Assignment 1: Mixed-Integer Programming

COMP4691-8691 2024
School of Computing, ANU

Guidelines

- Answers to go in `answers.pdf` (replace the file with your pdf).
- In order to get full marks for the implementation parts, your code should be clear, well commented, sensibly structured and correct.
- You might find it easier to write out your MILP model mathematically first, before translating that into a PuLP model.

Space Invader Attack!

Your employer has come up with the concept for a game that turns space invaders on its head and mixes it with tower defence. It is sure to be a flop, but you'll get paid to develop it all the same.

The player's goal is to invade Earth, by carefully selecting spaceship (UFO) attack trajectories. The computer AI uses fixed position lasers to attempt to target and destroy these UFOs, before they reach Earth.

Your job is to come up with the AI to operate the lasers and defend earth from the player. The game AI will have three difficulties: an easy mode that utilises a heuristic, a hard mode that utilises MILP, and an ultra-hard mode that utilises MILP and changes the game rules so that the lasers can move.

AI Problem Description (Easy and Hard Mode)

Space: The game takes place in a 2D continuous space, represented by x and y coordinates. The x-axis represents the Earth's surface (a zero or negative y coordinate is the Earth, a positive y coordinate is free space).

Distance: Distances should be calculated as a "Manhattan" distance, rather than an Euclidean distance (think why this might be convenient when implementing a MILP).

Time: The game takes place over a number (`timesteps` in the `Problem` class) of discrete time steps.

UFO: Each UFO has a fixed trajectory, represented by a (x, y) point for each time step. Each UFO starts off with a different amount of "health". When a UFO's health reaches zero, it transitions from being active to destroyed.

Laser: Lasers are located at fixed positions in space. The decision variable for each laser in each time step is which UFO to target, if any. Each laser can only target a single UFO in each time step. A UFO is a **valid target** when the following conditions are met:

- it is active;
- it hasn't yet reached the Earth (when it still has a positive y position); and
- it is within range (distance) of the relevant laser, given by a laser's minimum and maximum range parameters.

A laser will deal a variable amount of damage to a targeted UFO, that decreases linearly with distance. This is represented by the base damage and falloff parameters through the following relationship:

$$damage(distance) = base\_damage + falloff \times distance \quad (1)$$

Objective: The objective is to destroy as many UFOs as possible within the fixed number of time steps (or equivalently to minimise the remaining active UFOs).

Events: The order of events in each time step are:

1. Each UFO moves to its new position.
2. Each laser targets and shoots at most a single UFO in range that has a positive y coordinate, doing damage to their health that is distance dependent.
3. A UFO is destroyed if its health is reduced to zero.

Framework and Implementation

You have been given some code to get you started. You will program the AI in `invaders.py`, utilising the classes `Problem`, `Laser` and `UFO`, and visualisation in `framework.py` as convenient. You can modify the code in `framework.py` if it helps for debugging and other purposes, but you should not need to.

`invaders.py` can be called as a script, by providing it with one of the instance files `invad-{1,2,3}.json`. By default it saves the solutions your AI comes up with as JSON and PNG files (e.g., see figure 1 for example PNG output). Additionally, it can produce an animation of the solution if you pass it the right flag (but please do **not** commit these MP4 files to your git repository).

The script has two other command line flags, that are intended to be used to enable the extra modelling details that are introduced in later parts of this assignment. Please made use of these in your code, so that that we can check your code for all parts of the assignment, by enabling and disabling these flags.

1 Easy Mode Heuristic [5 marks]

The "easy mode" heuristic is as follows for each time step: using lexical ordering (from first to last as they appear in the list), each laser selects its closest (by Manhattan distance) UFO that is a **valid target**. Ties are broken with lexical ordering over the UFOs.

Complete the heuristic for the AI which has been started in the function called `invaders_easy`. Ignore the `cooldowns` and `disabling` flags for now, they will come in later questions. Similarly, ignore the `cooldown`, `size` and `speed` attributes of the `Laser` class.

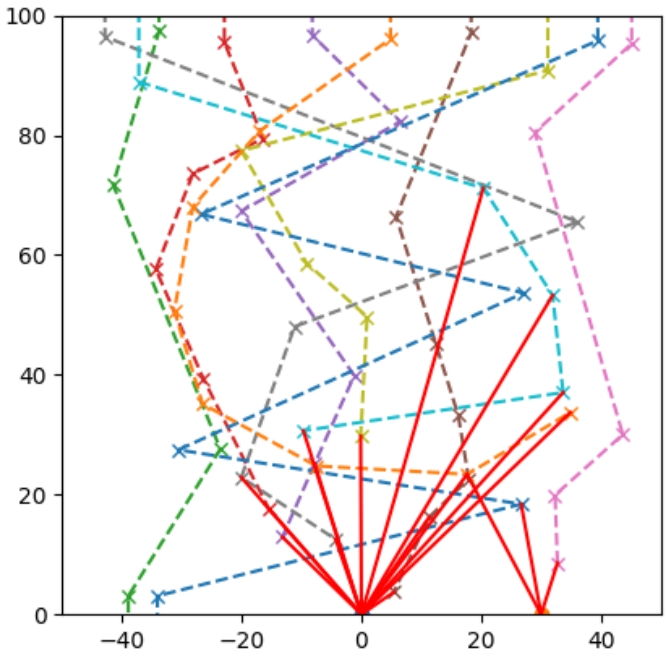

Figure 1: The easy mode solution to `invad-3.json`, with two lasers stationed at the bottom, and ten UFOs, all of but two of which are destroyed.

Hint: If you get stuck with the heuristic implementation, you can perhaps repurpose some of the code in the `when_destroyed` function in `framework.py`.

Hint: You should expect to get a solution of 10 out of 18 remaining on instance `invad-2.json`.

2 Hard Mode MILP [30 marks]

Complete the "hard mode" MILP implementation of the AI in the function `invaders_hard`. This should produce an optimal solution to the AI problem.

If you relax any constraints (here or in later questions), be sure to explain this in a code comment and justify why you expect to still get an exact solution for the objective.

For each instance `invad-{1,2,3}.json`, report in a table the objective value (the number of UFOs remaining) obtained by the easy and hard mode models.

Hint: The health of a UFO is a continuously varying quantity. If you try to implement it as an integer you will run into problems with infeasibility.

3 Cooldowns [20 marks]

We will now add an additional complication to the problem, where each laser has to wait a fixed number of time steps between targeting different UFOs, given by its `cooldown` parameter.

Implement this for both the easy mode heuristic [**5 marks**] and hard mode MILP [**15 marks**] AIs. Ensure this feature can be switched on and off via the command line flag. Note that the performance of your MILP model will be taken into account (we expect solutions in under 30 seconds), so you may want to try a few different modelling approaches.

For each instance `invad-{1,2,3}.json` report in a table:

- The objective value (the number of UFOs remaining) obtained by the easy and hard mode models.
- The MILP solve time.
- The objective value of the MILP root node linear relaxation.
- The first feasible solution the MILP solver finds.
- The optimality gap of the easy mode heuristic solution, relative to the MILP root node linear relaxation.

Discuss whether or not our heuristic provides a superior solution compared to the first solution the MILP solver finds.

Hint: You'll want to turn on the CBC verbose message printing in PuLP. See the lectures for how to do so. Also, MIP lab question 7 on car assembly will help with understanding the solver output.

4 Disablement [20 marks]

We will add a further complication to the problem by adding the ability for active UFOs to disable our lasers if they touch them. A UFO touches a laser if it its position gets to within a (Manhattan) distance given by the `size` parameter of a laser. If disabled, the laser remains disabled for the remainder of the game, and can no longer target any UFOs. Make use of the `disabling` command line flag to enable this feature of the problem, and implement it for the easy mode heuristic [**5 marks**] and hard mode MILP [**15 marks**].

With this addition, the order of events at each time step becomes:

1. Each UFO moves to its new position.
2. If an active UFO is "touching" any laser, that laser is disabled for the remainder of the game.
3. Each laser that hasn't been disabled targets and shoots at most a single UFO in range that is has a positive y coordinate, doing damage to their health that is distance dependent.
4. A UFO is destroyed if its health is reduced to zero.

For each instance `invad-{1,2,3}.json` report in a table the objective value (the number of UFOs remaining) obtained by the heuristic and MILP models with both cooldowns and disabling enabled at the same time.

5 Ultra-Hard Mode MILP [25 marks]

We similarly use MILP to solve the ultra-hard mode AI, but change up the rules of the problem to turn the odds in the AI's favour. We allow lasers to fly about instead of remaining in a fixed position. The `pos` parameter for a laser now indicates the starting position. The laser can move at a maximum speed given by the `speed` parameter. This is the maximum Manhattan distance the laser can move between one time step and the next.

The additional modifications are that we **no longer** have to consider cooldowns, or lasers being disabled, and we can also ignore the laser minimum range.

Complete the "ultra-hard mode" MILP implementation of the AI in the

function `invaders_ultra` [**20 marks**]. Note that this function should also output the movement information of each laser in addition to the targeting information. You should copy over any code that is still relevant from your `invaders_hard` function.

You can assume the laser `falloff` parameters will always be negative. You will also likely need to assume some bound on the maximum distance that a laser can be away from any UFO at any point in time. A value of 300 should be a good starting point.

Report the number of remaining UFOs you get for your ultra-hard mode implementation on each of the three instances.

Experiment with the assumed maximum distance bound described above on the `invad-2.json` instance. If you instead use value of 150 and 600, what happens to the solution / solve process? Explain these changes given what you know about your model and MILP solving [**5 marks**].