Operations Research, Homework 2

B09705039 LIU, WEI-EN

1 Problem 1

1.1 (a)

Standard Form:

$$\begin{aligned} \min \quad & x_1 + 2x_2 + 3x_3 \\ \text{s.t.} \quad & x_1 + x_2 - x_4 = 4 \\ & x_1 + x_2 + x_3 + x_5 = 9 \\ & x_3 - x_6 = 3 \\ & x_i \geq 0 \quad \forall i \in \{1, ..., 6\} \end{aligned} \tag{1}$$

1.2 (b)

Basis	Bfs	Basic solutions
(x_1, x_3, x_4)	Yes	$(6, 0, 3, 2, 0, 0)$
(x_1, x_3, x_5)	Yes	$(4, 0, 3, 0, 2, 0)$
(x_1, x_3, x_6)	Yes	$(4, 0, 5, 0, 0, 2)$
(x_1, x_4, x_6)	No	$(9, 0, 0, 5, 0, -3)$
(x_1, x_5, x_6)	No	$(4, 0, 0, 0, 5, -3)$
(x_2, x_3, x_4)	Yes	$(0, 6, 3, 2, 0, 0)$
(x_2, x_3, x_5)	Yes	$(0, 4, 3, 0, 2, 0)$
(x_2, x_3, x_6)	Yes	$(0, 4, 5, 0, 0, 2)$
(x_2, x_4, x_6)	No	$(0, 9, 0, 5, 0, -3)$
(x_2, x_5, x_6)	No	$(0, 4, 0, 0, 5, -3)$
(x_3, x_4, x_5)	No	$(0, 0, 3, -4, 6, 0)$
(x_3, x_4, x_6)	No	$(0, 0, 9, -4, 0, 6)$
(x_4, x_5, x_6)	No	$(0, 0, 0, -4, 9, -3)$

Table 1: Table of basic solutions $(x_1, x_2, x_3, x_4, x_5, x_6)$ and basic feasible solutions(Bfs).

1.3 (c)

6 extreme points (x_1, x_2, x_3):

(6, 0, 3), (4, 0, 3), (4, 0, 5), (0, 6, 3), (0, 4, 3), (0, 4, 5)

1.4 (d)

Phase 1 LP:

$$\begin{aligned} \min \quad & x_7 + x_8 \\ \text{s.t.} \quad & x_1 + x_2 - x_4 + x_7 = 4 \\ & x_1 + x_2 + x_3 + x_5 = 9 \\ & x_3 - x_6 + x_8 = 3 \\ & x_i \geq 0 \quad \forall i \in \{1, ..., 8\} \end{aligned} \tag{2}$$

0	0	0	0	0	0	-1	-1	0
1	1	0	-1	0	0	1	0	$x_7 = 4$
1	1	1	0	1	0	0	0	$x_5 = 9$
0	0	1	0	0	-1	0	1	$x_8 = 3$

$\xrightarrow{\text{adjust}}$

1	1	1	-1	0	-1	0	0	7
[1]	1	0	-1	0	0	1	0	$x_7 = 4$
1	1	1	0	1	0	0	0	$x_5 = 9$
0	0	1	0	0	-1	0	1	$x_8 = 3$

$\rightarrow$

0	0	1	0	0	-1	0	3
1	1	0	-1	0	0	0	$x_1 = 4$
0	0	1	-1	1	0	0	$x_5 = 5$
0	0	[1]	0	0	-1	1	$x_8 = 3$

$\rightarrow$

0	0	0	0	0	0	0
1	1	0	-1	0	0	$x_1 = 4$
0	0	0	-1	1	1	$x_5 = 2$
0	0	1	0	0	-1	$x_3 = 3$

Phase 2 LP:

-1	-2	-3	0	0	0	0
1	1	0	-1	0	0	$x_1 = 4$
0	0	0	-1	1	1	$x_5 = 2$
0	0	1	0	0	-1	$x_3 = 3$

$\xrightarrow{\text{adjust}}$

0	-1	-3	-1	0	0	4
1	1	0	-1	0	0	$x_1 = 4$
0	0	0	-1	1	1	$x_5 = 2$
0	0	1	0	0	-1	$x_3 = 3$

$\xrightarrow{\text{adjust}}$

0	-1	0	-1	0	-3	13
1	1	0	-1	0	0	$x_1 = 4$
0	0	0	-1	1	1	$x_5 = 2$
0	0	1	0	0	-1	$x_3 = 3$

The standard form LP:
Optimal solution = $(x_1, x_2, x_3, x_4, x_5, x_6)$ = (4, 0, 3, 0, 2, 0)
Objective value = 13

The original LP:
Optimal solution = (x_1, x_2, x_3) = (4, 0, 3)
Objective value = 13

In the minimization problem an iteration has no improvement when all the coefficient of it's objective function is ≤ 0. When this situation happens, we skip these iterations. Thus, there isn't any iteration presented above that has no improvements.

2 Problem 2

2.1 (a)

Linear Relaxation:

$$\begin{aligned} \max \quad & 2x_1 + 2x_2 + 5x_3 + 11x_4 + 10x_5 \\ \text{s.t.} \quad & x_1 + 4x_2 + 3x_3 + 5x_4 + 3x_5 \leq 20 \\ & x_i \geq 0 \quad \forall i \in \{1, ..., 5\} \end{aligned} \tag{3}$$

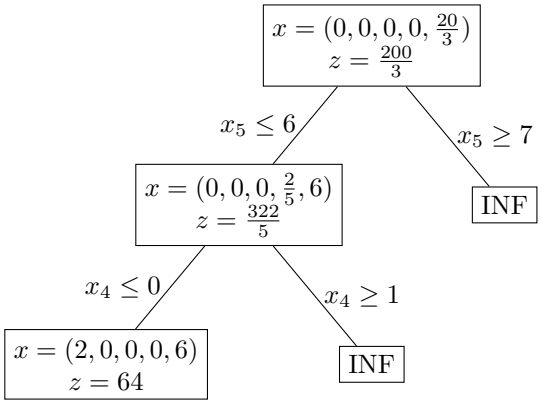
Ratio for greedy(objective function coefficient / constraint 1 coefficient): $x_5 > x_4 > x_1 > x_3 > x_2$

LR optimal solution: $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 0, 0, \frac{20}{3})$

objective value = $\frac{200}{3}$

2.2 (b)

Branch-and-bound tree:



optimal solution $(x_1, x_2, x_3, x_4, x_5) = (2, 0, 0, 0, 6)$

objective value = 64

3 Problem 3

3.1 (a)

Ranges:

$N = \{1, \ldots, n\}$ as range of jobs.

$M = \{1, \ldots, m\}$ as range of machines.

Coefficients:

p_j is the processing time(hr) of job j, $j \in J$.

b_j is the benefit of job j, $j \in J$.

Variables:

y represent the total benefit of the least total benefit machine.

x_{mj} is 1 if job j is scheduled to machine m, 0 otherwise.

$$\begin{aligned} \max \quad & y \\ \text{s.t.} \quad & y \leq \sum_{j \in N} x_{mj} b_j \quad \forall m \in M \\ & \sum_{m \in M} x_{mj} \leq 1 \quad \forall j \in N \\ & \sum_{jinN} x_{mj} p_j \leq K \quad \forall m \in M \\ & x_{mj} = \{0, 1\} \quad \forall m \in M, j \in N \end{aligned} \tag{4}$$

3.2 (b)

Linear relaxation:

x_{mj} is the proportion of job j scheduled to machine m.

Others are same as (a).

$$\begin{aligned} \max \quad & y \\ \text{s.t.} \quad & y \leq \sum_{j \in N} x_{mj} b_j \quad \forall m \in M \\ & \sum_{m \in M} x_{mj} \leq 1 \quad \forall j \in N \\ & \sum_{jinN} x_{mj} p_j \leq K \quad \forall m \in M \\ & x_{mj} = [0, 1] \quad \forall m \in M, j \in N \end{aligned} \tag{5}$$

The linear relaxation optimal solution is to schedule the whole job $1, 3$ to machine 1, job $7, 10$ to machine 2, job $4, 5$ to machine 3, do not schedule job 9. Split job 2 into $(0.8333, 0.119, 0.0476)$, job 6 into $(0.25, 0.0, 0.75)$, job 8 into $(0.0, 0.2738, 0.3095)$ and schedule them into machine 1 to 3 respectively.

The objective value is 17.416666666666668.

It is an upper bound of the objective value of an IP-optimal solution, because it maximizes the objective function and according to the proposition discussed in class the linear relaxation objective value is lager than a maximization IP objective value.

Python Code:

```
# Import libraries
from gurobipy import *
import pandas as pd

# Input
m = 3
n = 10
b = [5, 8, 4, 6, 3, 7, 6, 9, 5, 8]
p = [3, 6, 5, 4, 1, 8, 5, 12, 7, 6]
K = 15

N = range(n)
M = range(m)

# Modeling
# add var
p3_1 = Model("problem3-1")      # build a new model

x = p3_1.addVars(M, N, lb = 0, ub = 1, vtype = GRB.CONTINUOUS, name = "x"
    )
y = p3_1.addVar(lb = 0, vtype = GRB.CONTINUOUS, name = "y")

# setting the objective function
p3_1.setObjective(
    y
    , GRB.MAXIMIZE)

# add constraints and name them
for i in M:
    p3_1.addConstr(y <= (quicksum(x[i, j] * b[j] for j in N)), name = f"
        min benefit")

for j in N:
    p3_1.addConstr(quicksum(x[i, j] for i in M) <= 1, name = f"distribute
        to no more than 1 machine")

for i in M:
    p3_1.addConstr(quicksum(x[i, j] * p[j] for j in N) <= K, name = f"
        satisfy capacity")

p3_1.optimize()

# Results
print("Results:\n")

# objective value
LR_ov = p3_1.objVal
print("objective value =", LR_ov)
print("")

# x
print("x:")
print("J \ M", end="")
print("\tMachine1 \tMachine2 \tMachine3")
for j in N:
    print("Job" + str(j+1), "\t", end="")
```

```
    for i in M:
        print(round((x[i, j].x), 4), "\t\t", end="")
    print("")
print("")

# y
print("y:")
print(y.x)
```

3.3 (c)

The optimal solution is to schedule job $10, 2$ to machine 1, job $20, 11$ to machine 2, job $3, 14, 15$ to machine 3, job $19, 25, 6$ to machine 4 and don't schedule the rest of the jobs.

Then we will get the optimal objective value (machine earning the least benefit) 23.

Machine 1 earns 23, machine 2 earns 23, machine 3 earns 23, machine 4 earns 28 benefits.

The percentage optimality gap with respect to the bound obtained by linear relaxation is 0.2432432432432433 (about 24.3%).

Python Code:

```
# Sorting fuctions
def B_bubbleSort(b, job):
    for i in N:
        for j in range(0, n-i-1):
            if b[job[j]] < b[job[j+1]]:
                job[j], job[j+1] = job[j+1], job[j]

def M_bubbleSort(bm, machine):
    for i in M:
        for j in range(0, m-i-1):
            if bm[machine[j]] > bm[machine[j+1]]:
                machine[j], machine[j+1] = machine[j+1], machine[j]

# Input
m = 4
n = 25
b = [5,8,12,4,6,6,3,7,6,15,9,5,8,10,1,5,3,7,12,14,5,8,9,8,10]
p = [3,6,7,5,4,2,6,3,5,8,10,2,4,7,1,5,8,3,6,4,12,4,8,4,7]
K = 15

N = range(n)
M = range(m)

# Solve
# Step 1: Sorting
job_sorted = []
for i in N:
    job_sorted.append(i)

B_bubbleSort(b, job_sorted)

# Step 2: Scheduling
job_to_machine = [0] * n
Pm = [0] * m
```

```
Bm = [0] * m

for i in job_sorted:
    # sort machine in priority
    machine_sorted = []
    for k in M:
        machine_sorted.append(k)
    M_bubbleSort(Bm, machine_sorted)

    # schedule jods to machine
    scheduled = False
    for j in machine_sorted:
        if (Pm[j] + p[i] <= K):
            job_to_machine[i] = j
            Pm[j] += p[i]
            Bm[j] += b[i]
            scheduled = True
            break
    if (scheduled == False):
        job_to_machine[i] = -1

# print results (machine number = 0 in the display part if it is not
    scheduled)
for i in job_sorted:
    print("job", i + 1, ", machine", job_to_machine[i] + 1)
display("Accumulated processing time:", Pm)
display("Accumulated benefits:", Bm)
HBF_ov = min(Bm)
print("objective value:", HBF_ov)
print("HBF optimality gap:", (LR_ov - HBF_ov) / LR_ov)
```

3.4 (d)

The optimal solution is to schedule job $20, 24, 3$ to machine 1, job $6, 13, 10$ to machine 2, job $12, 18, 22, 5$ to machine 3, job $8, 19, 1, 15$ to machine 4 and don't schedule the rest of the jobs.

Then we will get the optimal objective value (machine earning the least benefit) 25.

Machine 1 earns 34, machine 2 earns 29, machine 3 earns 26, machine 4 earns 25 benefits.

The percentage optimality gap with respect to the bound obtained by linear relaxation is 0.17743830787309056 (about 17.7%).

Python Code:

```
# Sorting fuctions
def R_bubbleSort(b, job):
    for i in N:
        for j in range(0, n-i-1):
            if (b[job[j]]/p[job[j]]) < (b[job[j+1]]/p[job[j+1]]):
                job[j], job[j+1] = job[j+1], job[j]

def M_bubbleSort(bm, machine):
    for i in M:
        for j in range(0, m-i-1):
            if bm[machine[j]] > bm[machine[j+1]]:
                machine[j], machine[j+1] = machine[j+1], machine[j]
```

```
# Solve
# Step 1: Sorting
job_sorted = []
for i in N:
    job_sorted.append(i)

R_bubbleSort(b, job_sorted)

# Step 2: Scheduling
job_to_machine = [0] * n
Pm = [0] * m
Bm = [0] * m

for i in job_sorted:
    # sort machine in priority
    machine_sorted = []
    for k in M:
        machine_sorted.append(k)
    M_bubbleSort(Bm, machine_sorted)

    # schedule jods to machine
    scheduled = False
    for j in machine_sorted:
        if (Pm[j] + p[i] <= K):
            job_to_machine[i] = j
            Pm[j] += p[i]
            Bm[j] += b[i]
            scheduled = True
            break
    if (scheduled == False):
        job_to_machine[i] = -1

# print results (machine number = 0 in the display part if it is not
    scheduled)
for i in job_sorted:
    print("job", i + 1, ", machine", job_to_machine[i] + 1)
display("Accumulated processing time:", Pm)
display("Accumulated benefits:", Bm)
HRF_ov = min(Bm)
print("objective value:", HRF_ov)
print("HRF optimality gap:", (LR_ov - HRF_ov) / LR_ov)
```

4 Problem 4

4.1 (a)

Instance	HBF	HRF	Self define(HRBF)
1	0.0184	0.0184	0.0116
2	0.0659	0.0344	0.0344
3	0.0863	0.068	0.068
4	0.058	0.0303	0.0303
5	0.1809	0.0806	0.0806
6	0.0655	0.0396	0.0309
7	0.0949	0.0353	0.0472
8	0.0859	0.0728	0.0728
9	0.0671	0.0671	0.0671
10	0.0101	0.0173	0.0173
11	0.1718	0.0535	0.0535
12	0.0565	0.0307	0.0242
13	0.0756	0.1092	0.1261
14	0.1069	0.0611	0.0611
15	0.0686	0.0599	0.0427
16	0.0583	0.0743	0.0743
17	0.0116	0.0665	0.0665
18	0.1332	0.0648	0.0648
19	0.0387	0.1099	0.1099
20	0.102	0.0694	0.0694
average	0.0778	0.0581	0.0576

Table 2: Table for optimality gaps of the three algorithms.

Comment of the performance of the two heuristic algorithms:

According to the table 2 above, the average optimality gap of HBF is 0.0778, HRF is 0.0581, $0.0778 > 0.0581$, which implies that HRF is closer to the bound obtained by linear relaxation. Thus, HRF is a better heuristic algorithm compared with HBF.

4.2 (b)

(1) Inspired by the above two algorithms, I had an idea of gaining advantages of each algorithm by combining both. Since HRF is better than HBF in the previous problem, I started to sort the jobs by ratio first, and then sort jobs that have the same ratio with benefit first. By combining the benefits of both methods, I expected this self defined algorithm to have a slightly better solution than the original HRF algorithm.

(2) The results are in the last column (Self define(HRBF)) in table 2 above. It performed as I expected, the average optimality gap is slightly better than the HRF algorithm and it is the best among all heuristic algorithm in the table.

Python Code:

```
# Sorting fuctions
def RB_bubbleSort(b, job):
    for i in N:
        for j in range(0, n-i-1):
            if (b[job[j]]/p[job[j]]) < (b[job[j+1]]/p[job[j+1]]):
                job[j], job[j+1] = job[j+1], job[j]
    for i in N:
        for j in range(0, n-i-1):
```

```
            if (b[job[j]]/p[job[j]]) == (b[job[j+1]]/p[job[j+1]]) and (b[
                job[j]] < b[job[j+1]]):
                job[j], job[j+1] = job[j+1], job[j]

def M_bubbleSort(bm, machine):
    for i in M:
        for j in range(0, m-i-1):
            if bm[machine[j]] > bm[machine[j+1]]:
                machine[j], machine[j+1] = machine[j+1], machine[j]

# file input ('in01.txt' for example)
f = open('in01.txt', 'r')
line1 = f.readline()
line2 = f.readline()
line3 = f.readline()

array = line1.split( )
array = list(map(int, array))

m = array[0]
n = array[1]
K = array[2]

b = line2.split( )
b = list(map(int, b))

p = line3.split( )
p = list(map(int, p))

N = range(n)
M = range(m)
f.close()

# Solve
# Step 1: Sorting
job_sorted = []
for i in N:
    job_sorted.append(i)

RB_bubbleSort(b, job_sorted)
'''
for i in job_sorted:
    print(i, b[i])
print("")
'''


# Step 2: Scheduling
job_to_machine = [0] * n
Pm = [0] * m
Bm = [0] * m

for i in job_sorted:
    # sort machine in priority
    machine_sorted = []
    for k in M:
        machine_sorted.append(k)
```

```
    M_bubbleSort(Bm, machine_sorted)

    # schedule jods to machine
    scheduled = False
    for j in machine_sorted:
        if (Pm[j] + p[i] <= K):
            job_to_machine[i] = j
            Pm[j] += p[i]
            Bm[j] += b[i]
            scheduled = True
            break
    if (scheduled == False):
        job_to_machine[i] = -1

# print results (machine number = 0 in the display part if it is not
    scheduled)
for i in job_sorted:
    print("job", i + 1, ", machine", job_to_machine[i] + 1)
display("Accumulated processing time:", Pm)
display("Accumulated benefits:", Bm)
HRBF_ov = min(Bm)
print("objective value:", HRBF_ov)
print("HRBF optimality gap:", (LR_ov - HRBF_ov) / LR_ov)
```

5 Problem 5

$i_k = (x_1, x_2)$ of iteration k.

$f(i_k) = 3x_1^2 + 2x_2^2 + 4x_1x_2 + 6e^{x_1} + x_2$

Gradient(G(i_k)) =

$$\begin{bmatrix} 6x_1 + 4x_2 + 6e^{x_1} \\ 4x_2 + 4x_1 + 1 \end{bmatrix}$$

Hessian matrix(H(i_k)) =

$$\begin{bmatrix} 6 + 6e^{x_1} & 4 \\ 4 & 4 \end{bmatrix}$$

5.1 (a)

Iteration 0:

$$i_0 = (x_1, x_2) = (0, 0), f(i_0) = 6, G(i_0) = \begin{bmatrix} 6 \\ 1 \end{bmatrix}$$

Iteration 1:

$a_0 = argmin_{a \geq 0} f(i_0 - aG(i_0)) = argmin_{a \geq 0} 134a^2 - a + 6e^{-6a} \Rightarrow a = 0.0845924274342, a_0 = 4.48609$

i_1 = (-0.5075545646052, -0.0845924274342), objective value = 4.48609

5.2 (b)

Iteration 0:

$$i_0 = (x_1, x_2) = (0, 0), G(i_0) = \begin{bmatrix} 6 \\ 1 \end{bmatrix}, H(i_0) = \begin{bmatrix} 12 & 4 \\ 4 & 4 \end{bmatrix}$$

Iteration 1:

$$G(i_0) + H(i_0)(i_1 - i_0) = 0 \Rightarrow i_1 = i_0 - [H(i_0)]^{-1} G(i_0)$$

$$\Rightarrow i_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \frac{1}{8} \begin{bmatrix} 1 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 6 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{5}{8} \\ \frac{3}{8} \end{bmatrix}$$

$i_1 = (-\frac{5}{8}, \frac{3}{8})$, objective value = 4.102193571113942