Universidad de Oviedo

Intelligent Scheduling and Optimization

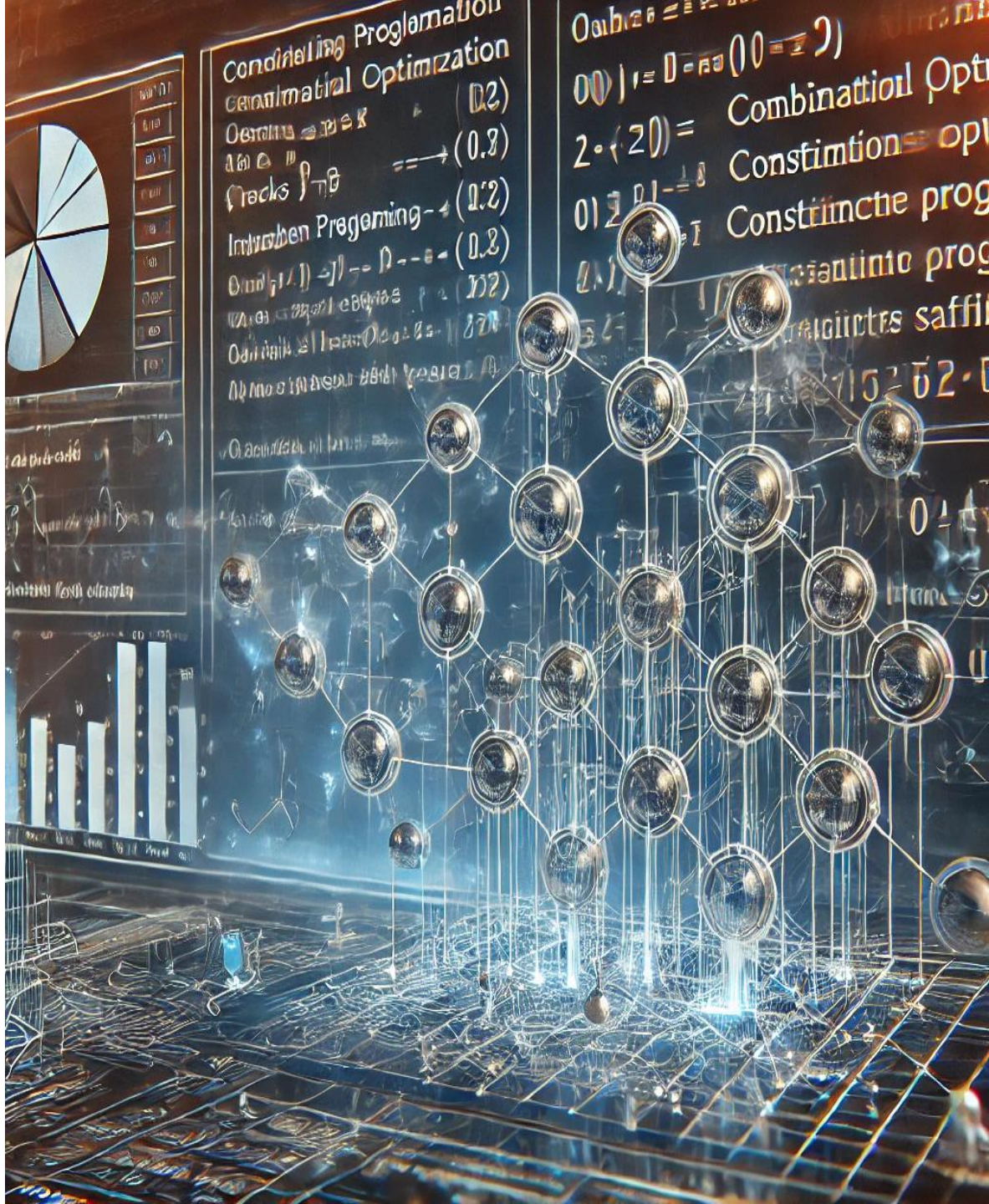
Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

Tema 2: Introducción a los problemas de Scheduling

María Rita Sierra Sánchez
{sierramaria}@uniovi.es

Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática

ceisIA Centro de Estudios sobre el Impacto Social de la Inteligencia Artificial | Universidad de Oviedo



Pseudocódigo Planificador Básico


```
Entrada: instancia de un problema JSS
A <- Tareas candidatas a ser planificadas (primeras sin planificar que pueden empezar)
Mientras no se hayan planificado todas las tareas Hacer
    tarea<- Elegir una tarea de A con algún criterio (regla)
    Planificar(tarea):
        Asignarle tiempo de inicio (máximo tiempo de fin entre el de fin de la tarea
        anterior en su trabajo y en su máquina).
        Registrar el tiempo de fin de la máquina que requiere la tarea planificada

    Actualizar A (borrar la tarea planificada e insertar la siguiente tarea en su
    trabajo)
    Actualizar el coste de la función objetivo, tras planificar la tarea
FinMientras
Retornar coste (valor de la función objetivo) de la solución
```

Código Prácticas Planificadores

Clase **"Problem"**: Almacena los datos de una instancia del problema JSS

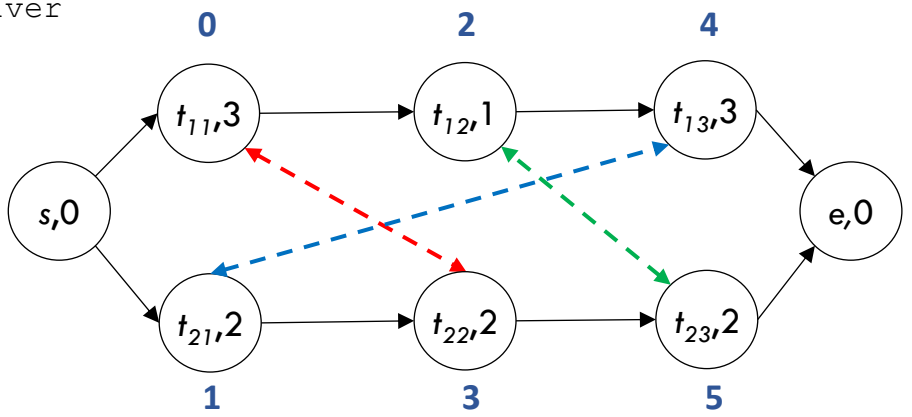
- **Atributos**
  - file_name: nombre del archivo con la instancia del problema a resolver
  - name: nombre de la instancia
  - num_jobs: número de trabajos
  - num_machines: número de máquinas
  - pi: duraciones de las tareas
  - mi: máquinas que requieren las tareas

```
file_name: "ft03.txt"
name: "ft03"
num_jobs: 2
num_machines: 3
pi:[3, 2, 1, 2, 3, 2]
mi:[0, 2, 1, 0, 1, 1]
```

- **Métodos**
  - __init__(file_name = ""): constructor, recibe el nombre del fichero con la instancia e inicializa los atributos
  - _read_data(file_name): lee los datos del problema "file_name"
  - _print_data(): muestra por pantalla los datos del problema leído

ft03.txt

2	3				
0	3	1	1	2	3
2	2	0	2	1	2



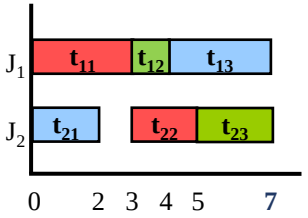
Las tareas se almacenan en vertical:

La tarea 3 es la t_{22} . Así: pi[3] es 2 y mi[3] es 0

Código Prácticas Planificadores

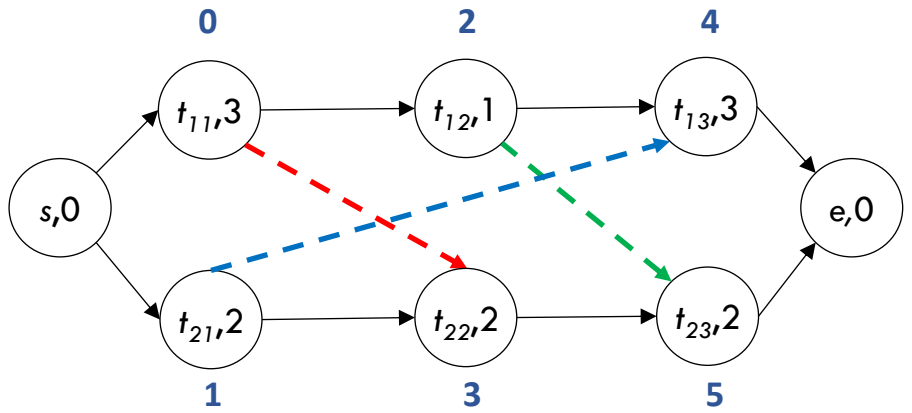
Clase **"Solution"**: Almacena la solución de una instancia del problema JSS

- **Atributos**
  - num_jobs: número de trabajos
  - num_machines: número de máquinas
  - prob: objeto Problem
  - st: tiempos de inicio de las tareas
  - objectiveCost: coste de la solución
  - endTimeMachine: tiempos de fin de las máquinas
  - scheduled_tasks: tarea planificada (V) o no (F)



ft03.txt

2	3				
0	3	1	1	2	3
2	2	0	2	1	2




```
num_jobs: 2
num_machines: 3
prob:objeto Problem con datos de "ft03.txt"
st: [0, 0, 3, 3, 4, 5]
objectiveCost: 7
endTimeMachine: [5 7 7]
```

Código Prácticas Planificadores

Clase **"Solution"**: Almacena la solución de una instancia del problema JSS

- **Métodos**
  - `__init__(problem)`: constructor, recibe un objeto problema e inicializa los atributos que no cambian al construir la solución
  - `init_solution()`: inicializa los atributos que cambian al construir la solución: st, objectiveCost, endTimeMachine, scheduled_tasks
  - `print_solution()`: muestra por pantalla la solución al problema (contenido de st y objectiveCost)
  - `save_solution(file_name)`: guarda en un archivo de texto los datos del problema y su solución
  - `paint_gantt(graphic_name)`: genera una imagen ".jpg" con el diagrama de Gantt de la solución
  - `_list_to_string(list)`: convierte una lista a un string
  - `_generate_machine_colors(num_machines)`:genera un vector de colores (hexadecimales), uno por cada máquina.

Código Prácticas Planificadores

Clase **"SchedulerGenerator"**: clase generadora de planificaciones solución para una instancia del problema JSS

- **Atributos**
  - prob: objeto Problem
  - sol: objeto solution
  - configuration: cadena con la configuración de la planificación que no cambia (independientemente del tipo de planificación)
  - graphName: cadena con la descripción completa de la planificación (configuración + tipo planificación)
  - _select_rule: regla a aplicar
  - _objective_function: función objetivo a considerar

Código Prácticas Planificadores

Clase **"SchedulerGenerator"**: clase generadora de planificaciones solución para una instancia del problema JSS

- **Métodos**
  - `__init__(file_name, rule, objectiveF)`: constructor, recibe nombre del fichero con la instancia a resolver, el nombre de la regla y el de la función objetivo, e inicializa atributos
  - `execute_basic_scheduler()`: construye una planificación semi-activa con el planificador "Básico" (_basic)
  - `execute_GyT()`: construye una planificación activa con el planificador "G&T" (_gyt)
  - `paint_gannt_schedule()`: pinta el diagrama de Gannt de la solución calculada
  - `save_solution()`: guarda a fichero de texto la solución

Código Prácticas Planificadores

Clase **"SchedulerGenerator"**: clase generadora de planificaciones solución para una instancia del problema JSS

- **Métodos**
  - `_initialize(scheduleType)`: inicializa el objeto solución y la descripción de la planificación
  - `_first_unschedule_tasks()`: calcula el conjunto de las tareas candidatas a ser planificadas (primeras tareas sin planificar de cada trabajo)
  - `_schedule_task(task)`: planifica la tarea que recibe como argumento "task"
  - `_update_US(self, A, task)`: actualiza el conjunto A de las tareas candidatas a ser planificadas: borra "task" de A, e inserta en A su sucesora en el trabajo
  - `_calculate_set_B(A)`: construye a partir del conjunto de tareas candidatas a ser planificadas A el conjunto B, con las tareas cuya planificación garantiza mantener un schedule activo (**Implementar**)
  - `_calculate_st_US_Tasks(A)`: calcula los tiempos de inicio de las tareas candidatas a ser planificadas (A) (**Implementar**)
  - `_task_ends_before(self, A, st_A)`: calcula, teniendo en cuenta las tareas candidatas a ser planificadas (A) y sus tiempos de inicio (st_A), el menor tiempo de fin de las tareas en A (C), la tarea que estableció C, y la máquina de esta (**Implementar**)

Código Prácticas Planificadores

Fichero **"rules"**: contiene las reglas que se pueden emplear en los planificadores, un diccionario de reglas y una "factoría" de reglas.

- **Funciones**
  - `random_rule(A,_)`: selecciona aleatoriamente una tarea de A (tareas candidatas) para ser planificada
  - `SPT_rule(A,pro)`: teniendo en cuenta los datos de la instancia del problema (objeto pro), selecciona la tarea de A (tareas candidatas) con menor tiempo de procesamiento para ser planificada. En caso de empate elige aleatoriamente
  - `LPT_rule(A,pro)`: teniendo en cuenta los datos de la instancia del problema (objeto pro), selecciona la tarea de A (tareas candidatas) con mayor tiempo de procesamiento para ser planificada. En caso de empate elige aleatoriamente
  - ……
  - `switchRule={"RR" : random_rule, "SPT": SPT_rule, "LPT":LPT_rule, "MWR":MWR_rule, "LWR":LWR_rule, "MOR":MOR_rule, "LOR":LOR_rule}` → Diccionario de reglas
  - `rule_factory(name)`: a partir del nombre de la regla devuelve el nombre de la función que la implementa
  - `_rules_to_string()`:retorna una cadena con todos los nombres de las reglas contempladas en el diccionario de reglas

Código Prácticas Planificadores

Fichero **"objective_functions"**: contiene las funciones objetivo que se pueden emplear en los planificadores, un diccionario de funciones objetivo y una "factoría" de funciones objetivo.

- **Funciones**
  - `makespan(prob,sol)`: calcula, teniendo en cuenta los datos del problema y la solución construida calcula su "Makepan" ($C_{max} = \max C_{ij}$)
  - `total_flow_time(prob,sol)`: calcula, teniendo en cuenta los datos del problema y la solución construida calcula su "Total Flow Time" ($C_{sum} = \Sigma C_i$)
  - `switchOF = {"MK" : makespan, "TFT": total_flow_time}` → Diccionario de funciones objetivo
  - `objective_fuction_factory(name)`: a partir del nombre del objetivo devuelve el nombre de la función que la implementa
  - `__functions_to_string ()`:retorna una cadena con todos los nombres de las funciones objetivo contempladas en el diccionario de funciones

Código Planificador Básico

```
def execute_basic_scheduler(self):
    #Inicializar
    self._initialize("_gyt") # Estructuras de la solución
    unscheduledTasks = self.prob.num_jobs*self.prob.num_machines  # contador tareas planificar
    # Calcular conjunto tareas candidatas a planificar (primeras sin planificar de cada trabajo)
    A = self._first_unschedule_tasks()
    # Mientras no se hayan planificado todas las tareas
    while (unscheduledTasks > 0):
      task = self._select_rule(A, self.prob) # Elegir tarea a planificar con función de prioridad
      self._schedule_task(task) # Planificar la tarea
      # Actualizar conjunto A: quitar tarea planificada y añadir sucesora en trabajo
      A = self._update_A(A,task)
      # Actualizar la función objetivo
      self.sol.objectiveCost=self._objective_function(self.prob, self.sol)
      # Decrementamos el contador de las tareas sin planificar
      unscheduledTasks -= 1
    return self.sol.objectiveCost
```

Código Planificador G&T

```
def execute_GyT(self):
    #Inicializar
    self._initialize("_gyt") # Estructuras de la solución
    unscheduledTasks = self.prob.num_jobs*self.prob.num_machines  # contador tareas planificar
    # Calcular conjunto tareas candidatas a planificar (primeras sin planificar de cada trabajo)
    A = self._first_unschedule_tasks()
    # Mientras no se hayan planificado todas las tareas
    while (unscheduledTasks > 0):
      # Construir el conjunto B
      B = self._calculate_set_B(A)
      task = self._select_rule(B, self.prob) # Elegir tarea a planificar con función de prioridad
      self._schedule_task(task) # Planificar la tarea
      # Actualizar conjunto A: quitar tarea planificada y añadir sucesora en trabajo
      A = self._update_A(A,task)
      # Actualizar la función objetivo
      self.sol.objectiveCost=self._objective_function(self.prob, self.sol)
      # Decrementamos el contador de las tareas sin planificar
      unscheduledTasks -= 1
    return self.sol.objectiveCost
```

Código Prácticas Planificadores

Fichero **"Scheduler"**: contiene el programa principal desde el que se lanzan los planificadores (Básico y G&T) para resolver instancias del JSS. Su ejecución desde el IDE (sin realizar ningún cambio) resuelve la instancia "ft03".

- **Funciones**
  - check_args(vArgs): verifica si los argumentos de la ejecución que se le pasan por línea de comandos al programa principal son o no correctos

También se puede ejecutar desde el terminal en línea de comandos:

- Escribiendo. Ej: python "Scheduler.py" .\Instances\ft03.txt RR MK
- Con un "batch file" o "archivo de lotes" (".bat"). Archivo de texto que ejecuta secuencias de comandos y permite automatizas tareas que requieren muchas instrucciones o que son repetitivas. En la práctica lo podemos emplear para lanzar la ejecución de varias instancias.

Escribe: .\ Ejecuta10_ft03_RR.bat

Si archivo no existe
"**>>**" y "**>**" lo crean
Si existe
- "**>>**" añade
- "**>**" sobreescribe

Ejecuta10_ft03_RR.bat

```
python "Scheduler.py" .\Instances\ft03.txt RR MK >> .\Res\sal_ft03_RR_MK.txt
python "Scheduler.py" .\Instances\ft03.txt RR MK  >> .\Res\sal_ft03_RR_MK.txt
python "Scheduler.py" .\Instances\ft03.txt RR MK  >> .\Res\sal_ft03_RR_MK.txt
python "Scheduler.py" .\Instances\ft03.txt RR MK  >> .\Res\sal_ft03_RR_MK.txt
python "Scheduler.py" .\Instances\ft03.txt RR MK  >> .\Res\sal_ft03_RR_MK.txt
python "Scheduler.py" .\Instances\ft03.txt RR MK  >> .\Res\sal_ft03_RR_MK.txt
….
```

Practiquemos con el código de los SGS

Trabajamos con la Excel "ResultadosEjecuciones_Individuales"

- Para la instancia ft03, hoja "FT03"
  1. Ejecuta el planificador básico por programa y rellena los valores de MK y TFT
  2. Ejecuta el planificador básico y el G&Y. Para ello, ejecuta el script: Ejecuta10_ft03_RR.bat
     - ¿Cuántas soluciones de costes de MK distintos has calculado para el ft03?
     - Analiza cómo son las planificaciones de las distintas soluciones calculadas para el problema ft03. Pega las imágenes en la hoja correspondiente de la Excel "ResultadosEjecuciones_Individuales" y responde a las cuestiones que en ella se plantean.
- Para la instancia jss01, hoja "JSS01"
  1. Ejecuta el planificador básico y el G&Y. Para ello, ejecuta el script: Ejecuta10_jss01_RR.bat
     - ¿Cuántas soluciones de costes de MK distintos has calculado para el jss01?
     - Analiza cómo son las planificaciones de las distintas soluciones calculadas para el problema jss01. Pega las imágenes en la hoja correspondiente de la Excel "ResultadosEjecuciones_Individuales" y responde a las cuestiones que en ella se plantean.

Practiquemos con el código de los SGS

Trabajamos con la Excel "ResultadosEjecuciones_Grupo"

- Ejecuta las instancias ft03, ft06 y jss01 con todas las reglas y completa la hoja "FT03-FT06-JSS01"
  1. Para ellos emplea los scripts:
     - Ejecuta10_ft03_AllRules.bat, Ejecuta10_ft06_AllRules.bat, Ejecuta10_jss01_AllRules.bat
  2. Registra los resultados en la hoja correspondiente.
- Ejecuta las instancias LA01-LA05 con todas las reglas y completa la hoja "LA01-LA05"
  1. Para ello puedes emplear los scripts:
     - Ejecuta_la01-la05_RR_MK.bat
     - Ejecuta_la01-la05_SPT_MK.bat, Ejecuta_la01-la05_LPT_MK.bat
     - Ejecuta_la01-la05_MWR_MK.bat, Ejecuta_la01-la05_LWR_MK.bat
     - Ejecuta_la01-la05_MOR_MK.bat, Ejecuta_la01-la05_LOR_MK.bat
  2. Registra los resultados en la hoja "LA01-LA05".
  3. Analiza los resultados de la hoja "Resumen Resultados" y reflexiona sobre las cuestiones propuestas.