Assignment 4 of Algorithm

Zuyao Chen 201728008629002
zychen.uestc@gmail.com

1. Linear-inequality feasibility

Proof:

suppose that we have an algorithm for linear programming, that means we can solve

$$\begin{aligned} \max \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$

given a linear inequality feasibility problem , our goal is to check whether there exist $\boldsymbol{x}^*$,

$$\boldsymbol{A}'\boldsymbol{x}^* \le \boldsymbol{b}', \boldsymbol{x}^* \ge \boldsymbol{0}$$

which equals to

$$\begin{aligned} \max \quad & \boldsymbol{0} \cdot \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}'\boldsymbol{x} \le \boldsymbol{b}' \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$

this problem can be solved using the same algorithm.

2. Airplane Landing Problem

Let $x_1, x_2, ..., x_n$ be the exact landing time of each airplane respectively, the problem can be written as

$$\begin{aligned} \max \quad & \min(x_2 - x_1, x_3 - x_2, ..., x_n - x_{n-1}) \\ \text{s.t.} \quad & s_i \le x_i \le t_i, i = 1, 2, \cdots, n \end{aligned}$$

LP form

$$\begin{aligned} \max \quad & d \\ \text{s.t.} \quad & s_i \le x_i \le t_i, i = 1, 2, \cdots, n \\ & x_k - x_{k-1} \ge d, k = 2, 3, \cdots, n \\ & d \ge 0 \end{aligned}$$

Now we show that how to obtain the dual form of this question, first we minimize $-d$,

$$\begin{aligned} \min \quad & z = -d \\ \text{s.t.} \quad & s_i \le x_i \le t_i, i = 1, 2, \cdots, n \\ & -x_{k+1} + x_k \le -d = z, k = 1, 2, \cdots, n-1 \\ & z \le 0 \end{aligned}$$

using Lagrange multiplier,

$$L(z, x, \lambda, \alpha, \varphi, \phi) = z + \sum_{i=1}^{n} \lambda_i (x_i - t_i) + \sum_{i=1}^{n} \alpha_i (-x_i + s_i) + \sum_{i=1}^{n-1} \varphi_i (-x_{i+1} + x_i - z) + \phi z$$

$$\begin{aligned}
\frac{\partial L}{\partial z} &= 1 - \sum_{i=1}^{n-1} \varphi_i + \phi = 0 \\
\frac{\partial L}{\partial x_1} &= \lambda_1 - \alpha_1 + \varphi_1 = 0 \\
\frac{\partial L}{\partial x_n} &= \lambda_n - \alpha_n - \varphi_{n-1} = 0 \\
\frac{\partial L}{\partial x_i} &= \lambda_i - \alpha_i + \varphi_i - \varphi_{i-1} = 0, i = 2, 3, \cdots, n-1
\end{aligned}$$

thus

$$\begin{aligned}
g(\lambda, \alpha, \varphi, \phi) &= \inf_{z,x} L(z, x, \lambda, \alpha, \varphi, \phi) \\
&= z(1 - \sum_{i=1}^{n} \varphi_i + \phi) + x_1(\lambda_1 - \alpha_1 + \varphi_1) + x_n(\lambda_n - \alpha_n - \varphi_{n-1}) - \sum_{i=1}^{n} \lambda_i t_i + \sum_{i=1}^{n} \alpha_i s_i \\
&= -\sum_{i=1}^{n} \lambda_i t_i + \sum_{i=1}^{n} \alpha_i s_i
\end{aligned}$$

Minimizing $-d$ is equivalent to

$$\begin{aligned}
\text{max} \quad & -\sum_{i=1}^{n} \lambda_i t_i + \sum_{i=1}^{n} \alpha_i s_i \\
\text{s.t.} \quad & \lambda_i \geq 0, \alpha \geq 0, i = 1, 2, \cdots, n \\
& \varphi_i \geq 0, i = 1, 2, \cdots, n-1 \\
& \lambda_1 - \alpha_1 + \varphi_1 = 0 \\
& \lambda_n - \alpha_n - \varphi_{n-1} = 0 \\
& \lambda_i - \alpha_i + \varphi_i - \varphi_{i-1} = 0, i = 2, 3, \cdots, n-1 \\
& 1 - \sum_{i=1}^{n-1} \varphi_i \leq 0
\end{aligned}$$

Hence the original problem that Maximizing d is equivalent to

$$\begin{aligned}
\text{min} \quad & \sum_{i=1}^{n} \lambda_i t_i - \sum_{i=1}^{n} \alpha_i s_i \\
\text{s.t.} \quad & \lambda_i \geq 0, \alpha_i \geq 0, i = 1, 2, \cdots, n \\
& \varphi_i \geq 0, i = 1, 2, \cdots, n-1 \\
& \lambda_1 - \alpha_1 + \varphi_1 = 0 \\
& \lambda_n - \alpha_n - \varphi_{n-1} = 0 \\
& \lambda_i - \alpha_i + \varphi_i - \varphi_{i-1} = 0, i = 2, 3, \cdots, n-1 \\
& 1 - \sum_{i=1}^{n-1} \varphi_i \leq 0
\end{aligned}$$

for instance, we have $n = 4, [10, 20], [40, 60], [75, 80], [100, 120]$ (here the minute is the metric of time), using tool cvxpy we can obtain the optimal solution 35 with optimal variables $x_1 = 10, 45, 80, 116$.

```python
#!/usr/bin/env python

import cvxpy as cvx
import numpy as np
import ecos

x = cvx.Variable(4,integer=True)
left = np.array([10,40,75,100])
right =np.array([20,60,80,120])

objective = cvx.Maximize(cvx.min(x[1:] - x[:3]))

constraints = [ x >= left, x <= right]

prob = cvx.Problem(objective,constraints)

prob.solve(solver = cvx.ECOS_BB)
print 'using solver ECOS_BB...'
print 'status:', prob.status
print 'optimal value:', prob.value
print 'optimal var:', x.value
```

```
joseph_chen@JosephChen-Desktop: ~/Documents/algorithms_course/python
joseph_chen@JosephChen-Desktop:~/Documents/algorithms_course/python$ ./landing.p
y
using solver ECOS_BB...
status: optimal
optimal value: 34.9999999999
optimal var: [  10.   45.   80.  116.]
joseph_chen@JosephChen-Desktop:~/Documents/algorithms_course/python$
```

3. Dual Simplex Algorithm

the result is showed below.

```
directly solving...
problem has 3 equalities and 0 inequalities
starting optimization...
status: feasible!
optimal value: -16.5
optimal x: [  0.    0.    0.    1.5   2.5  16.5]
solving dual problem...
problem has 0 equalities and 6 inequalities
starting optimization...
status: feasible!
optimal value: -16.5
optimal y: [ 2.5  6.   0. ]
using tool cvxpy
optimal value: -16.5
optimal x: [ -5.45737158e-22   6.11583402e-22  -4.40096362e-22   1.50000000e+00
   2.50000000e+00   1.65000000e+01]
```