MA4702. Programación Lineal Mixta: Teoría y Laboratorio. 2024.
Profesor: José Soto.
Profesor Auxiliar: Álvaro Márquez
Profesor Auxiliar: Paolo Martiniello

Laboratorio 3: Marco Teórico.
Agendamiento y coloreos

En este laboratorio se modelarán varias versiones de un problema de agendamiento de tareas a máquinas, y de coloreos de grafos. En este documento veremos la notación estándar para los problemas y algunos aspectos básicos a considerar previamente.

1. Agendamiento

Sea $I = \{1, 2, \ldots, m\}$ un conjunto de *máquinas* y $J = \{1, 2, \ldots, n\}$ un conjunto de *trabajos*, además sea $E \subseteq I \times J$ un conjunto tal que $ij \in E$ si y solo si el trabajo j puede ser procesado en la máquina i. Además, para cada $ij \in E$ se tiene un *tiempo de proceso* $p_{ij} \in \mathbb{R}_+$ que representa cuanto tiempo le toma a la máquina i procesar el trabajo j. Un agendamiento es una *asignación* que a cada trabajo $j \in J$ le asigna una sola máquina $i \in I$ y un tiempo de partida $S_j \in \mathbb{R}_+$. Más aún si j es asignado a la máquina i con tiempo S_j entonces j es procesado en la máquina i en el intervalo abierto de tiempo $(S_j, S_j + p_{ij})$. Normalmente llamamos $C_j = S_j + p_{ij}$ al *tiempo de completación* del trabajo j.

Decimos que un agendamiento es factible si los trabajos asignados a cada máquina $i \in I$ son procesados en intervalos disjuntos de tiempo. En muchos problemas se desea encontrar agendamientos factibles que cumplan restricciones adicionales que describiremos más abajo, y que minimicen cierta función objetivo. A continuación damos un pequeño glosario de las restricciones y objetivos típicos que aparecen en estos problemas, junto con la notación estándar involucrada.

Restricciones y notación típica

1. Tiempos de liberación (release times). Para cada $ij \in E$, se tiene un tiempo de liberación $r_{ij} \geq 0$ tal que el trabajo j no puede procesarse en la máquina i antes de este tiempo. El caso sin tiempos de liberación puede modelarse fijando todos los r_{ij} a ser iguales a 0.
2. Deadlines. Para cada $j \in J$, se tiene un tiempo límite $D_j \geq 0$ tal que el trabajo j (idealmente) debe completarse antes del tiempo D_j. Los deadlines pueden ser firmes (son restricciones que deben cumplirse) o débiles (se permite romper la restricción pero al hacerlo algo cambia en la función objetivo).
3. Restricciones de precedencia. Se tiene un orden parcial $(J, \preceq)$ tal que si j, k son dos trabajos distintos con $j \preceq k$ entonces j debe ser completamente procesado antes que empiece a procesarse el trabajo k. El caso sin precedencias puede modelarse usando el orden parcial vacío.
4. Si las máquinas son idénticas, se usan p_j y r_j para denotar el tiempo de proceso y liberación común de j.

Objetivos típicos

1. Llamamos *makespan* de un agendamiento factible al tiempo mínimo $M \in \mathbb{R}_+$ en el que las máquinas se desocupan por completo, equivalentemente es el máximo de los tiempos de completación de todos los trabajos. Un objetivo típico consiste en minimizar el makespan.

2. Dadas prioridades o pesos $w\colon J \to \mathbb{R}_+$, se puede buscar minimizar la suma de los tiempos de completación ponderados por sus prioridades: $\sum_{j\in J} w_j C_j$

3. En el caso que todas las máquinas son idénticas, se puede buscar minimizar el número de máquinas que tienen algún trabajo asignado.

4. En el caso de deadlines débiles, se llama tardanza T_j del trabajo j al máximo entre 0 y $C_j - D_j$. Se puede buscar un agendamiento factible que minimize la suma de las tardanzas o bien el máximo de las tardanzas.

2. Coloreo de grafos

Dado un grafo $G = (V, E)$ y un entero no negativo $k \in \mathbb{N}$, llamamos k-coloreamiento de G a una asignación $c\colon V \to [k]$ donde cada vértice v recibe un *color* $c(v) \in [k]$ tal que pares de vértices conectados por alguna arista reciben colores distintos. Un problema típico consiste en encontrar un coloreo con la menor cantidad de colores posibles.

3. Agendamiento/Coloreo de intervalos

Un problema que se puede modelar tanto como agendamiento como coloreo es el siguiente. Sea una colección $(s_i, t_i)_{i\in I}$, donde cada $(s_i, t_i) \subseteq \mathbb{R}$ es un intervalo abierto. Deseamos asignar cada intervalo a una máquina de modo tal que intervalos con igual asignación sean disjuntos. Se desea minimizar el número de máquinas usadas.

4. Quiebre de simetría

Uno de las dificultades computacionales más típicas de los problemas lineales enteros es la presencia de simetría. Como ejemplo, para el problema de coloreo es fácil ver que si c es un k-coloreo entonces también lo es cualquier *permutación* de los colores, es decir, cualquier función de la forma $\pi \circ c$ con $\pi\colon [k] \to [k]$ biyectiva. Otro ejemplo ocurre en agendamiento con máquinas idénticas donde permutar las máquinas resulta de otra solución con el mismo valor objetivo. Similarmente, en algunos casos (como en el problema de minimizar makespan, sin precedencias, tiempos de liberación ni deadlines) permutar el orden en el cual se procesan los trabajos en la misma máquina produce agendamientos con el mismo valor objetivo.

La presencia de simetría en un programa entero puede causar árboles de BnB innecesariamente profundos y con subárboles con exactamente la misma estructura. En otras palabras, todas las soluciones que un solver encuentra en un subárbol se encuentran repetidas en otro subárbol. Si bien es cierto algunos solvers actuales son capaces de detectar en cierta medida cuando esto ocurre (procediendo a simplificar el árbol de BnB en el camino), esto puede depender fuertemente de como se modele el problema.

Una manera simple de romper simetría es incorporar restricciones que reduzcan considerablemente la cantidad de soluciones simétricas. Por ejemplo, en el caso de coloreo en grafos, se puede asignar un peso a cada vértice del grafo y luego solicitar para todo i, que el peso total de vértices de color i sea mayor que el peso total de vértices de color j. Algo similar se puede realizar para romper la simetría entre máquinas idénticas o entre los posibles ordenes internos de trabajos asignados a la misma máquina.

Como alternativa a esto se puede buscar modelamientos que naturalmente eviten la fuente de simetría. Por ejemplo, en el caso de agendamiento con máquinas idénticas sin restricciones se puede usar variables que indiquen, para cada par de trabajos i y j, si ambos se asignan a la misma máquina, con i asignado antes que j y no hay elementos k asignados entremedio (básicamente si j es el *siguiente* trabajo asignado a i en alguna máquina).

5. Preparación para laboratorio

1. Usando como base los archivos de laboratorios anteriores, prepare una función
   `leeragendamientogeneral(nombrearchivo)`
   que sea capaces de leer y guardar apropiadamente en arreglos p, r, D, prec (como usted lo considere útil para luego modelar los problemas), la información incluido en archivos de texto que vienen con el siguiente formato:
   - La primera linea contiene 3 enteros, m n $|E|$;
   - Las siguientes $|E|$ lineas contienen información para cada uno de los $ij \in E \subseteq [m] \times [n]$ formateada como una lista de enteros, i j p_{ij} r_{ij}.
   - La siguiente linea contiene todos los deadlines en orden separados por espacios D_1 D_2 $\ldots$ D_n
   - Finalmente hay 0 o más líneas que contienen precedencias, es decir pares ordenados de enteros j k tales que j debe preceder a k.

   Como aclaración y ejemplo. Si el orden parcial de precedencia es un orden total $1 \leq 2 \leq 3$ en 3 elementos, entonces debe dar lo mismo si en las últimas lineas del archivo aparecen:

   ```
   1 2
   1 3
   2 3
   ```

   o

   ```
   1 2
   2 3
   ```

   pues ambas implican el mismo orden parcial.

2. Prepare una función
   `leeragendamientosimple(nombrearchivo)`
   que sea capaces de leer entradas para problemas de agendamiento en m máquinas idénticas (donde cualquier trabajo puede ir a cualquier máquina) con pesos en los trabajos y sin restricciones de liberación, deadline o precedencia, guardando los datos en arreglos p, w, D (como usted lo considere útil para luego modelar los problemas), la información incluido en archivos de texto que vienen con el siguiente formato:
   - La primera linea contiene 2 enteros, m n;
   - La siguiente linea contiene todos los tiempos de proceso en orden separados por espacios p_1 p_2 $\ldots$ p_n
   - La siguiente linea contiene todos los pesos en orden separados por espacios w_1 w_2 $\ldots$ w_n

3. Prepare una función
   `leergrafo(nombrearchivo)`
   que sea capaces de leer entradas para problemas de coloreo de grafos, donde $G = (V, E)$ y $V = [n]$ para algún n, guardando los datos en un formato que a usted le parezca útil, la información incluido en archivos de texto que vienen con el siguiente formato:
   - La primera linea contiene 2 enteros, $|V|$ $|E|$;
   - Las siguientes $|E|$ lineas contienen pares de vértices i j. Donde $1 \leq i < j \leq n$ e $ij \in E$.

4. Prepare una función
   `leerintervalos(nombrearchivo)`

que sea capaces de leer entradas para problemas de agendamiento de intervalos, guardando los datos en un formato que a usted le parezca útil, la información incluido en archivos de texto que vienen con el siguiente formato:

- La primera linea contiene 1 enteros: $|I|$, que es el número de intervalos
- Las siguientes $|I|$ lineas contienen pares de números $s_i \;\; t_i$; con $s_i < t_i$, representando el intervalo (s_i, t_i).