MA4702. Programación Lineal Mixta. 2020.
Profesor: José Soto
Escriba(s): Matías Azócar, Carolina Chiu y Francisco Muñoz.
Fecha: 14 de abril de 2020.

Cátedra 2

Introducción

En esta cátedra se verán dos temas principales: se comenzará dando un recorrido por la historia de los algoritmos PL/PLM a modo de motivación, para después retomar el contenido de la clase pasada y estudiar más en detalle el concepto de las *formulaciones*.

1. Historia de Algoritmos PL/PLM

¿Por qué utilizamos algoritmos de PL/PLM sobre programación no lineal o convexa? Algunas razones son:

- Permiten modelar una gran variedad de problemas de la vida real con suficiente generalidad.
- Existen buenos algoritmos (a nivel práctico y teórico) y heurísticas[1] que permiten solucionarlas.
- Han tenido éxito industrial (existen solvers bien implementados).

Orígenes de la Programación Lineal Pura

- **1827 y 1836**: *Fourier* y *Motzkin* (por separado) describen un método **finito** para sistemas de desigualdades (conocido como *esquema de eliminación de Fourier-Motzkin*).
- **1937**:*Kantorovich* (Premio Nobel 1975) desarrolla una teoría de dualidad en programación lineal para temas de economía.
- **1946**: *George Dantzig* desarrolló la base del método Simplex mientras trabajaba para la fuerza aérea de Estados Unidos. Por otro lado, *Jon Von Neumann* desarrolla la teoría de la dualidad en programación lineal. Ambos junto a Kantorovich se consideran los padres de la programación lineal.

Definición 1 (Algoritmo Polinomial)**.** Se define como *algoritmo polinomial* a un algoritmo con entrada de N bits, y que ejecuta sus instrucciones en un orden de $\mathcal{O}(p(N))$, donde p es un polinomio. Es decir, un algoritmo con tiempo de ejecución $N^{\mathcal{O}(1)}$.

Definición 2 (Algoritmo Exponencial)**.** Se define como *algoritmo exponencial* a un algoritmo con entrada de N bits, y que ejecuta sus instrucciones en un orden de $\mathcal{O}(e^N)$. Es decir, por ejemplo, un algoritmo al cual le toma $2^{\mathcal{O}(N)}$ operaciones en ejecutarse.

Ejemplo 1. Un algoritmo que se tome como máximo N^{50} operaciones en ejecutarse, es un algoritmo polinomial[2].

Ejemplo 2. El algoritmo de Kruskal es polinomial.

La gran ventaja de este tipo de algoritmos es que si el tamaño de la instancia crece, por ejemplo, se duplica, entonces el tiempo de ejecución sólo se multiplica por una constante. Por ejemplo, un algoritmo con cota N^k en su tiempo de ejecución, al ser alimentado por una instancia de tamaño $2N$, su tiempo cambia a $(2N)^k = 2^k N^k$.

En cambio, si se está trabajando con un algoritmo exponencial, al duplicar el tamaño de la instancia, la complejidad del algoritmo podría crecer demasiado. Por ejemplo, para un algoritmo con cota 2^{cN} en su tiempo de ejecución, la cota para una instancia de tamaño $2N$ se eleva al cuadrado (pues $2^{2cN} = (2^{cN})^2$).

Definición 3 (Clase P)**.** La *clase P* contiene a aquellos problemas que son solubles en tiempo polinómico.

[1] Una **heurística** es una técnica diseñada para resolver un problema más rápidamente cuando los métodos clásicos son demasiados lentos, o bien, encontrar una solución cercana al óptimo cuando los métodos clásicos fallan.
[2] Aunque no uno muy eficiente.

Ejemplo 3. El problema de calcular un matching es de clase P, pues existe una solución en tiempo polinomial.

Existe una clase de problemas, llamados NP-completos (cuya definición formal no daremos en este curso) que ocurren muchas veces en la práctica. Se conjetura (no ha sido demostrado) que todos estos problemas no están en P.

Observación 1. *La pregunta ¿$P = NP$? es aún un problema abierto y es uno de los problemas del milenio*[3].

Se han definido estos conceptos de complejidad para responder a la siguiente pregunta: ¿Qué clase de problema es PL? la pregunta tiene su importancia, pues Klee y Minty (1973) demostraron que Simplex **no es polinomial**[4], a pesar de que es muy rápido en la mayoría de las instancias.

Entonces, ¿Es PL demasiado general como para no estar en P? La respuesta es: ¡No!, en efecto en el año 1979 se demuestra que **PL está en P**.

Programación Lineal Pura hoy en día

Ha sido trabajo de los matemáticos el de reducir el tiempo de ejecución del PL. En lo que sigue, n corresponde a variables y L al número de bits que se usan para codificar la entrada.

- **1979**: *Khachiyan* desarrolla el método de la elipsoide para PL (Complejidad $\mathcal{O}(n^6 L)$).
- **1984**: *Karmakar* desarrolla el método de punto interior (Barrera) para PL (Complejidad $\mathcal{O}(n^{3,5} L)$).
- **2015**: *Lee y Sidford* desarrollan la primera mejora en 20 años para flujo en redes (Complejidad $\mathcal{O}(n^{2,5} L)$).
- **2019**: *Cohen, Lee y Song* obtienen una complejidad de $\mathcal{O}(n^{2,37} L)$ usando multiplicación rápida de matrices.

Observación 2. *Notar que una complejidad de $\mathcal{O}(n^6 L)$ es bastante alta, pero al menos es polinomial. Por otro lado, $\mathcal{O}(n^{2,37} L)$ ya es una complejidad cercana a una cuadrática, el cuál es un orden más aceptable.*

Observación 3. *En la práctica, ninguno de estos algoritmos se utiliza (ya que son muy difíciles de implementar, y algunos resultan peores que un algoritmo exponencial en regímenes pequeños). Lo que se usa son mezclas de Simplex, Simplex-dual y Barrera.*

Respondiendo a la pregunta de por qué se prefiere PL/PLM, es porque hoy en día tenemos muchos métodos de resolución a estos problemas. De forma que, se ha avanzado tanto en esta área, que programas lineales que antes tomaban 20 años de procesar, hoy en día se pueden resolver en un segundo. Esto permite utilizar millones de variables y restricciones y resolverlos en un tiempo prudente.

¿Qué pasa con PLE y PLM?

- En la década de 1960 sale el primer solver comercial.
- Las siguientes décadas se dedican a crear mejoras en la implementación de PL.
- En la década de 1990 aparecen solvers nuevos, como *CPLEX* (ILOG), *MINTO* (Gerogia Tech) y *XpressMP* (comercial).
- En el año 2007 IBM compra ILOG.
- Los creadores originales de *CPLEX* no se involucran con IBM, y en su lugar fundan *Gurobi* en el año 2009.

Hoy en día, los problemas de programación lineal entera se pueden resolver de forma rápida debido a que se han incorporado una serie de ideas (que se verán más adelante en el curso) a los solvers previamente mencionados. Algunas de ellas son:

- Aproximación sucesiva de PLE por PL.
- Agregar planos cortantes, es decir, incluir restricciones sin perder puntos factibles (mejora en la formulación).

[3] Es decir, si es que alguien demuestra o refuta la conjetura, ¡se ganará un millón de dolares!.

[4] Más precisamente, demostraron que hay instancias para los cuales Simplex debe tomar un número superpolinomial de pasos

- *Branch and Bound*: es la base de los algoritmos PLE. La idea es que si hay un conjunto finito de puntos que pueden ser solución, se pueden enumerar de manera sistemática (como árbol) y es posible darse cuenta que hay ramas del árbol que no pueden ser solución.
- Planos cortantes genéricos: cortes que se generan automáticamente (Gomory, MIR, etc.)
- Presolver (Técnicas de reducción de tamaño).
- Generación de filas y columnas; Simetría; Heurísticas; entre otras.

Probablemente PLE no está en P (se puede probar que es NP-Completo). Sin embargo, los solvers disponibles actualmente son capaces de:

- Realizar buenas aproximaciones de soluciones de PLE con millones de variables y restricciones. Estas soluciones son factibles y pueden reportar que se encuentran cerca del óptimo. (para hacer esto, no se necesita que el usuario sea un experto, sólo basta con saber ocupar dichos solvers).
- Resolver a optimalidad muchas clases de subproblemas.
- Todo eso se hace prácticamente sin intervención del usuario.

2. Formulaciones

Definición 4 (Poliedro)**.** Un *poliedro* P es un conjunto lineal puro, es decir

$$P = P(A, b) = \{x \in \mathbb{R}^n : Ax \leq b\}$$

con $A \subseteq \mathbb{R}^{n \times m}$ y $b \in \mathbb{R}^n$

- Si A y b pueden ser escogidos racionales (es decir, sus componentes pueden ser escogidas racionales): P es un **poliedro racional**.
- Si P es acortado: se llama **polítopo**.

Observación 4. *Dado que los poliedros son intersecciones finitas de semiplanos, y los semiplanos son conjuntos convexos, entonces los poliedros son conjuntos convexos.*

Definición 5 (Formulación)**.** Un poliedro P es una *formulación* para un conjunto $S \subseteq \mathbb{Z}^E \times \mathbb{R}^C$ si:

$$S = P \cap (\mathbb{Z}^E \times \mathbb{R}^C)$$

En otras palabras, una formulación es un poliedro que aproxima externamente bien a algún conjunto.

Ejemplo 4. En la Figura **??**, se puede observar un poliedro P (conjunto verde) que es una formulación de un conjunto S (puntos rojos), pues

$$S = P \cap \mathbb{Z}^E$$

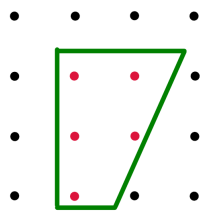

Figura 1: Primer ejemplo de una formulación.

Podemos notar que las formulaciones no son necesariamente únicas, en efecto, en el Ejemplo **??** se puede encontrar otro poliedro P', tal como se presenta en el siguiente ejemplo:

Ejemplo 5. En la imagen **??**, se puede observar un poliedro P' (conjunto azul) que es una formulación de un conjunto S (puntos rojos), pues

$$S = P' \cap \mathbb{Z}^E$$

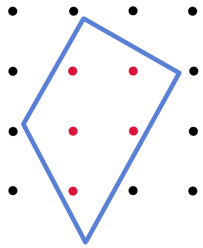

Figura 2: Segundo ejemplo de una formulación.

Dicho esto, nos podemos preguntar: ¿Cuál de todas las formulaciones ocupar? La respuesta que nace naturalmente es: ocupar la formulación minimal para S. Esto se puede lograr ocupando como formulación para S su envoltura convexa, vale decir, conv$S$. Una representación de lo anteriormente dicho se puede observar en la siguiente figura:

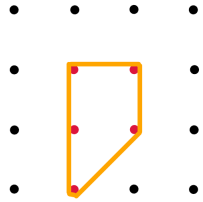

Figura 3: Formulación minimal para S: su envoltura convexa.

Formalizando lo comentado anteriormente en los ejemplos, P es buena formulación de S si es que lo *aproxima* apropiadamente. Para empezar, al menos, sabemos que debemos tener lo siguiente

$$S \subseteq P$$

- Sean $P_1 \subseteq P_2$ dos formulaciones para el mismo conjunto. En este contexto, diremos que P_1 es *mejor* que P_2.

  Posterior a esto, no es difícil pensar que el mejor de todos los casos será tomar como formulación la envoltura convexa (esto puede, y debería, ser demostrado). Justamente por eso tenemos lo siguiente

- Si $P = \text{conv}(S)$ entonces P es *formulación ideal* de S.

Observación 5. *No todos los conjuntos lineales mixtos admiten formulaciones ideales.*

En la siguiente figura se puede apreciar un ejemplo de lo anterior, pues, para la primera restricción (gráficamente, la diagonal) no hay valores que se alcancen en la frontera. Esto, pues habría que solucionar $-\sqrt{2}x + y = 0$ con $x,\ y$ enteros. Sin embargo, por la irracionalidad de $\sqrt{2}$ no habrá un punto (x, y) que resuelva tal ecuación. Lo que sucederá es que iremos tomando una cantidad infinita de puntos que se acerquen, pero eso no define un poliedro. Así, se evidencia que hay conjuntos que no admiten formulación ideal.

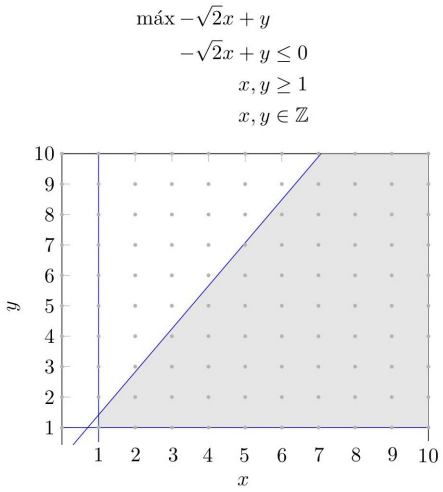

Figura 4: Ejemplo de PLM donde no existe formulación ideal.

Notemos que lo anterior no sucede para un PL puro, recordando del curso de optimización, que todo PL factible y con objetivo acotado alcanza su óptimo.

Teorema 1. *Sea (M) un PLM con datos racionales. Si su relajación lineal (P) es acotado en la dirección de optimización, entonces o bien (M) tiene óptimo finito racional y alcanzable, o bien (M) es infactible.*

Ejemplo 6 (Uncapacitated Facility Location)**.** Un problema clásico en Investigación de Operaciones es el de *Uncapacitated Facility Location*. Definimos ahora lo que se recibe como información y lo que busca hallar el problema.

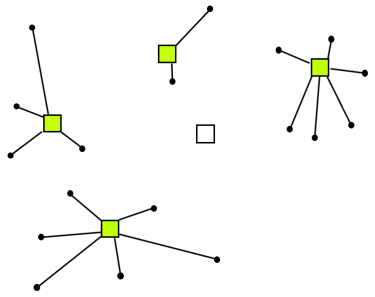

Figura 5: Visualización del problema *Uncapacitated Facility Location*.

Entrada: n clientes y m ubicaciones de bodegas. Existe un costo fijo $c_j \geq 0$, $\forall j \in [m]$ asociado a la apertura de una bodega y un costo de conexión $d_{ij} \geq 0$ por conectar a un cliente i con una bodega j, $\forall i \in [n]$, $\forall j \in [m]$.

Objetivo: Abrir bodegas y conectar clientes a bodegas abiertas incurriendo en un costo total mínimo.

Construiremos una formulación para el problema:

Para ello haremos uso de dos variables binarias:

$$y_j = \begin{cases} 1 \text{ si la bodega } j \text{ abre.} \\ 0 \text{ si no lo hace} \end{cases} \qquad x_{ij} = \begin{cases} 1 \text{ si el cliente } i \text{ conecta con la bodega } j \\ 0 \text{ si no lo hace} \end{cases}$$

De esta forma, procederemos a plantear la formulación. Queremos minimizar el costo total, así, el objetivo será:

$$\text{(FL1)} \qquad \text{mín} \sum_{j \in [m]} c_j y_j + \sum_{i \in [n]} \sum_{j \in [m]} x_{ij} d_{ij}$$

El primer término será el asociado a los costos fijos de las bodegas y el de la derecha el costo de conexión cliente-bodega total.

Ahora, las restricciones serán las siguientes:

$$\begin{aligned} s.a. \quad & \sum_{j\in[m]} x_{ij} = 1 && \forall i \in [n] \\ & \sum_{i\in[n]} x_{ij} \leq n \cdot y_j && \forall j \in [m] \\ & x_{ij},\ y_j \in \{0,1\} && \forall i \in [n]\ \forall j \in [m] \end{aligned}$$

La primera restricción es la que indica que de todas las bodegas, el cliente *debe* estar asociado a una. (Podría ser reemplazado por la restricción de estar asociado al menos con una). La segunda viene de la siguiente situación:

- Si la bodega no abre ($y_j = 0$) no pueden haber clientes asociados a esa bodega, luego, como los x_{ij} son no negativos, deben ser todos cero.
- Si la bodega abre ($y_j = 1$) entonces pueden haber como máximo n clientes asociados a esa bodega (pues ese es el número total de clientes)

La tercera restricción viene del hecho de que definimos x_{ij} y y_j como variables binarias.

El objetivo junto a las tres restricciones definen la formulación que hemos construido.

Ejemplo 7 (Problema de Asignación)**.** Un problema al que probablemente usted ha podido enfrentarse en Algoritmos Combinatoriales es el *problema de asignación.*

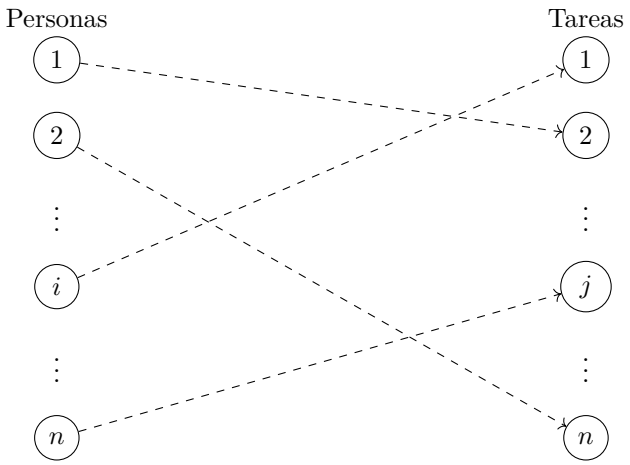

Figura 6: Problema de Asignacion.

Entrada: n tareas y n personas. Cada persona debe realizar *exactamente* una tarea. c_{ij} es el costo incurrido por la persona i al realizar la tarea j.

Objetivo: Encontrar la asignación de costo total mínimo.

Construiremos una formulación para el problema:

Para ello haremos uso de una variables binaria:

$$x_{ij} = \begin{cases} 1 \text{ si la persona } i \text{ realiza la tarea } j \\ 0 \text{ si no lo hace} \end{cases}$$

Así, procedemos. Queremos minimizar el costo total, así, el objetivo será:

$$(A) \qquad \text{mín} \sum_{i\in[n]} \sum_{j\in[n]} c_{ij} x_{ij}$$

El término proviene de sumar los costos de las personas (suma sobre i) asociadas a las tareas (suma sobre j), los cuales contaremos solo si la persona i realiza la tarea j.

Ahora, las restricciones serán las siguientes:

$$\begin{aligned} s.a. \quad & \sum_{j\in[n]} x_{ij} = 1 \quad \forall i \in [n] \\ & \sum_{i\in[n]} x_{ij} = 1 \quad \forall j \in [n] \\ & x_{ij}, \in \{0,1\} \quad \forall i \in [n] \ \forall j \in [n] \end{aligned}$$

El objetivo junto a las tres restricciones definen la formulación que hemos construido.

Ahora, definiremos lo siguiente:

- S: Dominio de (A)
- D: Relajación de (A)

Gracias a un importante resultado de Birkhoff y Von Neumann sabemos que todo punto en P es combinación convexa de puntos enteros (en S), Luego, P es ideal.

Para finalizar la clase, se presenta un caso interesante.
Sea (M) un PLE (totalmente) acotado, con dominio S. Luego S es finito. ¿Tiene formulación ideal?

$$\begin{aligned} S &= \{s_i\}_{i=1}^K \\ \text{conv}(S) &= \{x = \sum_{i=1}^K \lambda_i s_i : \lambda_i \geq 0, \ \sum_{i=1}^K \lambda_i = 1\} \end{aligned}$$

Surge así la pregunta... ¿La envoltura convexa de un conjunto finito de puntos es un poliedro? (esto puede ser respondido a través del estudio de teoría poliedral).