Curso: Heurística – CM0439
Profesor: Juan Carlos Rivera Agudelo
Julio 31 de 2023
Métodos constructivos y aleatorizados

METODOS CONSTRUCTIVOS Y ALEATORIZADOS

Teniendo en cuenta el problema asignado para trabajar durante el semestre (Job Shop Scheduling Problem – JSSP –, ver Anexo 1), se deben implementar tres algoritmos: un algoritmo constructivo y dos algoritmos constructivos aleatorizados (construcción GRASP, construcciones con ruido, construcciones basadas en colonias de hormigas). Los algoritmos deben poderse parametrizar para ejecutarse con diferentes valores de cada parámetro según sea el caso (ver Anexo 4 – ejemplo de definición de parámetros).

El algoritmo debe ser desarrollado en Python o Matlab (otros lenguajes pueden ser utilizados sujetos a previa aprobación por parte del profesor).

Las pruebas del algoritmo deben ser realizadas con las instancias de prueba disponibles en las librerías para el problema seleccionado. Las instancias están disponibles en Interactiva Virtual con el formato descrito en el Anexo 2.

Cuando los algoritmos desarrollados sean basados en ideas de otros autores, i.e. método de los ahorros, NEH u otros, se debe indicar explícitamente en la descripción del método.

Formato de Entrega:

1. Realizar una presentación en diapositivas en la que se incluya:
    - Descripción de los algoritmos implementados.
    - Descripción de los resultados obtenidos, incluyendo:
        - Comparación con una cota inferior.
        - Comparaciones utilizando diferentes valores para cada parámetro.
        - Comparación entre los métodos implementados.
        - Tiempo de cómputo.
    - Conclusiones.
    - Se deben incluir todas las referencias bibliográficas que hayan servido de apoyo.
    - Se pueden incluir otras secciones en el informe.
2. Enviar por Interactiva Virtual los archivos correspondientes al código, los archivos de resultados y la presentación (con ayudas audiovisuales, formato libre).
3. Se realizará una sustentación oral individual en horario acordado con el profesor.

Se debe adjuntar los archivos correspondientes a los códigos de los algoritmos. Éstos deben poderse ejecutar sin necesidad de ingresar datos o modificaciones adicionales de forma manual. El algoritmo debe producir, para cada instancia disponible, un archivo de resultado con el nombre y formato descrito en el Anexo 3.

FECHA DE ENTREGA: La fecha límite de entrega es el lunes 4 de septiembre de 2023 (vía Buzón de Interactiva Virtual).

Curso: Heurística – CM0439
Profesor: Juan Carlos Rivera Agudelo
Julio 31 de 2023
Métodos constructivos y aleatorizados

UNIVERSIDAD EAFIT

ANEXO 1

DEFINICIÓN Y FORMULACIÓN DEL PROBLEMA

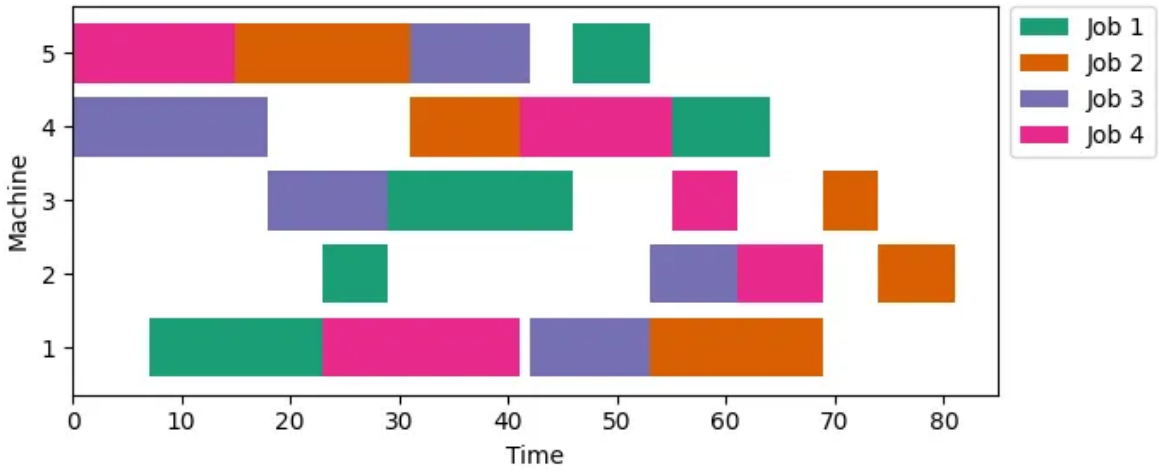
The job-shop scheduling problem (JSSP) is a widely studied optimization problem with several industrial applications. The goal is to define how to minimize the makespan required to allocate shared resources (machines) over time to complete competing activities (jobs). As for other optimization problems, mixed-integer programming can be an effective tool to provide good solutions, although for large-scale instances one should probably resort to heuristics.

Problem statement

Suppose a set of jobs needs to be processed in a set of machines, each in a given order. For instance, job number 1 might need to be processed in machines (1, 4, 3, 2), whereas job number 2 in (2, 3, 4, 1). In this case, before going to machine 4, job 1 must have gone to machine 1. Analogously, before going to machine 1, job 2 must have been processed in machine 4.

Each machine can process only one job at a time. Operations are defined by pairs (machine, job) and each has a specific processing time. Therefore, the total makespan depends on how one allocates resources to perform tasks.

The figure below illustrates an optimal sequence of operations for a simple instance with 5 machines and 4 jobs. Notice that each machine processes just one job at a time and each job is processed by only one machine at a time.



Example of JSSP with five machines and four jobs.

Curso: Heurística – CM0439
Profesor: Juan Carlos Rivera Agudelo
Julio 31 de 2023
Métodos constructivos y aleatorizados

Mixed-integer linear programming models

Following the study of Ku & Beck (2016), two formulations for the JSSP are presented: the disjunctive model (Manne, 1960) and the time-index model (Bowman, 1959; Kondili, 1993). Those interested might refer to Wagner (1959) for a third formulation (rank-based). The disjunctive model is surely the most efficient of the three for the original problem. However, others might be easier to handle when incorporating new constraints that might occur in real-world problems.

In the disjunctive model, let us consider a set J of jobs and a set M of machines. Each job j must follow a processing order $(\sigma_1^j, \sigma_2^j, \dots, \sigma_k^j)$ and each operation (m, j) has a processing time $p_{m,j}$. The decision variables considered are the time that job j starts on machine m, x_{mj}; a binary that marks precedence of job i before j on machine m, z_{ijm}; and the total makespan of operation, C_{max}, which is itself the minimization objective.

We need to create constraints to ensure that:

1. The starting time of job j in machine m must be greater than the starting time of the previous operation of job j plus its processing time.
2. Each machine processes just one job at a time. To do this, we state that if i precedes j in machine m, the starting time of job j in machine m must be greater than or equal to the starting time of job i plus its processing time.
3. Of every pair of jobs i, j one element must precede the other for each machine m in M.
4. The total makespan is greater than the starting time of every operation plus its processing time.

And we get the following formulation:

$$
\begin{aligned}
\min \quad & C \\
\text{s.t.} \quad & x_{\sigma_{h-1}^j,j} + p_{\sigma_{h-1}^j,j} \leq x_{\sigma_h^j,j} && \forall\ j \in J; h \in (2, \dots, |M|) \\
& x_{m,j} + p_{m,j} \leq x_{m,k} + V(1 - z_{m,j,k}) && \forall\ j,k \in J, j \neq k; m \in M \\
& z_{m,j,k} + z_{m,k,j} = 1 && \forall\ j,k \in J, j \neq k; m \in M \\
& x_{\sigma_{|M|}^j,j} + p_{\sigma_{|M|}^j,j} \leq C && \forall\ j \in J \\
& x_{m,j} \geq 0 && \forall\ j \in J; m \in M \\
& z_{m,j,k} \in \{0,1\} && \forall\ j,k \in J; m \in M
\end{aligned}
$$

In which, V is an arbitrarily large value (big M) of the "either-or" constraint.

The next formulation explored is the time-indexed model. It is limited in the sense that only integer processing times can be considered, and one can notice that it produces a constraint matrix with several nonzero elements, which makes it computationally more expensive than the disjunctive model. Furthermore, as processing times increase, the number of decision variables increases as well.

In the time-indexed model, we consider the same sets of jobs J and machines M, besides a set of discrete intervals T. The choice of the size of T might be oriented in the same way as the definition of V: the sum of all processing times. The same parameters of the order of jobs and processing times will be used too. However, in this approach, we only consider binary variables that mark it job j starts at machine m at instant t, x_{mjt}, besides the real-valued (or integer) makespan C_{max}.

Let us formulate the constraints:

1. Each job j at machine m starts only once.
2. Ensure that each machine processes just one job at a time. And this is the hard one in the time-indexed approach. To do this, we state that at most one job j can start at machine m during the time span between the current time t and p_{mj} previous times. For each machine and time instant.
3. The starting time of job j in machine m must be greater than the starting time of the previous operation of job j plus its processing time.
4. The total makespan is greater than the starting time of every operation plus its processing time.

$$\min \quad C$$

$$\text{s.t.} \quad \sum_{t \in T} x_{m,j,t} = 1 \qquad \forall\, j \in J; m \in M$$

$$\sum_{j \in J} \sum_{t' \in (t - p_{m,j} + 1, \ldots, t)} x_{m,j,t'} \leq 1 \qquad \forall\, m \in M; t \in T$$

$$\sum_{t \in T} (t + p_{\sigma^j_{h-1},j}) x_{\sigma^j_{h-1},j,t} \leq t x_{\sigma^j_h,j,t} \quad \forall\, j \in J; h \in (1, 2, \ldots, |M|)$$

$$\sum_{t \in T} (t + p_{m,j}) x_{m,j,t} \leq C \qquad \forall\, j \in J; m \in M$$

$$x_{m,j,t} \in \{0, 1\} \qquad \forall\, j; m \in M; t \in T$$

Kondili, E., & Sargent, R. W. H. (1988). *A general algorithm for scheduling batch operations* (pp. 62–75). Department of Chemical Engineering, Imperial College.

Ku, W. Y., & Beck, J. C. (2016). *Mixed integer programming models for job shop scheduling: A computational analysis*. Computers & Operations Research, 73, 165–173.

Manne, A. S. (1960). On the job-shop scheduling problem. *Operations research*, *8*(2), 219–223.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval research logistics quarterly*, *6*(2), 131–140.

Curso: Heurística – CM0439
Profesor: Juan Carlos Rivera Agudelo
Julio 31 de 2023
Métodos constructivos y aleatorizados

ANEXO 2

FORMATO DE DATOS DE LAS INSTANCIAS

En el archivo "JSSP Instances.zip" (disponible en EAFIT Interactiva) se encuentran 16 instancias con datos para el JSSP, donde el número de trabajos varía entre 15 y 100, y el número de máquinas varía entre 15 y 20. El formato de cada instancia es el siguiente:

La primera fila indica el número de trabajos (n) y el número de máquinas (m) de la instancia.

Las siguientes n filas indican, para cada trabajo, el tiempo de procesamiento de cada trabajo en cada proceso.

Posteriormente, las siguientes n filas indican, para cada trabajo, el en cual máquina se realiza cada proceso; es decir, en que orden debe pasar cada trabajo por el conjunto de máquinas.

A continuación se presenta un ejemplo con $n = 3$ y $m = 4$:

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

Curso: Heurística – CM0439
Profesor: Juan Carlos Rivera Agudelo
Julio 31 de 2023
Métodos constructivos y aleatorizados

UNIVERSIDAD EAFIT

ANEXO 3

FORMATO DE ARCHIVO DE RESULTADOS

Los resultados obtenidos se deben registrar en un archivo de Excel en el cual cada hoja contenga los resultados de cada instancia. El nombre del archivo debe ser JSSP_<nombre_estudiante>_<método>.xlsx, donde <nombre_estudiante> debe ser reemplazado por el nombre (nombre, o apellido, o iniciales) de cada estudiante, y <método> debe ser reemplazado por un nombre indicador del método utilizado para hallar los resultados (constructivo, GRASP, ACO, LS, VNS, GA,…). Cada hoja del archivo de Excel debe ser nombrada con el nombre de la instancia respectiva (usar los mismos nombres utilizados en el archivo de datos).

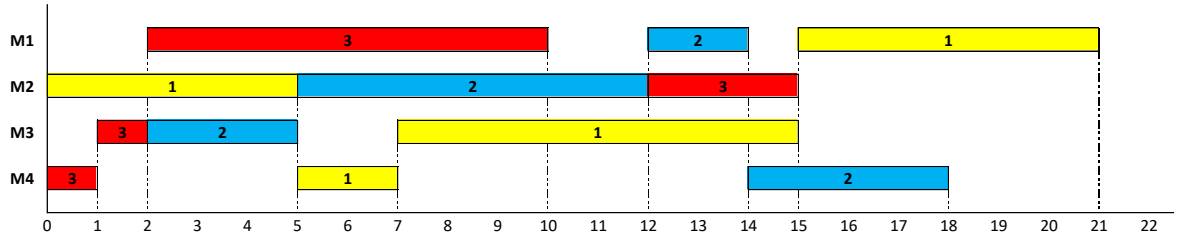
Los resultados deben tener el siguiente formato: Cada solución se debe componer de un conjunto ordenado de trabajos a ejecutar en cada máquina. Así, cada hoja debe contener M filas, una por cada máquina, indicando la secuencia ordenada de trabajos ordenados. La última fila (M+1) debe contener el valor de la función objetivo (tiempo de terminación del trabajo que termina más tarde) y el tiempo de cómputo (en milisegundos redondeado al valor entero más cercano).

El siguiente es un ejemplo de una solución factible para la instancia de ejemplo presentada en el Anexo 2.

3 2 1
1 2 3
3 2 1
3 1 2
21 1

En la solución anterior, la primera máquina ejecuta los trabajos en el orden 3, 2 y 1. La segunda máquina los ejecuta en el orden 1, 2 y 3. La tercera y cuarta máquina ejecutan los trabajos en el orden 3, 2, 1 y 3, 1, 2, respectivamente. El makespan es 21 unidades (correspondiente al tiempo de terminación del trabajo 1 en la máquina 1. El tiempo de ejecución es 1 milisegundo.

La siguiente figura ilustra la solución descrita.



UNIVERSIDAD EAFIT®

ANEXO 4

EJEMPLO DE DEFINICIÓN DE PARÁMETROS

La siguiente figura muestra un algoritmo en lenguaje de programación Python donde luego de cargar librerías y funciones se definen los valores de los 4 parámetros a utilizar: nsol, alpha, K y r. En los algoritmos implementados los parámetros deben tener valores constantes; no deben ser leídos de algún archivo, ni solicitados al usuario en cada ejecución.

```
import xlwt
from xlwt import Workbook

from Constructive import Constructive

from GRASP1 import GRASP1
from GRASP2 import GRASP2
from Noise import Noise


# Data reading
from Read import read_fsspsc


wb = Workbook()
sheet1 = wb.add_sheet('Randomized')

nsol=100   # Number of solutions
alpha=0.05 # GRASP parameter
K=5        # Maximum RCL size
r=5        # Noise

for id in range(1,21):
    print(id)
    n,m,L,dur = read_fsspsc(id)

    Z0,S0,I0,F0,Fi0,t0=Constructive(n,m,dur,L,id)
    print("C:\t",Z0,"\t",t0)

    Z1,S1,I1,F1,Fi1,t1=GRASP1(n,m,dur,L,id,alpha,nsol)
    print("GRASP1:\t",Z1,"\t",t1)

    Z2,S2,I2,F2,Fi2,t2=GRASP2(n,m,dur,L,id,K,nsol)
    print("GRASP2:\t",Z2,"\t",t2)

    Z3,S3,I3,F3,Fi3,t3=Noise(n,m,dur,L,id,r,20)
    print("Noise:\t",Z3,"\t",t3)
```