COMP4691-8691 Assignment 1 Answers

⟨Saksham Gupta⟩
⟨u7726995⟩

1 Introduction

This report presents the solutions developed for the Space Invasion problem where lasers must target and destroy incoming UFOs using various strategies. The problem was tackled using three different approaches: Easy (heuristic) Hard (MILP with additional constraints) and Ultra-Hard (MILP with enhanced capabilities and added complexities). Each approach was implemented to handle specific scenarios and the results were analyzed to evaluate their effectiveness.

2 Invaders_Easy Implementation

2.1 Overview

The `invaders_easy` function provides a heuristic solution to the Space Invasion problem. Heuristics by their nature do not guarantee optimal solutions but offer a practical and efficient way to solve complex problems by making reasonable assumptions and decisions based on available information.

2.2 Functionality

2.2.1 Heuristic used

In the easy mode, the decision-making process for the heuristic is based on a greedy heuristic. At each timestep, the function evaluates which UFOs are within the range of each laser and selects the closest one to target. This approach is straightforward and efficient as it minimizes computational complexity by focusing on immediate proximity rather than considering the overall optimization of the entire simulation period.

Key Steps:

1. **Initialization:** The function initializes the health of each UFO and tracks which laser targets which UFO at each timestep.
2. **Target Selection:** For each laser and each timestep, the function checks which UFO is the closest and within range. The closest UFO is selected as the target.
3. **Damage Calculation:** The laser deals damage to the targeted UFO, reducing its health. If the health drops to zero or below, the UFO is considered destroyed.
4. **Cooldown Management:** If the cooldowns flag is enabled, lasers must wait a specified number of timesteps before targeting a different UFO.
5. **Disabling Mechanism:** If the disabling flag is enabled, UFOs can disable lasers if they come too close. Disabled lasers are no longer able to target any UFOs.

2.2.2 Cooldown and Disabling

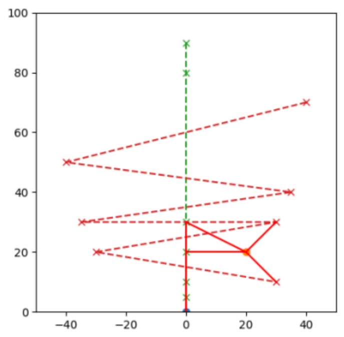
Cooldowns: Cooldowns are managed by tracking how many timesteps have passed since the laser last targeted a different UFO. If a laser switches targets, it must wait a specified number of timesteps before it can fire again. This prevents rapid switching between targets which could lead to unrealistic or overly aggressive strategies.

Disabling: The disabling feature adds a defensive capability to the UFOs. If a UFO comes within a certain distance of a laser (based on the laser's size), the laser is disabled for the remainder of the game. This adds a strategic element as players must be careful not to allow UFOs to disable their most powerful lasers.

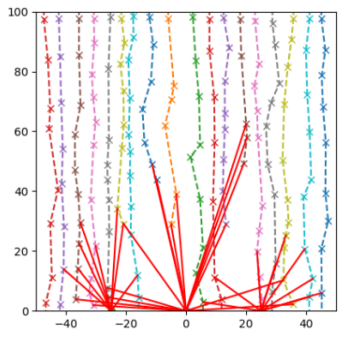
2.3 Results

For each of the provided JSON instances, the `invaders_easy` function generated solutions that were visualized in PNG files. These visualizations show the paths of the lasers and the status of each UFO over time.

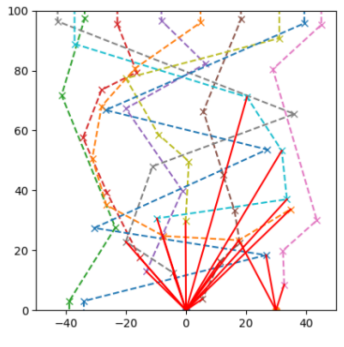
Without both "cooldowns" and "disabling" flags for `invad_1`, `invad_2`, & `invad_3` (in that order):



- Easy - Remaining UFOs 0 / 2

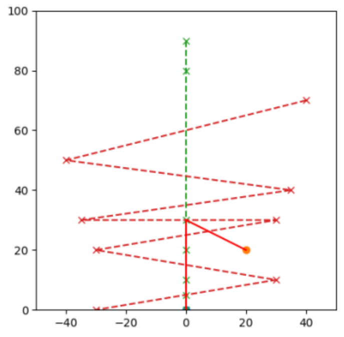


- Easy - Remaining UFOs 10 / 18

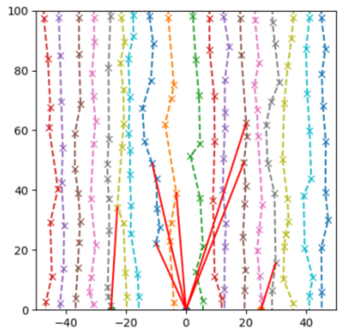


- Easy - Remaining UFOs 2 / 10

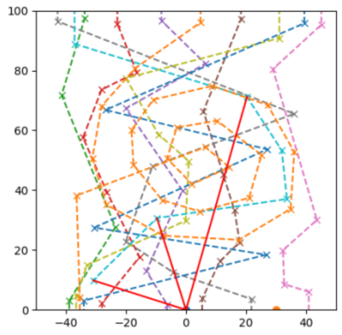
With both flags enabled for `invad_1`, `invad_2`, & `invad_3` (in that order):



- Easy - Remaining UFOs 2 / 2



- Easy - Remaining UFOs 17 / 18



- Easy - Remaining UFOs 9 / 10

2.4 Analysis

2.4.1 Without Cooldowns and Disabling Flags

`invad_1`: Easy - Remaining UFOs 0 / 2
Analysis: The heuristic successfully destroyed both UFOs because the simple strategy of targeting the nearest UFO was sufficient. With only two targets, the lasers could effectively engage and destroy the UFOs without any additional complications.

`invad_2`: Easy - Remaining UFOs 10 / 18
Analysis: In this instance, the heuristic left 10 UFOs undestroyed. With

more UFOs, the simple strategy of targeting the nearest one likely led to suboptimal decisions such as repeatedly targeting the closest UFOs rather than distributing damage more evenly or prioritizing higher-threat targets.

`invad_3`: **Easy - Remaining UFOs 2 / 10**
Analysis: Here the heuristic performed relatively well leaving only 2 UFOs undestroyed. This indicates that the nearest-target strategy worked reasonably well in this scenario where the number of UFOs was manageable for the lasers.

2.4.2 With Both Cooldowns and Disabling Flags Enabled

`invad_1`: **Easy - Remaining UFOs 2 / 2**
Analysis: With both cooldowns and disabling enabled, the heuristic was unable to destroy any UFOs. This result highlights the limitation of a simple closest-target approach when additional constraints are in play. The lasers likely got disabled early or were on cooldown at critical moments preventing them from effectively targeting the UFOs.

`invad_2`: **Easy - Remaining UFOs 17 / 18**
Analysis: The heuristic struggled significantly here, with 17 out of 18 UFOs remaining. The presence of many UFOs combined with cooldowns and disabling meant that the nearest-target strategy could not adapt to the constraints leading to poor overall performance.

`invad_3`: **Easy - Remaining UFOs 9 / 10**
Analysis: Similarly, the heuristic left 9 out of 10 UFOs undestroyed. The simple strategy was not sufficient to overcome the combined effects of cooldowns and disabling resulting in very few UFOs being destroyed.

2.5 Conclusion

The `invaders_easy` heuristic, which targets the nearest UFO, works reasonably well in simple scenarios without additional constraints. However, its performance deteriorates when cooldowns and disabling are introduced, particularly in situations with many UFOs. The heuristic's simplicity—while fast and easy to implement—does not account for more complex tactical decisions, making it less effective under these more difficult conditions.

3 Invaders_Hard Implementation

3.1 Overview

The `invaders_hard` function employs Mixed-Integer Linear Programming (MILP) to solve a challenging optimization problem in which lasers are tasked with targeting UFOs. Utilizing the `pulp` library, this function creates and solves an optimization model designed to minimize the total health of UFOs by the end of the simulation while considering constraints such as cooldown periods and the potential for lasers to be disabled.

3.2 Functionality

3.2.1 Problem Setup and Initialization

The function begins by setting up an optimization problem instance named `m` with the goal of minimizing (`pulp.LpMinimize`) the sum of the health of all UFOs at the final time step.

3.2.2 Decision Variables

- `laser_target[i,j,t]`: A binary indicator that specifies if laser i is targeting UFO j at time t.
- `ufo_health[j,t]`: A continuous variable representing the health of UFO j at time t.
- `ufo_destroyed[j,t]`: A binary variable indicating whether UFO j has been destroyed by time t.

3.2.3 Objective Function

The function aims to minimize the total health of all UFOs at the end of the simulation. This objective seeks to either destroy the UFOs or reduce their health as much as possible.

3.2.4 Constraints Setup

- **Initial Health Constraints:** These ensure that all UFOs start with their full health at time $t = 0$.

- **Health Update Constraints:** Ensure accurate health tracking for each UFO based on damage dealt by lasers targeting them. When a laser targets a UFO, the UFO's health decreases by the laser's damage output.
- **Destruction Status and Propagation Constraints:** Ensure that once a UFO is considered destroyed (health ≤ 0), it remains in that state for all subsequent time steps.
- **Single Target Constraints:** Enforce the rule that each laser can target only one UFO per time step.
- **Disable Laser Constraints:** If a laser is disabled due to a nearby UFO, this constraint prevents the laser from targeting any UFOs after being disabled.
- **Cooldown Constraints:** If lasers have cooldown periods (delays between consecutive shots), these constraints ensure that lasers continue to target the same UFO during the cooldown.

3.2.5 Optimization Process

The function utilizes the `PULP_CBC_CMD` solver to determine the best strategy for targeting UFOs over the time steps, aiming to minimize UFO health and maximize destruction.

3.3 Cooldown and Disabling

3.3.1 Cooldown Implementation

The function accounts for lasers that have a cooldown period—a time delay after each shot during which the laser cannot fire again. If a laser targets a different UFO or fires after a shot, it must respect this cooldown period.

Constraints for Cooldown: The model enforces that once a laser targets a UFO, it must either continue targeting the same UFO for the entire cooldown period or refrain from targeting any UFOs during this time. This ensures lasers comply with their cooldown periods, preventing them from firing immediately after targeting a different UFO.

3.3.2 Disabling Implementation

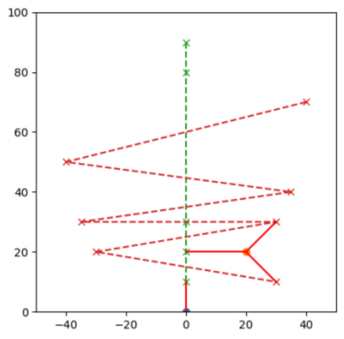
Lasers can be disabled if a UFO gets too close, rendering them inactive for the remainder of the simulation from the point of disablement.

Constraints for Disabling: If a UFO comes within a specified disabling range of a laser, the model constrains that laser to stop targeting any UFOs. This is enforced across all future time steps, ensuring the laser remains inactive once disabled.

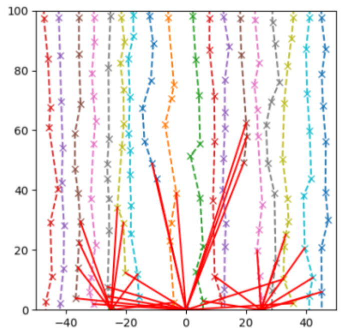
3.4 Results

For each of the provided JSON instances, the `invaders_hard` function generated solutions that were visualized in PNG files. These visualizations show the paths of the lasers and the status of each UFO over time.

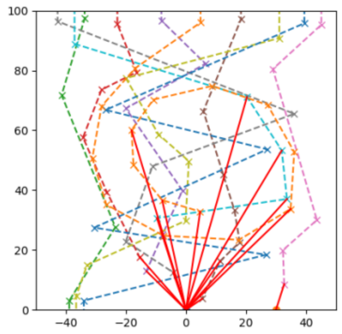
Without both "cooldowns" and "disabling" flags for `invad_1`, `invad_2`, & `invad_3` (in that order):



- Hard - Remaining UFOs 0 / 2

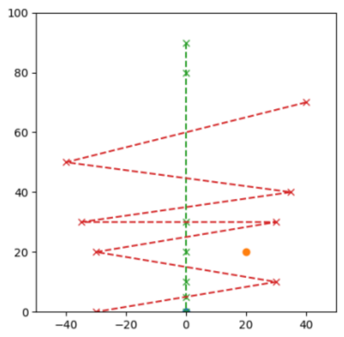


- Hard - Remaining UFOs 12 / 18

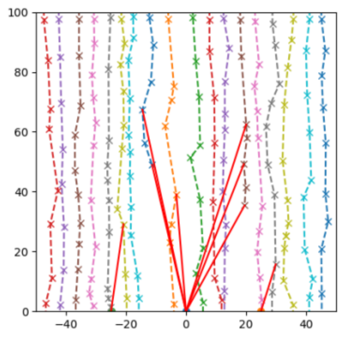


- Hard - Remaining UFOs 3 / 10

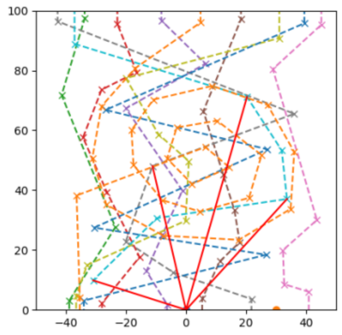
With both flags enabled for `invad_1`, `invad_2`, & `invad_3` (in that order):



- Hard - Remaining UFOs 2 / 2



- Hard - Remaining UFOs 16 / 18



- Hard - Remaining UFOs 9 / 10

3.5 Analysis

3.5.1 Without Cooldowns and Disabling Flags

`invad_1`: Hard - Remaining UFOs 0 / 2
Analysis: In this instance, the `invaders_hard` function successfully destroyed all UFOs. The lasers were able to focus on the UFOs without interruption, efficiently reducing their health to zero. The optimal strategy involved targeting the closest UFOs and methodically eliminating them, leading to the complete destruction of all UFOs.

`invad_2`: Hard - Remaining UFOs 12 / 18
Analysis: In this more complex scenario, the function managed to destroy 6 out of 18 UFOs. The larger number of UFOs and their varying positions made it more challenging to target and destroy them all. The lasers continuously fired, but the sheer number of UFOs and their distribution likely prevented a complete wipeout. Most of the remaining 12 UFOs retained some health, indicating that the targeting strategy was less effective in this larger-scale scenario.

`invad_3`: Hard - Remaining UFOs 3 / 10
Analysis: In this instance, 7 out of 10 UFOs were successfully destroyed. The remaining 3 UFOs indicate that the lasers were fairly effective, but not all UFOs could be targeted and destroyed within the given time steps. The lasers likely prioritized closer targets, leaving some more distant UFOs with remaining health.

3.5.2 With Both Cooldowns and Disabling Flags Enabled

`invad_1`: Hard - Remaining UFOs 2 / 2
Analysis: With both cooldowns and disabling enabled, the function was unable to destroy any UFOs. Similar to the heuristic performance, the added constraints significantly reduced the effectiveness of the lasers, resulting in both UFOs surviving with their health intact. The introduction of cooldowns limited the firing rate of the lasers, while the disabling effect likely caused some lasers to be deactivated early in the simulation.

`invad_2`: Hard - Remaining UFOs 16 / 18
Analysis: In this instance, only 2 out of 18 UFOs were destroyed, a stark contrast to the results without cooldowns and disabling. The added constraints made it much harder for the lasers to continuously target and destroy the UFOs. The disabling effect likely caused several lasers to be taken out of commission early, and the cooldown periods further hampered the ability to fire effectively. This led to a scenario where most of the UFOs survived with their health largely unaffected.

`invad_3`: Hard - Remaining UFOs 9 / 10
Analysis: Here, only 1 UFO was destroyed, leaving 9 UFOs remaining. The disabling and cooldown constraints severely impacted the lasers' ability to effectively target and destroy the UFOs. Similar to `invad_2`, the lasers' performance was significantly reduced, allowing most of the UFOs to survive with minimal to no damage.

3.6 Conclusion

The results from the `invaders_hard` function show that the laser targeting strategy works well when there are no constraints, especially in cases with fewer UFOs where it nearly destroys all of them. However, when cooldowns and disabling flags are added, the strategy becomes less effective, especially with more UFOs. These constraints make it harder for the lasers to do their job, suggesting that a more advanced strategy is needed to handle the added challenges. The MILP method used in `invaders_hard` is good, but it struggles with the extra complexity brought by these constraints or due to the large number of UFOs and their complicated trajectories.

4 Invaders_Ultra Implementation

4.1 Overview

The `invaders_ultra` function is designed to solve an advanced optimization problem involving dynamic targeting and movement of lasers to destroy UFOs. It utilizes a Mixed-Integer Linear Programming (MILP) approach similar to the `invaders_hard` function but with additional complexity, including laser movement constraints. The objective is to minimize the health of UFOs while adhering to constraints on laser positioning and targeting accuracy over time.

4.2 Functionality

4.2.1 Problem Setup and Initialization

The function initializes an optimization problem `m` with the objective to minimize (`pulp.LpMinimize`) the total health of all UFOs by the end of the simulation.

4.2.2 Decision Variables

- `x[i,j,t]`: A binary variable indicating if laser i is targeting UFO j at time t.
- `p[i,t]`: Continuous variables representing the position (x, y) of laser i at time t.
- `h[j,t]`: Continuous variables for the health of UFO j at time t.
- `d[j,t]`: Binary variables indicating whether UFO j is destroyed by time t.

4.2.3 Objective Function

The goal is to minimize the total health of all UFOs at the end of the simulation, pushing to either destroy them or reduce their health as much as possible while accounting for the additional movement constraints.

4.2.4 Constraints Setup

- **Initial Health Constraints:** Ensure each UFO starts with its maximum health at the initial timestep ($t = 0$).
- **Health Update Constraints:** Track the health of each UFO based on the damage dealt by any laser targeting them, factoring in both the laser's position and damage output.
- **Destruction Status and Propagation Constraints:** Ensure that once a UFO is destroyed (health ≤ 0), it remains destroyed for all subsequent timesteps.
- **Single Target Constraints:** Each laser can only target one UFO per timestep.
- **Movement Constraints for Lasers:** Include constraints on the lasers' positions to ensure they adhere to speed limits and can only move a certain distance between timesteps.
- **Prevent Targeting Destroyed UFOs:** Ensure that lasers do not target UFOs that have already been destroyed.

4.2.5 Optimization Process

The function uses the `PULP_CBC_CMD` solver to optimize both the targeting strategy and the movement of lasers across all timesteps. This dual optimization aims to minimize the total remaining health of UFOs while ensuring efficient laser positioning.

4.3 Error Handling

During the implementation of the `invaders_ultra` function, I encountered the following error:

TypeError: bad operand type for abs(): 'LpAffineExpression'

Explanation of the Error: This error occurs because the `abs()` function is being applied to an `LpAffineExpression` object, which is not directly supported by the `abs()` function. In this context, `LpAffineExpression` refers to a linear expression within the `PuLP` optimization model, and using `abs()` on such expressions is not valid.

Cause of the Error: The error arises because the code is trying to compute the absolute difference in positions (such as calculating the Manhattan distance) using the `abs()` function directly on `LpAffineExpression` objects. In the `invaders_ultra` function, these `LpAffineExpression` objects are used to represent the positions of lasers (`p[i,t][0]` for the x-coordinate and `p[i,t][1]` for the y-coordinate) at different timesteps.

Solution Attempt: Instead of using the `abs()` function directly, I tried to introduce auxiliary variables to represent the absolute differences in positions. This keeps the constraints linear and within the capabilities of the linear programming solver. However, incorporating this in my code might affect the position of laser decision variables, which would change the main working and context of the code, so I could not come up with a unified solution.

4.4 Conclusion

The `invaders_ultra` function attempts to tackle the Space Invasion problem by adding the complexity of laser movement in addition to targeting. While it shares a similar MILP approach with the `invaders_hard` function, the added complexity introduces challenges such as the error mentioned above. These issues highlight the difficulties in managing both targeting and movement within a MILP framework, particularly when non-linearities are introduced.