Scheduling Optimization

Manufacturing facilities employ expert schedulers and tools to help visualize and plan for production cycles, scheduled downtime, transitions, etc. This example is a comparison of three methods for scheduling problems:

- Exhaustive search
- Heuristic
- Integer Programming

The example demonstrates Gekko for solving scheduling problems and compares to exhaustive search and heuristic method.

Objective: Minimize the delay for the production of 5 products on 2 machines. Each product requires a different amount of time to process for each step.

There are 5 products that require 2 processing steps on separate machines. The two machines (machine Y and machine Z) work one after the other. A product must go through machine Y first and after that through machine Z. Both machines can work simultaneously, but each machine cannot work on more than 1 product at a time. The factory needs to minimize the time of production of all given products. This is equivalent to minimizing the idle time of machine Z.

Product	P0	P1	P2	P3	P4
Machine Y:	10	20	15	40	8
Machine Z:	20	30	10	25	18

Exhaustive Search

There are *n!* permutations for a production list of *n* length. For this specific problem with 5 products, there are *5!=120* combinations. An exhaustive search calculates the makespan (total time) for all possible combinations.

```
from itertools import permutations
import pandas as pd

Y = {0: 10, 1: 20, 2: 15, 3: 40, 4: 8}
Z = {0: 20, 1: 30, 2: 10, 3: 25, 4: 18}

products = Y.keys()
np = len(products)
pm = [] # list of permutations
tm = [] # list of makespan time
for order in permutations(products):
    # append permutation to the pm list
    pm.append(order)
    # lookup time for each step
    ty = [0]*(np+1); tz = [0]*(np+1)
    for i in range(np):
        ty[i] = Y[order[i]]
        tz[i+1] = Z[order[i]]
    # calculate makespan time
    t = 0
    for i in range(np+1):
        t += max(ty[i],tz[i])
    tm.append(t)

# convert to Pandas DataFrame for sorting
r = pd.DataFrame(pm,columns=['T0','T1','T2','T3','T4'])
r['makespan'] = tm
r.sort_values('makespan',inplace=True)
print(r.head(10))
```

[S[Get Code]]

The top 10 best schedules show that there are two best candidate schedules each with a makespan of 121.

```
  T0  T1  T2  T3  T4  makespan
   4   0   1   3   2       121
   4   2   0   1   3       121
   0   1   3   2   4       123
   0   1   3   4   2       123
   4   1   3   0   2       123
   4   1   3   2   0       123
   0   2   4   1   3       125
   0   4   1   3   2       125
   2   4   0   1   3       128
   2   0   1   3   4       128
```

While an exhaustive search is fast for this problem, it becomes impractical to calculate all combinations for more complex schedules. Heuristic or optimization methods are used when exhaustive search is too computationally demanding.

Heuristic Solution: Johnson's Rule

Similar to the Knapsack Optimization, there is a heuristic solution that is problem-specific and achieves optimal results under certain restrictive assumptions. Johnson's Rule is a method for scheduling the jobs in two work centers with constant job time. Job times are not affected by the sequence and products are processed on the first machine before moving to the second machine. Johnson's rule selects the product with the shortest activity time and assigns it to be processed first if the shortest time is for machine Y. If the shortest time is for machine Z, it assigns the product to the end of the list. The next shortest time is selected and assigned as the second or second to last product if those were selected previously. The process is repeated until all products are ordered.

- The lowest time is *P4* and it is assigned to be first because the lowest time is for machine Y.
- Then next lowest is *P2* and it is assigned to be last because the lowest time is for machine Z.

The order is *P4, P0, P1, P3, P2* with total delay of 26 on machine Z and total makespan time of 121 to complete all processing. This is the same as the first row in the sorted exhaustive search list.

Sequence	T0	T1	T2	T3	T4	T5
Machine Y:	8	10	20	40	15	
Machine Z:		18	20	30	25	10
Delay (26):	8	8	0	0	10	0
Time (121):	8	18	20	40	25	10

Optimization Solution: Integer Programming

The same heuristics do not work if there are any changes in the assumptions that would invalidate Johnson's rule. This may happen if the problem changes or there is a priority for finishing one product over another. There may also be additional constraints a third processing step could be introduced. A more flexible approach is to solve the problem with Integer Linear Programming (ILP). Variables, equations, and an objective are created to fit into a standard form that an optimizer can solve.

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

Solve the problem with an integer programming solver (APOPT) by setting up each decision as a binary variable (0 or 1). A zero (0) is a decision to not place the item in the schedule while a one (1) is a decision to include it. There are 5 products with processing times $ty_0, ..., ty_4$ and $tz_0, ..., tz_4$. The decision variables are a matrix of 25 values that can be 0 or 1:

$$x = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,0} & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix}$$

Each row represents a new processing step (5 total) and each column is the product (5 total). Only one product is selected to be processed first, second, third, and so on. This is mathematically expressed with the sum of each row equal to 1.

$$\sum_{j=0}^{4} x_{i,j} = 1 \quad \forall\, i = 0 \ldots 4$$

A product is only processed once with the sum of each row equal to 1.

$$\sum_{i=0}^{4} x_{i,j} = 1 \quad \forall\, j = 0 \ldots 4$$

The time to process on machine Y (ty) and machine Z (tz) is:

$$ty_i = \sum_{j=0}^{4} Y_j x_{i,j} = 1 \quad \forall\, i = 0 \ldots 4$$

$$tz_i = \sum_{j=0}^{4} Z_j x_{i,j} = 1 \quad \forall\, i = 0 \ldots 4$$

The delay $d \geq 0$ is calculated with help of a slack variable $s \geq 0$ that is non-zero only if the delay would be negative. This clips the calculated delay at a lower bound of 0 if the processing time for Z is less than the next processing time for Y.

$$d_i \geq tz_{i-1} - ty_i + s_i$$

The delay and slack variables are minimized to find the optimal solution.

```
from gekko import GEKKO
m = GEKKO()
Y = {0: 10, 1: 20, 2: 15, 3: 40, 4: 8}
Z = {0: 20, 1: 30, 2: 10, 3: 25, 4: 18}

x = m.Array(m.Var,(5,5),value=0,lb=0,ub=1,integer=True)
for i in range(5):
    # 5 time slots, 5 order options for machine Y and Z
    m.Equation(m.sum([x[i,j] for j in range(5)])==1)
    m.Equation(m.sum([x[j,i] for j in range(5)])==1)

# time to process on Y and Z
ty = m.Array(m.Var,5)
tz = m.Array(m.Var,5)
for i in range(5):
    # time to process on Y
    m.Equation(ty[i] == m.sum([x[i,j]*Y[j] for j in range(5)]))
    # time to process on Z
    m.Equation(tz[i] == m.sum([x[i,j]*Z[j] for j in range(5)]))

# delay time on Z
d = m.Array(m.Var,5,lb=0)
s = m.Array(m.Var,5,lb=0)
m.Equation(d[0]==ty[0])
m.Equation(s[0]==0)
for i in range(1,5):
    m.Equation(d[i]>=tz[i-1]-ty[i]+s[i])
m.Minimize(m.sum(d))
m.Minimize(1e-3*m.sum(s))

m.options.SOLVER=1
m.solve()

print('Order of processing, rows=order, columns=product')
print(x)
print('time on machine Y')
print(ty)
print('time on machine Z')
print(tz)
print('delay')
print(d)
```

[S[Get Code]]

The problem is solved with the APOPT solver in 0.1 sec with a minimized objective (time delay) of 11.

```
Order of processing, rows=order, columns=product
 [[[0.0] [0.0] [0.0] [0.0] [1.0]]
  [[0.0] [0.0] [1.0] [0.0] [0.0]]
  [[1.0] [0.0] [0.0] [0.0] [0.0]]
  [[0.0] [1.0] [0.0] [0.0] [0.0]]
  [[0.0] [0.0] [0.0] [1.0] [0.0]]]
time on machine Y
 [[8.0] [15.0] [10.0] [20.0] [40.0]]
time on machine Z
 [[18.0] [18.0] [20.0] [30.0] [25.0]]
delay
 [[8.0] [3.0] [0.0] [0.0] [0.0]]
```

[S[Get Code]]

The order is *P4, P2, P0, P1, P3* with total makespan time of 121 to complete all processing.

Sequence	T0	T1	T2	T3	T4	T5
Machine Y:	8	15	10	20	40	
Machine Z:		18	10	20	30	25
Delay (11):	8	3	0	0	0	0
Time (121):	8	18	10	20	40	25

The order of the sequence is different than Johnson's rule but the total makespan time (121) is the same. The optimizer found the equally optimal solution as the second row in the sorted exhaustive search list.

SEARCH

Go

COURSE INFORMATION

Syllabus
Book Chapters
Info Sheet
Schedule
Expectations
Competencies
Optimization Software
YouTube Playlist
YouTube

RELATED COURSES

Begin Python
Begin Matlab
Begin Java

Engineering Computing
Data Science
Machine Learning

Control (MATLAB)
Control (Python)

Optimization
Dynamic Optimization

HOMEWORK

Optimization Basics
Tubular Column
Two Bar Truss
Step Cone Pulley
Beam Column
Crane Hook
Rocket Launch
Spring Design
Heat Integration
Slurry Pipeline
Oxygen Storage
Quasi-Newton Methods
Discrete Design
Simulated Annealing
KKT Conditions
Interior Point Method

PROJECTS

Application Project
Solver Project

ACTIVITIES

1-MATLAB and Python
2-Equation Residuals
3-Financial Objectives
4-Parallel Computing
5-Advanced Programming
6-Logical Conditions
7-Simulated Annealing
8-Climate Control
9-Dynamic Estimation
10-Vapor Liquid Equilibrium
11-Ethyl Acetate Kinetics
12-Dye Fading Kinetics
13-Linear Regression
14-Nonlinear Regression
15-Knapsack Optimization
16-Schedule Optimization

LECTURE NOTES

Optimization Introduction
Mathematical Modeling
Unconstrained Optimization
Discrete Optimization
Genetic Algorithms
Constrained Optimization
Robust Optimization
Dynamic Optimization

EXTRA CONTENT

Box Folding
Circle Challenge
Linear Programming
Minimax or Maximin
Slack Variables

Admin