Taxi Routing Problem – ASP/Telingo Report

- **Francisco Manuel Vazquez Fernandez ([franciscomanuel.vazquez.fernandez@rai.usc.es])**
- **Gian Paolo Bulleddu ([gianpaolo.bulleddu@rai.usc.es])**

---

1. Problem Overview

The **Taxi Routing** problem is a planning task in which a fleet of taxis must cooperatively transport passengers to designated station cells within a grid-based environment. The environment contains static obstacles (walls) and multiple dynamic agents (taxis and passengers), requiring careful coordination to avoid conflicts.

Each taxi operates under strict constraints: it moves only in cardinal directions, can carry at most one passenger at a time, and must avoid collisions with other taxis. Additionally, passengers cannot share a cell unless one is inside a taxi, and taxis are forbidden from swapping positions in a single time step.

The objective is to compute a valid plan that results in all passengers being delivered to station cells and released from taxis, while satisfying all movement, collision, and capacity constraints.

2. Methodology and Implementation

The problem was modeled and solved using **Answer Set Programming (ASP)** with **Telingo**, an extension of Clingo that supports temporal reasoning.

2.1 Domain Encoding

The grid maps provided in ASCII format (domXX.txt) were converted into ASP facts using a Python script (encode.py). This script generates predicates representing taxis, passengers, walls, free cells, and station locations.

2.2 ASP Model Structure

The main logic is implemented in taxi.lp using Telingo's modular structure:

- **#program initial**
  Defines the initial positions of taxis and passengers.
- **#program dynamic**
  Encodes the system dynamics:

- **Action selection:** Each taxi chooses exactly one action per step (move, pick, drop, or wait).
- **State transitions:** Inertia rules and action effects determine how taxi and passenger locations evolve. The prime operator (') is used to reference the previous state.
- **Constraints:** Hard constraints enforce collision avoidance, wall blocking, valid pickup/drop behavior, capacity limits, and the prohibition of position swaps.

- **#program final**
  Specifies the goal condition: every passenger must be located at a station cell and not be inside any taxi.

This structure provides a clear and maintainable separation between initialization, evolution, and goal checking.

3. Experimental Results

The implementation was evaluated on the ten benchmark instances provided with the exercise (dom01–dom10). For each instance, the generated plan length was compared with the reference solution.

Here is a complete guide on how to run the Taxi Routing project and a breakdown of the code structure.

4. How to Run

Before running the project, ensure you have the following installed:

- **Python 3**
- **Telingo:** The ASP solver for temporal logic (pip install telingo or via Conda).
- **Pygame:** Required for the visualization (pip install pygame).

4.1. Project Structure

- **Root Directory:** Contains run.py, encode.py, taxi.lp, drawtaxi.py, compare.py, and problem.txt.
- **extaxi/:** Contains the input domain files (dom01.txt to dom10.txt) and reference solutions (sol01.txt...).
- **picstaxi/:** Contains the images (taxi.png, wall.png, etc.) used by the visualizer.

- **results/:** This will be created automatically to store generated .lp files and solutions.

4.2 Execution

Option A:

Use the run.py master script to execute the entire pipeline (Encoding → Solving → Visualization) for a specific instance.

Command:

```
python3 run.py <instance_number>
```

Example:

Bash

```
python3 run.py 02
```

- **What happens:**
  1. Encodes extaxi/dom02.txt into results/domain_02.lp.
  2. Runs **Telingo** using taxi.lp and the generated domain.
  3. Saves the output to results/solution_02.txt.
  4. Opens the graphical simulation.
- **Controls:** In the simulation window, press **SPACE** to advance one step, or **Q** to quit.

Option B: Manual Execution

If you want to run each step individually:

1. **Encode the Map:**

```
python3 encode.py extaxi/dom02.txt results/domain_02.lp
```

2. **Solve with Telingo:**

```
telingo taxi.lp results/domain_02.lp > results/solution_02.txt
```

3. **Visualize:**

```
python3 drawtaxi.py extaxi/dom02.txt results/solution_02.txt
```

4.3 Verify Results

To compare your generated solutions against the reference solutions provided in the extaxi folder:

```
python3 compare.py
```

This script checks the plan length and number of actions for all available solutions in the results/ folder.