Reasoning and Planning
2025/2026

Taxi routing
ASP multiagent planning

Master of Science in Artificial Intelligence

Report

Jose Carlos Bordón Maldonado
Cristian Marquina Buitrago

ÍNDICE

1. Introduction and Problem Definition

This report documents the solution to the Taxi Routing multiagent planning problem, as defined in the exercise statement, using the temporal ASP system *telingo*.
The problem involves routing *T* taxis (numbered 1 to 9) to pick up *P* passengers (named 'a' to 'z') and deliver them to *S* designated stations ('X')1. The environment is a rectangular grid with obstacles ('#'). The primary constraint is that the number of stations must be greater than or equal to the number of passengers ($S \geq P$).

1.1. Environment and Constraints

The solution must satisfy several constraints for safety and feasibility, including:

- A taxi cannot drive through a building cell ('#').
- Collision Avoidance: Two taxis cannot be in the same cell simultaneously.
- Mutual Exclusion: Two persons (passengers) cannot be in the same cell simultaneously.
- Adjacent Swap: Two adjacent taxis cannot swap their positions.

1.2. Actions and Goal

At each time step, every taxi must perform exactly one of the following actions: Move (up, down, left, or right), Pick a passenger, Drop a passenger, or Wait.
The final goal is reached when each passenger is outside a taxi at any of the station positions.

2. Implementation Methodology

The solution follows the standard ASP incremental planning workflow, separating the domain facts from the general planning rules.

2.1. Fact Generation (encode.py)

The Python script (encode.py) converts the ASCII grid input into ground facts for the ASP solver. This ensures the separation of the general encoding (taxi.lp) from the problem instance (domain.lp).

2.2. Temporal Model Structure (taxi.lp)

The core planning logic is implemented in taxi.lp using the *telingo* architecture:

- Initialization (#program initial): Defines the starting positions and taxi freedom at *t=0*.
- Dynamics (#program dynamic): Contains the laws of motion, effect axioms, and inertia axioms, crucially using the Previous Operator (●) represented by the single quote ('p). This block includes the choice rule to select one action per taxi.
- Goal (#program final): Defines the successful delivery (delivered(P)) and enforces the overall goal using a constraint (:- not goal.).

3. Design Questions

The following addresses key design and assessment questions inherent in the exercise statement.

3.1. What is the format of the solution (plan)?

The solution format is a finite temporal trace of stable models. This trace is presented as a sequence of the actions performed by all agents at each time.

- Output: The plan is explicitly shown using the #show directive. The output sequence contains the actions move(T, D), pick(T), drop(T), and wait(T), indexed by their time step (e.g., "State 1:", "State 2:", etc.)

3.2. How is the minimum plan length guaranteed?

The minimum plan length is guaranteed by the incremental ASP solving mechanism.

- Mechanism: The *telingo* system begins by searching for a solution at an initial time horizon and then increments the number of calls/states until a model is found. The first model obtained is, by definition, the shortest possible plan.

3.3. How is the heuristic rule "never pick the same person twice in the same taxi" implemented?

To implement the heuristic rule "never pick the same person twice in the same taxi," a state predicate is needed to track history. This requires leveraging the temporal nature of the encoding.
A new predicate, picked_before(T, P), can be defined and enforced using a constraint:

- Definition and Inertia:
  picked_before(T, P) :- 'picked_before(T, P).
  picked_before(T, P) :- pick(T), 'at(T, R, C), 'passenger_at(P, R, C), 'free(T).
- Constraint:
  :- pick(T), picked_before(T, P). (This forbids the action pick(T) if *T* has ever picked *P*).

4. Analysis of Solutions

The execution logs confirm that *telingo* successfully finds a minimum-length plan for each instance, reporting the time spent for the first model and total time.

4.1. Performance and Horizon Complexity

The solving time is strongly correlated with the plan length (horizon) and the complexity of coordination required. The total time includes solving time, time for finding the first model, and time spent proving unsatisfiability up to the required horizon.

- Low Complexity: Simple instances like sol02 (6 steps, 0.023s) and sol01 (8 steps, 0.029s) are solved very quickly, indicating straightforward paths.
- Single-Agent Complexity: Long traces, such as sol04 (20 steps, 0.341s) and especially sol07 (26 steps, 812.604s), demonstrate that even single-agent problems can become computationally expensive when the search horizon is large, requiring the solver to make many unsuccessful calls (Calls: 27) before finding the solution.
- High Coordination Cost: The most significant increase in time is seen in multi-agent problems, even for moderate plan lengths. This is due to the dense constraints (collision, anti-swap) that force the solver to navigate a vast, restricted search space.
  - sol08.txt (3 taxis, 22 steps) takes 24.095s.
  - sol09.txt (3 taxis, 12 steps) takes 31.212s, highlighting that the coordination difficulty is a greater factor than the plan length itself.
  - sol10.txt represents the highest complexity, taking 130.147s for a 23-step plan.

4.2. Conclusion on Performance

The results confirm that the solving time increases significantly with the plan length and the degree of synchronization required, validating the utility of the incremental approach despite the computational challenge of planning in TEL_f. The total number of Calls made by the solver accurately reflects the search horizon *k*.