Planificación basada en restricciones para una agenda diaria

Federico Becoña
Lorena Ferrando
Rodrigo López
Valentina Palladino
Julio Suaya

Teoría de la Computación
Ingeniería en Informática
Universidad Católica del Uruguay

ABSTRACT

Scheduling problems can be found in a wide variety of situations ranging from deciding the order in which jobs should be executed on a CPU in a multitasking operating system to assigning nurses to shifts.

This document studies the particular scheduling problem of optimizing a day`s schedule by defining an objective function, proposing two solutions, making Python simulations for both of them and finally comparing the results obtained to conclude which one is better optimizing the defined function.

I. ESTADO DEL ARTE

A. Programación con restricciones

La programación por restricciones es un paradigma utilizado en la resolución de muchos problemas de diversas áreas entre los cuales se incluyen los de planificación y scheduling.

Dentro de ellos se encuentran los problemas de optimización (Constraint Optimization Problem, COP) en los que se busca obtener una solución que optimice una función objetivo. Esta función se aplica sobre una posible solución del problema y devuelve un valor que indica que tan buena es esa solución. Este valor determina el beneficio de la solución y es lo que se desea maximizar o minimizar.

Resolver un problema mediante programación con restricciones implica dos etapas. En la primera se debe modelar el problema, definir las variables, el dominio para cada una de ellas y las restricciones que las relacionan mediante una función objetivo y en la segunda se aplica un mecanismo de resolución para conseguir la solución requerida para optimizarla.

B. Scheduling

El scheduling comprende problemas en los que se pide determinar un plan de ejecución para un conjunto de tareas disponiendo de ciertas máquinas de procesamiento paralelo.

En los problemas de scheduling se tiene un conjunto J de n tareas numeradas 1...n. Cada tarea j de J requiere unas ciertas unidades de tiempo en ser procesada que se determinan según p j. Si luego de que una tarea comience su ejecución no puede ser interrumpida se dice que se trata de un scheduling no preemptivo, de lo contrario, si se puede interrumpir y luego continuar una tarea el scheduling es preemptivo.

En un schedule S para tareas definidas en un conjunto J se define C s j como el valor que determina el momento en el que se termina la tarea j dentro del plan de ejecución S.

La finalidad de un problema de scheduling es obtener una solución de un plan de ejecución que cumpla con todas las características propias del problema, ejemplos pueden ser precedencias entre las tareas o que no se pueda realizar una cierta tarea hasta un cierto momento, y que sea óptima conforme a la función objetivo.

La función de optimización para la resolución de un problema está determinada por las variables que definamos y el peso que le demos a cada una. Hay problemas en los que las tareas tienen fechas y/u horas de caducidad y para ellos, por ejemplo, se podría querer obtener la mayor cantidad de tareas a tiempo o el menor retraso posible todas las tareas. Otro caso son los problemas en los que no todas las tareas tienen igual importancia y se podría pretender finalizar el mayor número de tareas importantes a tiempo.

C. *Single-machine*

En este caso particular existe una sola máquina de procesamiento que puede procesar sólo una tarea al mismo tiempo.

II. HERRAMIENTAS

En la actualidad existen diferentes soluciones de software para problemas de scheduling tales como las siguientes:

IBM ILOG CP Optimizer: Es una herramienta basada en programación con restricciones que calcula soluciones a problemas de planificación y optimización combinatoria.

Google OR-Tools: Es una herramienta de código abierto utilizada para resolver todo tipo de problemas de optimizacion tanto comerciales como academicos (job shop problem, employee scheduling, etc)

III. PROBLEMA

Cuando comenzamos cada día tenemos un listado de tareas para realizar. Por ejemplo: "Enviar Mail por Proyecto", "Programar algoritmo genético para IA Pco1", "Sacar a pasear al perro a las 17.00", "Investigar p1 del obligatorio de Teoría", etc.

Cada tarea tiene al menos las siguientes características:

- Tiempo que lleva realizar la tarea (es el tiempo que me insume hacerla, medida en minutos, También puede asociarse a la complejidad, si hago una correspondencia entre la complejidad y el tiempo que lleva).
- Prioridad (es el orden que le doy entre todo lo que tengo para hacer).
- Urgencia (puede confundirse con prioridad 1, pero es algo que tengo que hacer ya).
- Fecha/Hora de caducidad (puede tenerla o no).
- Tipificación (por ejemplo si son cosas personales o laborales).

Así que a la mañana, antes de comenzar a trabajar hacemos un listado de todas las cosas que tenemos en el día (si quieren agrupadas por tipo) y luego las ordenan según los criterios de arriba. Además comienzan a trabajar en el orden que impusieron. Si alguna tarea queda sin realizar al finalizar el día, quedará para el siguiente (y es probable que un buen algoritmo "envejezca esa tarea", por ejemplo aumentando un poquito la prioridad).

IV. MODELADO

A. *Single Machine*

Para el armado de los horarios de un día para una persona se puede pensar el problema como que existe una única unidad de procesamiento que solo puede realizar una actividad a la vez y un conjunto de tareas a ejecutar. De esta forma se puede realizar un modelado mediante un problema de scheduling single machine.

B. *Definiciones*

Para una tarea j:

- Urgencia (u_j): 1 si es urgente y 0 si no.
- Prioridad (p_j): Escala del 1 al 3.
- Envejecimiento (e_j): Aumenta 1 por cada día de retraso.
- Importancia (i_j): Es la suma de la prioridad y del envejecimiento con un máximo de 3.
- Tipificación (t_j);
    - Médica: t_j=3.
    - Laboral: t_j=2.
    - Personal: t_j=1.
- Duración (d_j): Tiempo que requiere para ser procesada por la persona.
- Hora de caducidad (hf_j): Hora hasta la cual se puede comenzar la tarea.
- Hora de comienzo (hi_j): Hora explícita a la cual se debe comenzar la tarea (como en el caso de pasear el perro a las 5 p.m).
- a_j: Vale 1 si hi_j no es cero y j fue asignada el schedule armado sino 0.
- z_j: Vale 1 si hf_j no es cero y j fue asignada al schedule armado sino 0.

C. *Función de optimización*

El objetivo es maximizar el peso total resultado de la suma de todas las j tareas de una agenda con ponderaciones según se muestra en la siguiente función:

$$\Sigma\ 15\ u_j + 10\ i_j + 2\ t_j + 2\ a_j + 2\ z_j$$

V. SOLUCIONES

A. FIFO (First In First Out)

A partir de una lista con N tareas se irán colocando en el orden en el que fueron insertadas en la cola.

1. Si la tarea no tiene hi_j ni hf_j se coloca en el primer espacio libre en el que quepa.
2. Si la tarea no tiene hi_j pero si hf_j:
    - Si puedo la asigno a continuación de la última tarea insertada.
    - Si no puedo asignarla antes de su hf_j la paso para el otro día y se aumenta su valor de envejecimiento.
3. Si la tarea tiene hi_j:
    - Si esa hora no está ocupada la asigno en ese horario.
    - Si no puedo asignarla la paso para el otro día.

B. MLQ (Multiple Level Queue)

Existen las siguientes colas:

1. Urgente: Dentro de esta se ordena primero por prioridad y luego por tipificación ambas de forma descendiente.
2. No urgente prioridad 3.
3. No urgente prioridad 2.
4. No urgente prioridad 1.

    Dentro de cada cola no urgente se ordena según tipificación descendente.

Para cada tarea, por cada día de envejecimiento se le suma un nivel de prioridad con un máximo de 3.

Urgente	Prioridad Max	...	Prioridad Min
No Urgente Prioridad Max	TareaX	TareaY	...
...	...	...	...
No Urgente Prioridad Min	...	...	...

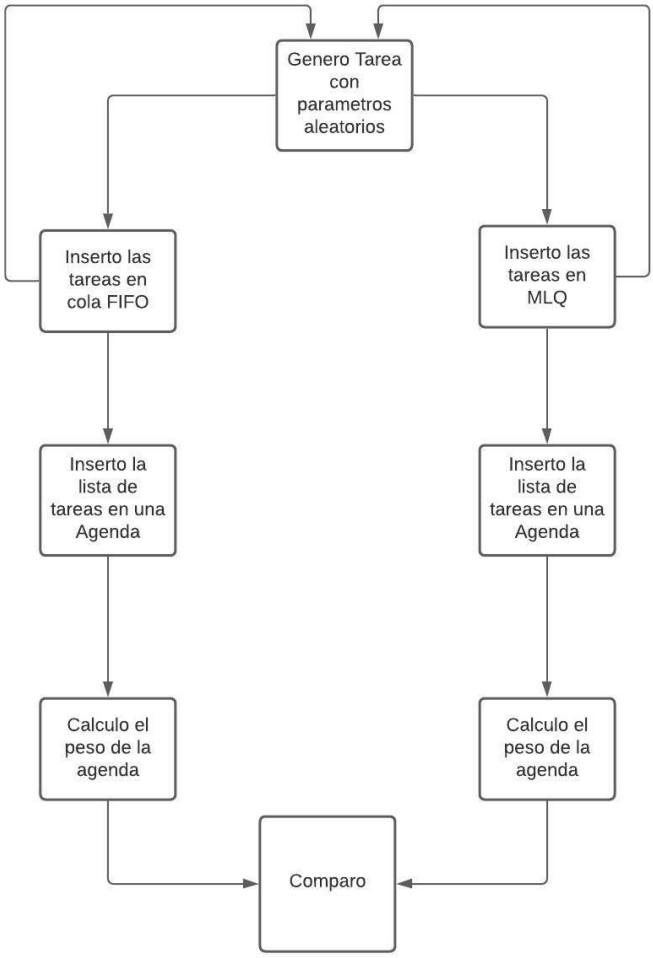
Al recibir una tarea se actuará de la siguiente manera:

1. Si la tarea no tiene hi_j ni hf_j se coloca en el primer espacio libre en el que quepa.
2. Si la tarea no tiene hi_j pero si hf_j:
    - Si puedo la asigno a continuación de la última tarea insertada.
    - Si no puedo asignarla antes de su hf_j la paso para el otro día.
3. Si la tarea tiene hi_j:
    - Si esa hora no está ocupada la asigno en ese horario.
    - Si no puedo la paso para el otro día.

VI. EXPERIMENTO

Se realizó un programa en Python para investigar cuál de las dos soluciones optimiza mejor la función objetivo definida. Para ello se crea un set de tareas aleatorias y para ese conjunto se arman dos calendarios, uno utilizando una cola FIFO y otro en base a una MLQ tal y como se describe cada solución en la sección anterior.

Representación del flujo del programa



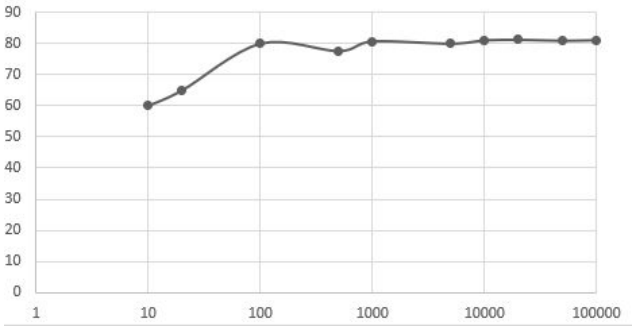
VII. MEDIDAS

Se realizaron sets de simulaciones de distintos tamaños. En cada caso se crea un conjunto de tareas aleatorias, a partir de ellas se hace un calendario para cada solución propuesta y se mide su peso según la función a optimizar.
En la siguiente imagen se observan los porcentajes de éxito de las diferentes soluciones en base a cual maximiza mejor la función objetivo en los distintos casos que componen los tres conjuntos de datos.
Encontramos que el número de veces en que la solución con MLQ es mejor a la FIFO se aproxima a un 80%.

```
Casos simulados: 10
Casos en los que MLQ es mejor: 6 casos - 60.0% del total
Casos en los que FIFO es mejor: 4 casos - 40.0% del total

Casos simulados: 20
Casos en los que MLQ es mejor: 13 casos - 65.0% del total
Casos en los que FIFO es mejor: 7 casos - 35.0% del total

Casos simulados: 100
Casos en los que MLQ es mejor: 80 casos - 80.0% del total
Casos en los que FIFO es mejor: 20 casos - 20.0% del total

Casos simulados: 500
Casos en los que MLQ es mejor: 388 casos - 77.6% del total
Casos en los que FIFO es mejor: 112 casos - 22.4% del total

Casos simulados: 1000
Casos en los que MLQ es mejor: 807 casos - 80.7% del total
Casos en los que FIFO es mejor: 193 casos - 19.3% del total

Casos simulados: 5000
Casos en los que MLQ es mejor: 3999 casos - 79.98% del total
Casos en los que FIFO es mejor: 1001 casos - 20.02% del total

Casos simulados: 10000
Casos en los que MLQ es mejor: 8103 casos - 81.03% del total
Casos en los que FIFO es mejor: 1897 casos - 18.97% del total

Casos simulados: 20000
Casos en los que MLQ es mejor: 16243 casos - 81.215% del total
Casos en los que FIFO es mejor: 3757 casos - 18.785% del total

Casos simulados: 50000
Casos en los que MLQ es mejor: 40466 casos - 80.932% del total
Casos en los que FIFO es mejor: 9534 casos - 19.068% del total

Casos simulados: 100000
Casos en los que MLQ es mejor: 81106 casos - 81.106% del total
Casos en los que FIFO es mejor: 18894 casos - 18.894% del total
```

Se puede observar que existe una convergencia asintótica de la proporción de casos en los cuales la solución con MLQ da un resultado más óptimo hacia un 80%.

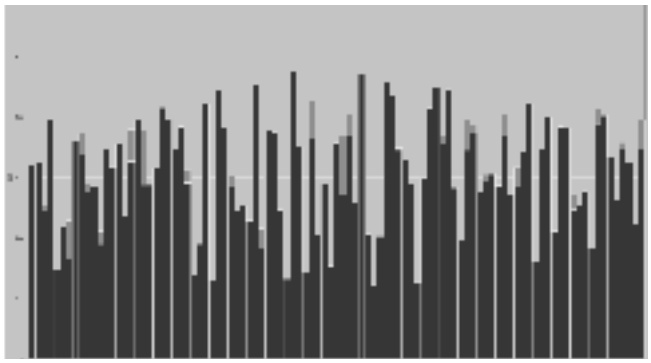
Convergencia asintótica



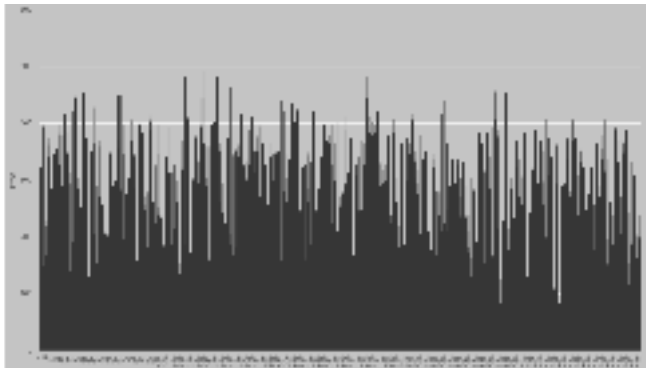
Gráficas que muestra para distinta cantidad de casos cuál solución da un mejor calendario en cada uno.

(En oscuro si es mejor MLQ y en claro si es mejor FIFO)

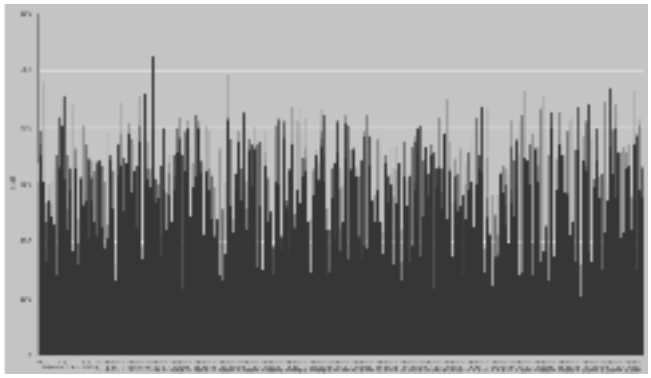
100 casos.



500 casos



1000 casos



VIII. CONCLUSIONES

Se puede obtener del experimento realizado que la solución planteada con MLQ da mejores resultados que la solución con la cola FIFO para la función de optimización definida. Además se pudo encontrar que existe una convergencia asintótica de este comportamiento al aumentar el tamaño de las muestras hacia una proporción de un 80% de los casos simulados.

IX. BIBLIOGRAFÍA

1. David Karger , Cliff Stein, Joel Wein. Scheduling Algorithms,https://people.csail.mit.edu/karger/Papers/scheduling.pdf (S,f)
2. Irene Barba Rodríguez, Algoritmos de planificación basados en restricciones para la sustitución de componentes defectuosos, http://gpd.sip.ucm.es/rafa/docencia/pr/tema1.pdf (S,f)
3. Rafael Caballero Roldán, Programación con restricciones,http://www.lsi.us.es/docs/doctorado/memorias/MemoInvestigIreneBarba.pdf (S,f)