Scheduling

Stochastic Simulations

Hy Nguyen
Tyler Tsang
Haohan Xu
Jake Poliner

Outline

01. Stochastic Simulation

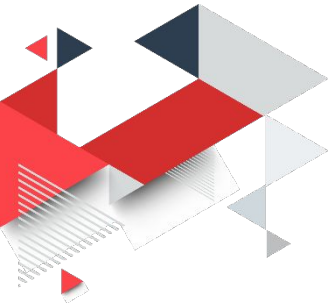
02. Minimize Weighted Makespan

03. Minimize Tardiness

04. Minimize Weighted Tardiness

Objective

- Throughout the class we have primarily focused on deterministic scheduling problems
- These can be hard (or even NP hard) to solve for optimal schedules
- Test different stochastic situations to see *how* optimal schedule changes when there is randomness involved
  - Normal and exponential distributions
  - Varying variances



Three Scheduling Problems

- Weighted completion
- Tardiness
- Weighted Tardiness

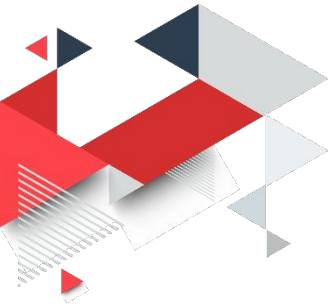
Three Scheduling Problems

For each one testing 4 stochastic scenarios

- Normal Low Var
- Normal High Var
- Normal High Var for low processing time and Low Var for high processing time
- Exponential

Testing Methodology

- Generate all permutations ordering of jobs
- For each permutation, ran **100,000** simulations and computed average objective function
- Brute forced permutations
- Returned the schedule with the objective function best minimized

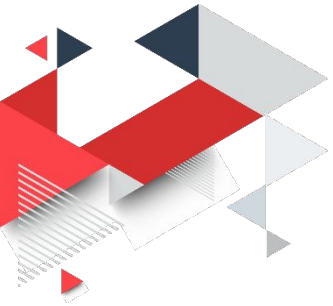


Minimize Weighted Makespan

3.1. Consider $1 \mid\mid \sum w_j C_j$ with the following weights and processing times.

$jobs$	1	2	3	4	5	6	7
w_j	0	18	12	8	8	17	16
p_j	3	6	6	5	4	8	9

```
def sumc(order):
    wc= 0.0
    c = 0.0
    for i in range(len(order)):
        c  = c  + order[i].lowp()
        wc = wc + c*order[i].w
    return(wc)
```



Minimize Weighted Makespan

- Normally solved by weighted shortest processing time
- Optimal schedules
  - 2 – 6 – 3 – 5 – 7 – 4 – 1
  - 2 – 6 – 5 – 3 – 7 – 4 – 1

Minimize Weighted Makespan: Stochastic Results

- Optimal Schedule
  - 2-6-3-5-7-4-1
  - 2-6-5-3-7-4-1
- Normal Low Var
  - 2-6-5-3-4-7-1
- Normal Mix
  - 2-3-6-5-7-4-1
- Normal High Var
  - 2-5-3-6-7-4-1
- Exponential
  - 3-2-6-5-4-7-1

```
#Weighted makespan

best_alg = 0 #0 means unassigned. 1 is first alg, 2 is second ...

#SUMC:
schedule = scenario1()
#get all the permutations of the input schedule
total_list = permeations(schedule)
list_of_sums =[]
#make a list of all the sumc's
for each in total_list:
    sumeach = 0
    for n in range(100000):
        sumeach = sumeach + sumc(each)
    avesum = sumeach/100000
    list_of_sums.append(avesum)

#find the indices that have the same max value (that specific ordering)
for i in range(len(list_of_sums)):
    if list_of_sums[i] == min(list_of_sums):
        schedule = []
        for j in range(7):
            schedule.append(total_list[i][j].n)
        print(schedule,list_of_sums[i])
```

```
[2, 6, 5, 3, 7, 4, 1] 1593.163792350388
```

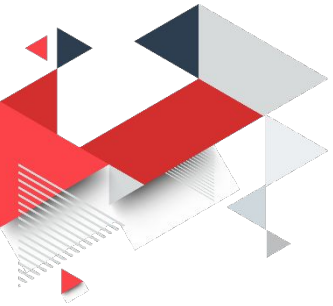
Minimize Tardiness

3.8. Find the optimal sequence for the following instance of the $1 \mid\mid \sum T_j$ problem.

$jobs$	1	2	3	4	5	6	7	8
p_j	6	18	12	10	10	11	5	7
d_j	8	42	44	24	26	26	70	75

```
def tardiness(order):
    c = 0
    t = 0
    for i in range(len(order)):

        c= c + order[i].lowp()
        if c > order[i].d:
            t = t+ c-order[i].d
    return(t)
```



Minimize Tardiness

- Need to use a dynamic processing algorithm to solve
- Optimal schedule:

1-4-5-6-3-2-7-8

Minimize Tardiness: Stochastic Results

- Optimal Schedule
  - 1-4-5-6-3-2-7-8
- Normal Low Var
  - 1-4-5-6-3-2-7-8
- Normal Mix
  - 1-4-6-5-3-2-7-8
- Normal High Var
  - 1-6-4-5-2-7-8-3
- Exponential
  - 1-6-5-4-2-3-8-7

```
#TARDINESS:

schedule = scenario2()
#get permutations
total_list = permeations(schedule)
list_of_tardiness = []
#make a list of all the tardinesses
for each in total_list:
    teach =0
    for n in range(100000):
        teach = teach + tardiness(each)
    avet = teach/100000
    list_of_tardiness.append(avet)


for i in range(len(list_of_tardiness)):
    if list_of_tardiness[i] == min(list_of_tardiness):
        schedule = []
        for j in range(8):
            schedule.append(total_list[i][j].n)
        print(schedule,list_of_tardiness[i])
```

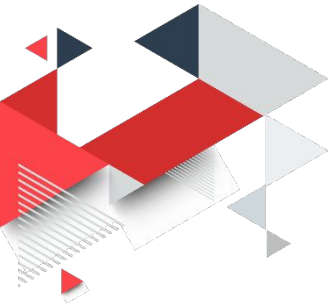
```
[1, 4, 6, 5, 3, 2, 7, 8] 41.05766297702805
```

Minimize Weighted Tardiness

3.10. Find the optimal sequence for the following instance of the $1 \mid\mid \sum w_j T_j$ problem.

$jobs$	1	2	3	4	5	6	7
p_j	6	18	12	10	10	17	16
w_j	1	5	2	4	1	4	2
d_j	8	42	44	24	90	85	68

```
def weightedtardiness(order):
    c=0
    wt=0
    for i in range(len(order)):
        c = c+ order[i].lowp()
        if c>order[i].d:
            wt = wt + (order[i].w)*(c-order[i].d)
    return(wt)
```



Minimize Weighted Tardiness

- Is NP Hard
- Need to use a dynamic processing algorithm to solve
- Optimal schedule: 1-4-2-3-7-6-5

Minimize Weighted Tardiness: Stochastic Results

- Optimal Schedule
  - 1-4-2-3-7-6-5
- Normal Low Var
  - 1-4-2-3-7-6-5
- Normal Mix
  - 1-4-2-3-7-6-5
- Normal High Var
  - 1-4-2-3-6-7-5
- Exponential
  - 1-2-4-6-3-7-5

```
#WEIGHTEDTARDINESS
schedule = scenario3()
#get permutations
total_list = permeations(schedule)
list_of_weighted_tardiness = []
#make a list of all the weightedtardinesses
for each in total_list:
    wteach = 0
    for n in range(100000):
        wteach = wteach + weightedtardiness(each)
    avewt = wteach/100000

    list_of_weighted_tardiness.append(wteach)


for i in range(len(list_of_weighted_tardiness)):
    if list_of_weighted_tardiness[i] == min(list_of_weighted_tardiness):
        schedule = []
        for j in range(7):
            schedule.append(total_list[i][j].n)
        print(schedule, list_of_weighted_tardiness[i])
```

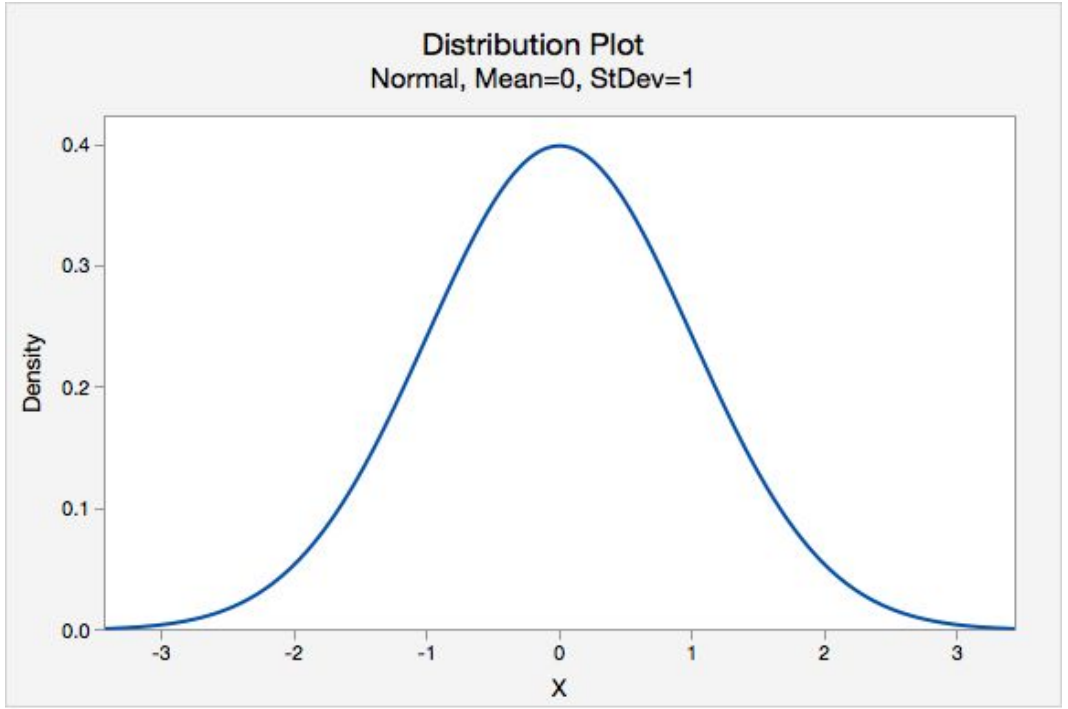
```
[1, 4, 2, 3, 7, 6, 5] 2025.282261698385
```

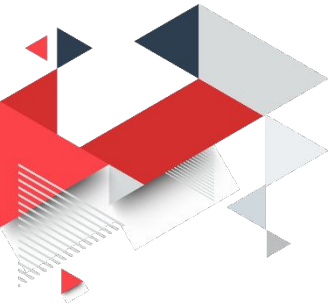
Conclusions

- Normal distribution had results very similar to deterministic
- As Variance grew the optimal schedule tended to change more
- Exponential distribution tended to have very different results from deterministic
- We think this is because there is symmetry in the normal distribution whereas the exponential distribution is not symmetrical
- The inclusion of randomness makes finding the optimal schedule very hard, and most if not all are NP hard

Conclusions

- Exponential distribution tended to have very different results from deterministic
- We think this is because there is symmetry in the normal distribution whereas the exponential distribution is not symmetrical





Limits of Experiment

- As there are more jobs, our brute force method becomes more unfeasible.
- However, we still find immense value in being able to:
  - Write flexible code
  - Obtain empirical answers

Potential Future Tests

- Test further distributions, preferably ones nonsymmetrical
- Make different scenarios where the variance is not related to processing time
- Parallel and other multi machine schedules

THANK YOU!

Questions & Comments