CS 695: Programming assignment (P3)

Md. Ridwan Hossain Talukder
Department of Computer Science
George Mason University
Fairfax, USA
mtalukd@gmu.edu

I. Introduction to constraint-based Optimization

A. Implementing Some Example Programs

```
Program 1

Objective value:  7.0
Number of variables:  4
x1 == 0.0
x2 == 2.0
x3 == 1.0
x4 == 3.0
```

```
Program 2

Objective value:  36.0
Number of variables:  2
x1 == 2.0
x2 == 6.0
```

B. A word problem

1) Answer: There are two variables (number of days/week each mine should be operated) and three constraints (to fulfill the smelting plant contract). If we also consider the allowed range for days (number of days/week can not be more than 6 or less than 0) then there are four more constraints.

```
Implementation of the MILP

solver = pywraplp.Solver.CreateSolver('SCIP')

# 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)
```

```
Mine Operation Problem

Objective value:  680.0000000000001
Number of variables:  2
x1 == 2.0
x2 == 1.9999999999999996
```

II. Trajectory Optimization with MILPs

A. Trajectory Optimization

1) Answer: The objective value decreases as the number of steps increases. As we sample over more steps the agent has more flexibility over the control variables to optimize the objective function and so as the sampling steps increases it helps smoothing and shortening the trajectories and as a result the objective function gets better optimization. We would get the optimized value(almost) in case we had an infinite sampling.

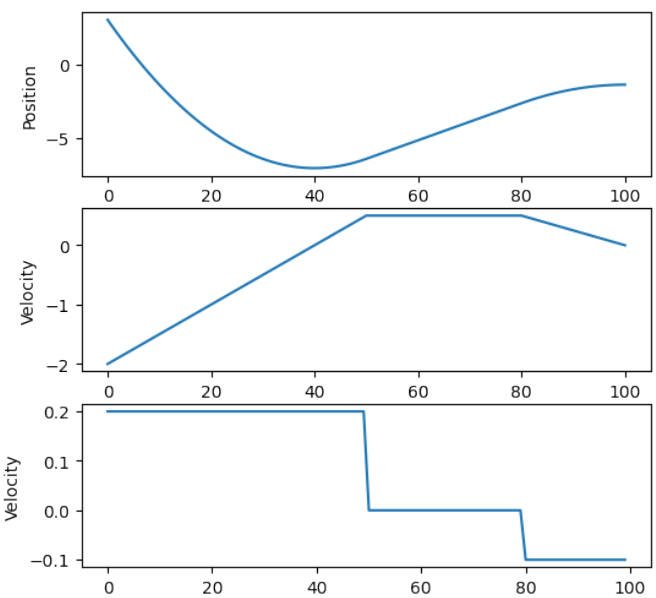
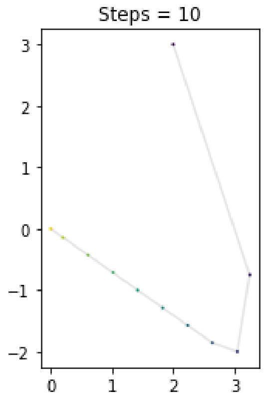


Fig. 1: The trajectory of a simple 1D space vehicle given known thrusts

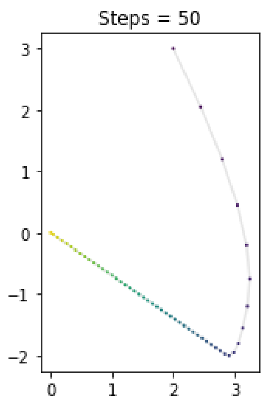
B. Adding an obstacle

```
Objective values (adding obstacles)

Objective value = 3.8461788617886286
Objective value = 0.5329970575872215
Objective value = 3.500582750582752
Objective value = 0.40816326530612257
```

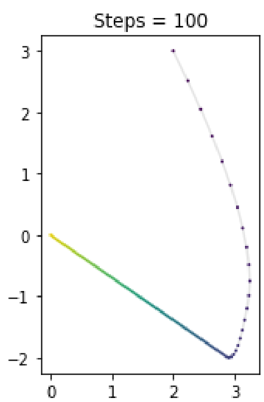
```
# Constraints on adding the obstacles

M = 10000
for ii, x in enumerate(xs[1:]):
    for obstacle in obstacles:
        solver.Add(x[0] <= obstacle[0] + M*b[ii][0])
        solver.Add(-x[0] <= -obstacle[1] + M*b[ii
    ][1])
        solver.Add(x[1] <= obstacle[2] + M*b[ii][2])
        solver.Add(-x[1] <= -obstacle[3] + M*b[ii
    ][3])
    solver.Add(b[ii][0] + b[ii][1] + b[ii][2] + b[ii
    ][3] <= 3)
```



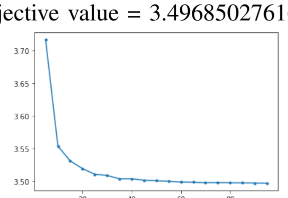
(a) Objective value = 3.553571428571429



(b) Objective value = 3.500582750582752

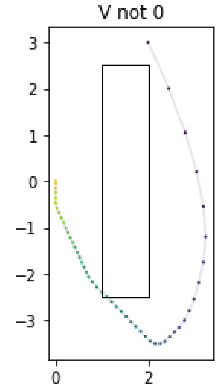


(c) Objective value = 3.4968502761606186

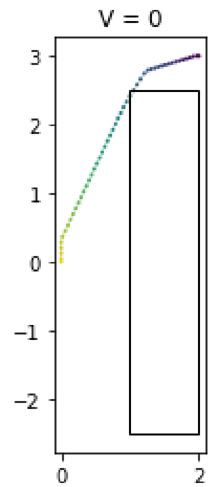


(d) Objective value comparison

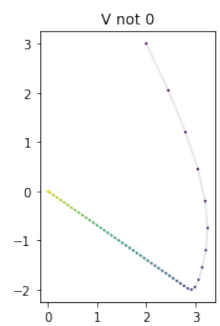
Fig. 2: Plots of Trajectory Optimization



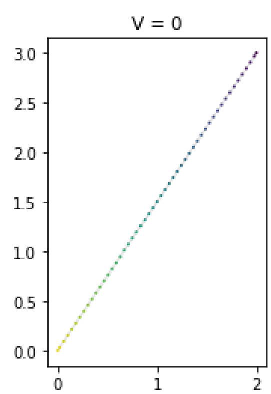
(a) Objective value = 3.8461788617886286



(b) Objective value = 0.5329970575872215



(c) Objective value = 3.500582750582752



(d) Objective value = 0.40816326530612257

Fig. 3: Plots of Trajectory Optimization (with obstacles)

III. PDDL

A. Running Fast Downward

TABLE I: Comparing astar and the ff heuristic

Heuristic	A*	FF
Completed Plans Length	34	48
No of Expanded States	1385559	106
Total Run Time (s)	9.22849	0.00645867

B. Answer

It really depends on what we want. A* heuristic is optimal hence admissible but FF heuristic is ”Satisficing” and we can see the evidence on the Table-I. From the values we can see that A* gives us the optimal plans length where the plans length generated by FF is a bit higher and is not optimal. But if we see the values for number of states expanded and total run time, FF heuristic outperforms A* in both cases. So while we are not navigating, rather planning eg: we want to know if a plan exists for a given problem or not (does not need to be an optimal plan) I’d prefer to use the FF heuristic. But if we want to execute the plan (navigation) in optimal lengths then we have to use astar heuristic. I have another observation, if the cost of expanding states is higher, in that case optimal plans may be costly than non-optimal plan, in that case too ff heuristic should perform better.

IV. An Example PDDL Problem

A. Without Warp-drive

When initially there is only one action $travel-impulse-speed$. The actions that minimize cost are following:

```
travel-impulse-speed enterprise earth vulcan (10)
travel-impulse-speed enterprise vulcan qonos (6)
travel-impulse-speed enterprise qonos levinia (500)
```

So in this case the total cost is 516. Which means we need 516 steps to get from earth to levinia.

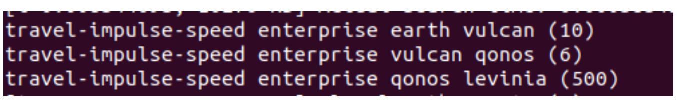

travel-impulse-speed enterprise earth vulcan (10)
travel-impulse-speed enterprise vulcan qonos (6)
travel-impulse-speed enterprise qonos levinia (500)

Fig. 4: Actions that minimize the total cost (without warp-drive)

B. With Warp-drive Fixed

Now we add three new actions, beam-up-supplies, enable-warp-drive and travel-warp-speed. Also we added two predicates onship and enable. When all the required things are onship we will enable-warp-drive and travel-with-warp-speed onwards. So with warp-drive fixed we get the plans length 147 which is less than 400 and Enterprise makes it to levinia in time.

```
(:predicates
  (enable)
  (onship ?p - supply ?s - ship)
  (at ?l - locatable ?p - location)
  (adjacent ?a - location ?b - location))
```

```
(:action travel-warp-speed
  :parameters (?s - ship ?from - location ?to -
    location)
  :precondition (and
    (enable)
    (at ?s ?from)
    (adjacent ?from ?to))
  :effect (and
    (not (at ?s ?from))
    (at ?s ?to)
    (increase (total-cost) (warp-distance ?from ?to)
    )
    ))

(:action beam-up-supplies
  :parameters (?s - ship ?l - location ?p - supply)
  :precondition (and
    (at ?s ?l)
    (at ?p ?l)
    (not (onship ?p ?s)))
  :effect (and
    (not (at ?p ?l))
    (onship ?p ?s)
    (increase (total-cost) 1)
    ))

(:action enable-warp-drive
  :parameters (?s - ship ?pc - plasmaconduit ?pi -
    plasmainjector ?wc - warpcoil ?di - dilithium)
  :precondition (and
    (onship ?pc ?s)
    (onship ?pi ?s)
    (onship ?wc ?s)
    (onship ?di ?s)
    (not (enable)))
  :effect (and
    (enable)
    (increase (total-cost) 3)
    ))
```

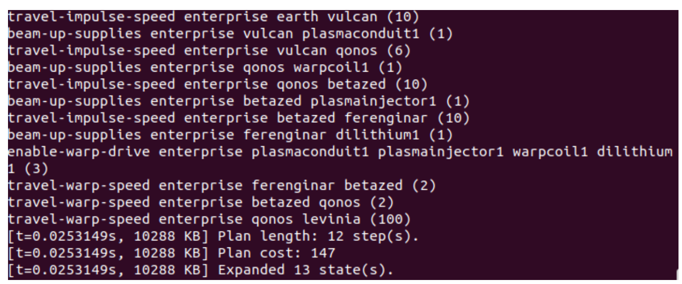

travel-impulse-speed enterprise earth vulcan (10)
beam-up-supplies enterprise vulcan plasmaconduit1 (1)
travel-impulse-speed enterprise vulcan qonos (6)
beam-up-supplies enterprise qonos warpcoil1 (1)
travel-impulse-speed enterprise qonos betazed (10)
beam-up-supplies enterprise betazed plasmainjector1 (1)
travel-impulse-speed enterprise betazed ferenginar (10)
beam-up-supplies enterprise ferenginar dilithium1 (1)
enable-warp-drive enterprise plasmaconduit1 plasmainjector1 warpcoil1 dilithium1 (3)
travel-warp-speed enterprise ferenginar betazed (2)
travel-warp-speed enterprise betazed qonos (2)
travel-warp-speed enterprise qonos levinia (100)
[t=0.0253149s, 10288 KB] Plan length: 12 step(s).
[t=0.0253149s, 10288 KB] Plan cost: 147
[t=0.0253149s, 10288 KB] Expanded 13 state(s).

Fig. 5: Actions that minimize the total cost (with warp-drive fixed)

```
travel-impulse-speed enterprise earth vulcan (10)
beam-up-supplies enterprise vulcan plasmaconduit1
    (1)
travel-impulse-speed enterprise vulcan qonos (6)
beam-up-supplies enterprise qonos warpcoil1 (1)
travel-impulse-speed enterprise qonos betazed (10)
beam-up-supplies enterprise betazed plasmainjector1
    (1)
travel-impulse-speed enterprise betazed ferenginar
    (10)
beam-up-supplies enterprise ferenginar dilithium1
    (1)
enable-warp-drive enterprise plasmaconduit1
    plasmainjector1 warpcoil1 dilithium1 (3)
travel-warp-speed enterprise ferenginar betazed (2)
travel-warp-speed enterprise betazed qonos (2)
travel-warp-speed enterprise qonos levinia (100)
```