned an empty list at the end, which limits overall throughput; enabling retry those agents during early replanning may allow more agents to reach their goals within deadlines and improve overall throughput.

Second, the implementation itself would benefit from improved code quality and maintainability. Over the course of iterative experimentation, the codebase accumulated technical debt (e.g., duplicated logic, partially overlapping data structures). A structured refactoring would reduce bugs, improve readability, and accelerate experimentation.

Finally, it is important to note that the contest server imposed strict hardware constraints: a single CPU core at 2 GHz and 4 GB of RAM. These limits strongly shaped design choices. Future work in richer environments could explore parallelisation, distributed planning, or more memory-intensive heuristics that were not feasible under these restrictions.

Together, these directions highlight opportunities to strengthen both the algorithmic capability and engineering robustness of the system, paving the way toward more scalable multi-agent planning solutions.

VII. REFERENCES

[1] Andreica, M.-I. (2019). *Winning Solution of the AIcrowd SBB Flatland Challenge 2019-2020*. ArXiv.org. https://arxiv.org/abs/2111.07876

[2] Erdmann, M., & T. Lozano-Perez. (1987). On multiple moving objects. *Algorithmica*, *2*(1-4), 477–521. https://doi.org/10.1007/bf01840371

[3] Laurent, F., Schneider, M., Scheller, C., Watson, J., Li, J., Chen, Z., Zheng, Y., Chan, S.-H., Makhnev, K., Svidchenko, O., Egorov, V., Ivanov, D., Shpilman, A., Spirovska, E., Tanevski, O., Nikov, A., Grunder, R., Galevski, D., Mitrovski, J., & Sartoretti, G. (2020). *Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World*. ArXiv.org. https://arxiv.org/abs/2103.16511

[4] Li, J., Chen, Z., Zheng, Y., Chan, S.-H., Harabor, D., Stuckey, P. J., Ma, H., & Koenig, S. (2021). Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. *Proceedings of the International Conference on Automated Planning and Scheduling*, *31*, 477–485. https://doi.org/10.1609/icaps.v31i1.15994

[5] Phillips, M., & Likhachev, M. (2011, May 1). *SIPP: Safe interval path planning for dynamic environments*. IEEE Xplore. https://doi.org/10.1109/ICRA.2011.5980306

[6] Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher & J. Puget (Eds.), *Principles and Practice of Constraint Programming — CP98* (pp. 417–431). Springer Berlin Heidelberg.