Housing Maintenance

Zachary Ho, Irin Phatraprasit, Emma Xie, and Ava Zhu

COLUMBIA UNIVERSITY FACILITIES | HOUSING

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science
Industrial Engineering & Operations Research

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Our Goal

- Background
  - Students can request for housing maintenance by filling in an online form, calling, or emailing housing
  - Jobs are to be completed within 3-14 days
  - No current scheduling rules

- Objective
  - Reduce maintenance request completion time to below 9 days and improve student life at Columbia University
  - Minimize the average time a student has to wait for the job to be complete

COLUMBIA UNIVERSITY
FACILITIES | HOUSING

Simulation

```
import math
import random
import numpy as np


def poisson(a):

    x = 0
    p = 1

    u = random.random()
    p *= u

    while p >= math.e**(-1*a):
        x += 1
        u = random.random()
        p = u*p

    return x

def exp(g):

    U = random.random()
    R = (-1/g)*np.log(U)

    return R
```

```
def pos(l,t):

    n = poisson(l*t)
    numbers = []
    severity = []
    service = []

    type = 0

    if n == 0:
        print("There are no arrivals by time T")

    else:
        for i in range(1,n):
            u = random.random()
            numbers.append(t*u)

            p = random.random()
            if p < .6:
                type = 3
            elif .6 <= p < .9:
                type = 2
            else:
                type = 1
            severity.append(type)

            service_time = 0

            if type == 3:
                service_time = exp(3)
            elif type == 2:
                service_time = exp(9)
            else:
                service_time = exp(14)

            service.append(service_time)


        numbers.sort()

        return numbers, severity, service
```

Formulation:

1. Grouped maintenance types into 3 severities - high(1), medium(2), low(3)
2. Assumptions:
   - Low severity job occurs with the highest probability
   - Service time for high, medium, and low severity is 3, 9, 14 days respectively
   - Poisson arrivals and exponential service times
   - Limiting scope to 1 residence hall
   - Recalculate schedule every time a new ticket is submitted

Generated Data

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Python output:

```
>>> pos(10,1)
([0.006861617329770642, 0.09350265140190561, 0.1780331402163582, 0.1940561619152
5743, 0.24880917028124128, 0.25799123721563977, 0.2771429728737147, 0.3221177688
0730303, 0.5323279318786958, 0.5408393126874096, 0.5533195042341313, 0.589551592
8438075, 0.6476539331859792, 0.7775375606245014, 0.810619127473538, 0.8214848448
025609, 0.9662560682906132], [2, 2, 2, 2, 3, 3, 3, 3, 2, 3, 2, 1, 2, 2, 3, 3, 3]
, [2.148472330298465, 1.4588176324556932, 0.0889729597522712 77, 0.30091664043344
69, 1.0228848120249119, 0.51148791266175364, 0.95542570855340603, 2.449468542882
57, 0.71380541191559566, 1.9726270791893936, 0.53189845553789539, 1.947456550826
6064, 1.153967693622423, 0.36483854494849643, 1.7393359157795325, 1.039295806239
0858, 0.50422698580331859])
```

Problem we are trying to solve:

Objective Function: 3 | rj | □wjCj

- ➔ NP-Hard
- ➔ Assume machines are number of workers: m = 3
- ➔ **rj** - the time when ticket is submitted
- ➔ **wj** - the severity of request
- ➔ **pj** - processing time (service time)

j	rj	wj	pj	wj/pj
1	0.01	2	2.15	0.93
2	0.09	2	1.46	1.37
3	0.18	2	0.09	22.22
4	0.19	2	0.3	6.67
5	0.25	3	1.02	2.94
6	0.26	3	0.51	5.88
7	0.28	3	0.96	3.13
8	0.32	3	2.45	1.22
9	0.53	2	0.71	2.82
10	0.54	3	1.97	1.52
11	0.55	2	0.53	3.77
12	0.59	1	1.95	0.51
13	0.65	2	1.15	1.74
14	0.78	2	0.36	5.56
15	0.81	3	1.74	1.72
16	0.82	3	1.04	2.88
17	0.97	3	0.5	6.00

Data snapshot at time t

COLUMBIA UNIVERSITY | FACILITIES | HOUSING

Method 1 - Smith's Rule

Algorithm:

Step 1: Construct non-preemptive schedule using Smith's rule, wj/pj decreasing

Step 2: Assign jobs to machine that has completed task

Step 3: Ensure that job does not start before release date

Results:

- ➔ **Average wait time (Cj - rj) = 2.67**
- ➔ wjCj = 124.39

Schedule:

Method 2 - CONVERT

According to the algorithm CONVERT, the 1-machine model works for multiple parallel machines.

Step 1: Construct pre-emptive schedule on single machine

- ➔ Algorithm by **Smith**: At any time we schedule the job j with the greatest *Smith ratio* w_j/p_j, where p_j is the remaining processing time of job j.

Step 2: Form a list L of the jobs, ordered by their pre-emptive completion times.

Step 3: List schedule non-preemptively using L, with the constraint that no job J starts before its release date rj.

- ➔ To slightly simplify our analysis, we specify that our list scheduling algorithm will not start the jth job in L until the (j - 1)st is completed (in the one machine environment) or started (in the parallel machine environment)

*There is an on-line non-preemptive 2-approximation algorithm for 1 | rj | □Cj

Given a schedule P for 1 | rj, pmtn | □wjCj, algorithm CONVERT produces a schedule N for the corresponding instance of 1 | rj l □wjCj with C_N ≤ 2C_P. *

Algorithm 2 - CONVERT

Results:

- ➔ Average wait time (Cj - rj) = 2.62
- ➔ wjCj = 126.99

Results and Recommendations

- Both algorithms delivered an average waiting time lower than the range provided by housing
- Smith's Algorithm and CONVERT Algorithms produce close results
- After various trials, CONVERT Algorithm was shown to have slight improvement over Smith's Algorithm

- Next Steps:
  - Run algorithm with real data
  - Explore other types of algorithms e.g. Maintenance Scheduling