Project 3 - Optimization and PDDL

Hoang-Dung Bui,
George Mason University
Fairfax, USA
hbui20@gmu.edu

I. Introduction to constraint-based Optimization

A. Implementing some example Programs

For the problem in Fig. 1, the objective value is $z = 7$, with $x_1 = 0, x_2 = 2, x_3 = 1, x_4 = 3$

$$\begin{aligned} \max \quad & z = x_1 + 2x_2 + x_4 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 = 3, \\ & x_1 + x_2 + x_4 \leq 5, \\ & x_3 \geq 1 \\ & x_1, x_2, x_4 \geq 0 \end{aligned}$$

Fig. 1

For the problem in Fig. 2, the objective value is $z = 36$, with $x_1 = 2, x_2 = 6$

$$\begin{aligned} \max \quad & z = 3x_1 + 5x_2 \\ \text{s.t.} \quad & x_1 \leq 4, \\ & 2x_2 \leq 12, \\ & 3x_1 + 2x_2 \leq 18 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Fig. 2

B. A word Problem

The problem is defined as follow:

$$\begin{aligned} \min \quad & z = 180x_1 + 160x_2 \\ \text{s.t.} \quad & 0 \leq x_1 \leq 6, \\ & 0 \leq x_2 \leq 6, \\ & 4x_1 + 2x_2 \geq 12, \\ & 5x_1 + 4x_2 \geq 8, \\ & 4x_1 + 8x_2 \geq 24, \end{aligned}$$

Fig. 3

Question How many variables are there in the MILP? How many constraints?
Answer: there are two variables: x_1, x_2 are the numbers of work day at mine Heigh Ho and Kessel, respectively. There are total five constraints.

The MILP problem to solve the problem is shown in Fig. 2
Results: the value of objective function is: 680, and $x_1 = 2, x_2 = 2$

```
solver = pywraplp.Solver.CreateSolver('SCIP')
infinity = solver.infinity()
# Define your variables (and their domains)
x1 = solver.NumVar(0, 6, 'x1')
x2 = solver.NumVar(0, 6, 'x2')

# Add the constraints and the objective
solver.Add(4*x1 + 2*x2 >= 12)
solver.Add(5*x1 + 4*x2 >= 8)
solver.Add(4*x1 + 8*x2 >= 24)
solver.Minimize(180*x1 + 160*x2)

# Solve
status = solver.Solve()

# Print the outputs
if status == pywraplp.Solver.OPTIMAL:
    print('Solution:')
    print('  Objective value: ', solver.Objective().Value()
    print('  Number of variables: ', solver.NumVariables())
    for var in solver.variables():
        print(f' {var.name()} == {var.solution_value()}')
else:
    print('The problem does not have an optimal solution.')
```

Fig. 4

II. Trajectory Optimization with MILPs

A. Computing a Trajectory Given Thrusts

After finishing the code for a simple 1D space vehicle, we have the plots of the vehicle's trajectory and velocity.

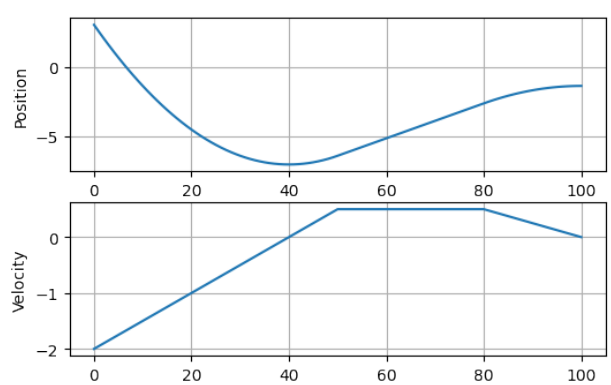

Fig. 5

As shown in Fig. 5, the final position of the vehicle is -1.375 and at the point the $velocity = 0.00$

B. Trajectory Optimization

After adding some code inside the *build_solver*, we plotted the vehicle's trajectories with different number of samples.

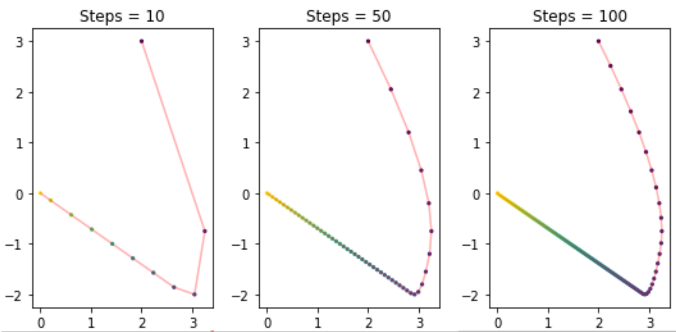

Fig. 6

As the number of sample points increases from 10 to 100, the trajectories are smoother and ending at the origin.

The optimization objective as a function of time step is shown in Fig. 7. As number of step increases, the objective value approaches the value 3.5.

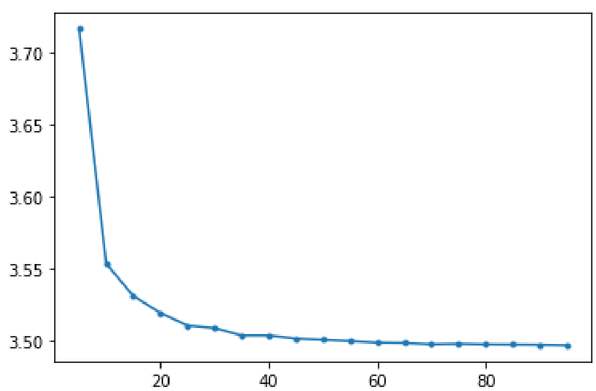

Fig. 7

C. Adding an obstacle

As adding an obstacle into the environment, I added the constraints into the motion planner, however, there is something wrong with the constraints, and the path did not avoid the obstacles. The results are shown in Fig. 8 and 8.

The constraints for the obstacle is shown in Fig. 10

III. PDDL

A. Running Fast Downward

We built *Fast Downward*, and ran the *probBLOCKS-10-0.pddl* module with two different planner: *astar* and *ff* heuristic. The results are shown in Table I

	plan length	expanded states	running time
astar	34	1 385 559	5.6877 s
ff heuristic	48	106	0.00478 s

TABLE I: Results of PDDL with two planners: Aster and ff heuristic

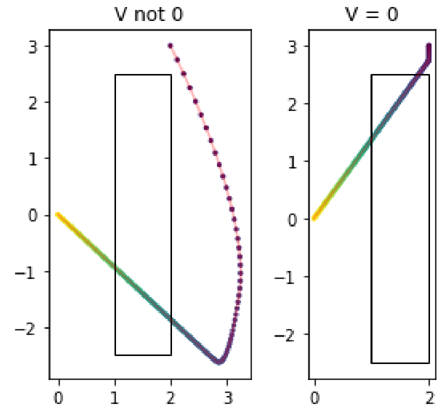

Fig. 8

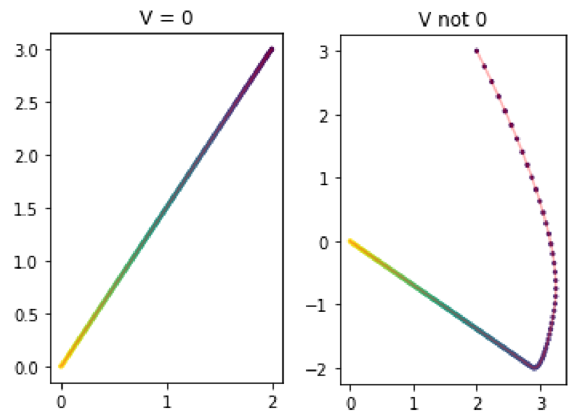

Fig. 9

The *astar* planner provided the optimal path (only 34 steps). However, to have that optimal path, it needed to discover a tremendous state space (1 385 559) and took a long time to find out (5.6877). This method can be used in small state space and simple scenario or used in experiment as a based line to compare other methods.

The *ff* heuristic one provided a non-optimal path (48 steps) (because the number of expanded state is small), but the computation time is very short. This method can be used in large or complicated environment with a huge state space. It also can be used in real time robot application.

B. An Example of PDDL Problem

After running the impulse drive, to travel from earth to Levinia, we will do in 3 steps:

1) Travel from the *earth* to *vulcan*: need cost of 10
2) Travel from *vulcan* to *qonos*: need of of 6
3) Travel from *qonos* to *Levinia*: the cost of 500.
4) The total cost is 516.

After building the *warp_drive* jump, the planner ran very well and the traveling as the following:

1) travel-impulse-speed enterprise earth vulcan (10)
2) beam-up-supplies enterprise vulcan plasmaconduit1 warpdrive1 (1)

```python
bs = [[solver.NumVar(0, 1, f'b_{step}_{side}')
       for side in range(4)]
       for step in range(num_steps)]
M = 100000
for ob in obstacles:
  for i in range(1,len(xs)):
      # print(i)
      solver.Add(xs[i][0] - ob[0] - M*bs[i-1][0] <= 0)
      solver.Add(xs[i][0] - ob[1] + M*bs[i-1][1] >= 0)
      solver.Add(xs[i][1] - ob[2] - M*bs[i-1][2] <= 0)
      solver.Add(xs[i][1] - ob[3] + M*bs[i-1][3] >= 0)
      solver.Add(bs[i-1][0] + bs[i-1][1] + bs[i-1][2]+ bs[i-1][3] <= 3)
```

Fig. 10

3) travel-impulse-speed enterprise vulcan qonos (6)
4) beam-up-supplies enterprise qonos warpcoil1 warpdrive1 (1)
5) travel-impulse-speed enterprise qonos betazed (10)
6) beam-up-supplies enterprise betazed plasmainjector1 warpdrive1 (1)
7) travel-impulse-speed enterprise betazed ferenginar (10)
8) beam-up-supplies enterprise ferenginar dilithium1 warpdrive1 (1)
9) enable-warp-drive enterprise plasmaconduit1 plasmainjector1 warpcoil1 dilithium1 warpdrive1 (3)
10) travel-warp-speed enterprise ferenginar betazed warpdrive1 (2)
11) travel-warp-speed enterprise betazed qonos warpdrive1 (2)
12) travel-warp-speed enterprise qonos levinia warpdrive1 (100)
13) The total cost is 147, meet the time requirement to rescue the people.

With the predicate *warp-drive-ready*, we can check whether the *warp_drive* is ready yet, then the planner can use it to save steps to travel between the planets. So, the total plan cost 147, which met the time requirement.