Assignment 4
Algorithm Design and Analysis

bitjoy.net

November 27, 2015

I choose problem 1,2,4,7.

1 Linear-inequality feasibility

1.1 Linear programming => linear-inequality feasibility problem

Suppose we have an algorithm for linear programming:

$$\begin{array}{rr} \min & \mathbf{c^T x} \\ s.t. & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{1}$$

We just change the objective in (1) like this:

$$\begin{array}{rr} \min & \mathbf{0} \\ s.t. & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{2}$$

and run linear programming again. If (2) has optimal solution, then linear-inequality

$$\begin{array}{r} \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{array} \tag{3}$$

is feasible, otherwise infeasible. (1) only costs polynomial time.

1.2 Linear-inequality feasibility => linear programming

Suppose we can get the feasible solution of primal problem and its dual problem, say a and b, but neither is optimal(c). As we know, the optimal solution of primal problem must be between these two values, say a>c>b. Then we get the median(m) of a and b, and add an inequality obj>m. If new inequality is feasible, then optimal should be [m,a], otherwise [b,m]. We check feasibility of new inequality recursively until the interval is small enough.

2 Airplane Landing Problem

Let $x_1, x_2, ..., x_n$ be the exact landing time of each airplane, the LP formulation of this problem should be:

$$\begin{array}{rll} \max & \min_{i=2...n}(x_i - x_{i-1}) & \\ s.t. & s_i \le x_i \le t_i & \text{for } i \text{ in 1...n} \end{array} \tag{4}$$

According to Robert Fourer's book *Optimization Models*[1], (4) is equivalent to

$$\begin{array}{rll} \max & z & \\ s.t. & z \le x_i - x_{i-1} & \text{for } i \text{ in 2...n} \\ & s_i \le x_i \le t_i & \text{for } i \text{ in 1...n} \end{array} \tag{5}$$

For the example in problem description, if the time window of landing three airplanes are [1,60], [80,100] and [120,140], use GLPK to solve it:

```
var x1 >= 1, <=60;
var x2 >= 80, <= 100;
var x3 >=120, <=140;
var z;
maximize gap: z;
s.t. gap1: z<=x2-x1;
s.t. gap2: z<=x3-x2;
end;
```

The optimal result is $z = 60$ and $x_1 = 1, x_2 = 80, x_3 = 140$. Thus, they land at 10:00, 11:20, 12:20 respectively.

4 Gas Station Placement

Let $x_1, x_2, ..., x_n$ be the place of each gas station, as $d_1, d_2, ..., d_n$ and r have been given, we can get the LP formulation like this:

$$\begin{array}{rll} \min & \max_{i=2...n}(x_i - x_{i-1}) & \\ s.t. & |x_i - d_i| \le r & \text{for } i \text{ in 1...n} \end{array} \tag{6}$$

Similarly, (6) is equivalent to

$$\begin{array}{rll} \min & z & \\ s.t. & z \ge x_i - x_{i-1} & \text{for } i \text{ in 2...n} \\ & |x_i - d_i| \le r & \text{for } i \text{ in 1...n} \end{array} \tag{7}$$

7 Simplex Algorithm

Consider the following linear program in standard form:

$$\begin{array}{rr} \max & \mathbf{c^T x} \\ s.t. & \mathbf{Ax} \le \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \end{array} \tag{8}$$

I have implemented the Simplex Algorithm in Python 3 according to Chapter 29 of *Introduction to Algorithms*:

[1] http://www.4er.org/CourseNotes/Book%20A/A-III.pdf

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 26 18:44:28 2015

@author: czl
"""

import numpy as np

class SIMPLEX:
    m = 0
    n = 0

    def __init__(self):
        pass

    def INITIALIZE(self, A, b, c):
        k = b.argmin()
        if b[k] >= 0: # Note, I only implemented the easy case.
            AA = np.zeros((self.m + self.n + 1, self.m + self.n + 1))
            bb = np.array([0.0] * (self.m + self.n + 1)) # 0.0 for float64
            cc = np.array([0.0] * (self.m + self.n + 1)) # 0.0 for float64
            AA[self.n + 1 : self.n + self.m + 1, 1 : self.n + 1] = A
            bb[self.n + 1 : self.n + self.m + 1] = b
            cc[1 : self.n + 1] = c
            return(np.arange(1, self.n + 1, 1), np.arange(self.n + 1, self.n + self.m + 1, 1), AA, bb, cc, 0)

    def PIVOT(self, N, B, A, b, c, v, l, e):
        AA = np.zeros((self.m + self.n + 1, self.m + self.n + 1))
        b[e] = b[l] / A[l][e]
        for j in N:
            if j != e:
                AA[e][j] = A[l][j] / A[l][e]
        AA[e][l] = 1 / A[l][e]
        for i in B:
            if i != l:
                b[i] = b[i] - A[i][e] * b[e]
                for j in N:
                    if j != e:
                        AA[i][j] = A[i][j] - A[i][e] * AA[e][j]
                AA[i][l] = - A[i][e] * AA[e][l]
        v = v + c[e] * b[e]
        for j in N:
            if j != e:
                c[j] = c[j] - c[e] * AA[e][j]
        c[l] = - c[e] * AA[e][l]
        c[e] = 0 # clear c of enter
        b[l] = 0 # clear b of leave
        NN = np.delete(N, np.where( N == e)[0][0])
        NN = np.append(NN, l)
        BB = np.delete(B, np.where( B == l)[0][0])
        BB = np.append(BB, e)
        return(NN, BB, AA, b, c, v)


    def SOLVE(self, A, b, c):
        self.m, self.n = A.shape
        N, B, A, b, c, v = self.INITIALIZE(A, b, c)
```

```
        while c.max() > 0:
            d = np.array([float('inf')] * (self.m + self.n + 1))
            e = c.argmax() # choose index of max c
            for i in B:
                if A[i][e] > 0:
                    d[i] = b[i] / A[i][e]
            l = d.argmin()
            if d[l] == float('inf'):
                return -2 # unbounded
            else:
                N, B, A, b, c, v = self.PIVOT(N, B, A, b, c, v, l, e)
        x = np.array([0] * self.n)
        for i in range(self.n):
            if i + 1 in B:
                x[i] = b[i + 1]
        return x


if __name__ == "__main__":
    A = np.array([[1,1,3],
                  [2,2,5],
                  [4,1,2]])
    b = np.array([30,24,36])
    c = np.array([3,1,2])
    s = SIMPLEX()
    x = s.SOLVE(A, b, c)
```

As I was fully occupied with lots of things, I only implemented the easy case in finding an initial solution, see function **INITIALIZE**. But I will finish another case in the future.

Here is a test example:

$$\begin{aligned} \max \quad & 3x_1 + x_2 + 2x_3 \\ s.t. \quad & x_1 + x_2 + 3x_3 \leq 30 \\ & 2x_1 + 2x_2 + 5x_3 \leq 24 \\ & 4x_1 + x_2 + 2x_3 \leq 36 \\ & x_1, x_2, x_3 \geq 0 \end{aligned} \tag{9}$$

We have:

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix} \qquad b = \begin{bmatrix} 30 \\ 24 \\ 36 \end{bmatrix} \qquad c = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$$

After running my implementation, we get:

$$x = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}$$

The optimal solution is $z = 3x_1 + x_2 + 2x_3 = 28$, the result is the same as GLPK's.