Programación Entera y Combinatorial: Resolviendo MIP's

IN77O

Universidad de Chile, DII

7 de marzo de 2011

Contenidos

Una vuelta por la Historia

- Primeros antecedentes en el problema de diseño de grupos de batalla óptimo para la marina.
- Gomory [1958] presenta el primer algoritmo para IP.
  - El método converge en un número finito de pasos.
- Primer método de B&B fue propuesto por Land y Diog [1960].
- Little [1963] lo usó en el contexto del TSP, y el método cobró relevancia como herramienta práctica.
- balas [1965] presentó el primer algoritmo de enumeración implícita para *IP* con variables binarias.

Definiendo el problema

El problema:

Consideramos problemas del tipo $\max\{cx : x \in S\}$ donde S es un conjunto polihedral con restricciones de integralidad, y para el cual queremos encontrar una solución óptima o ε-óptima.

- S puede escribirse de la forma $\{x \in \mathbb{R}^n : Ax \leq b,\ x_i \in \mathbb{Z},\ \forall i \in I\}$, y donde $I \subseteq \{1, \ldots, n\}$.
- $A \in \mathbb{Q}^{n \times m}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$.
- Dado lo anterior, podríamos restringirnos a $x \in \mathbb{Q}^n$.
- ¿Por qué?

Definiendo el problema

- ¿Qué significa obtener una solución ε-óptima?
  - Un punto $x \in S$ se dice factible.
  - El valor de la función objetivo de un punto factible x_o se denota Z_{x_o}.
  - El valor óptimo para un problema se denota Z_{ip}.
  - Un punto factible x_o es una solución ε-óptima si

$$|Z_{ip} - Z_{x_o}| \leq \varepsilon |Z_{ip}|$$

- ¿Qué tipo de algoritmos podemos esperar?
  - Recordemos que $3 - SAT \leq_p IP \leq_p MIP$, por lo que es difícil que exista algún algoritmo polinomial para resolverlo en general.
  - Existen casos especiales que se resuelven eficientemente? (polinomialmente).

Un Algoritmo genérico

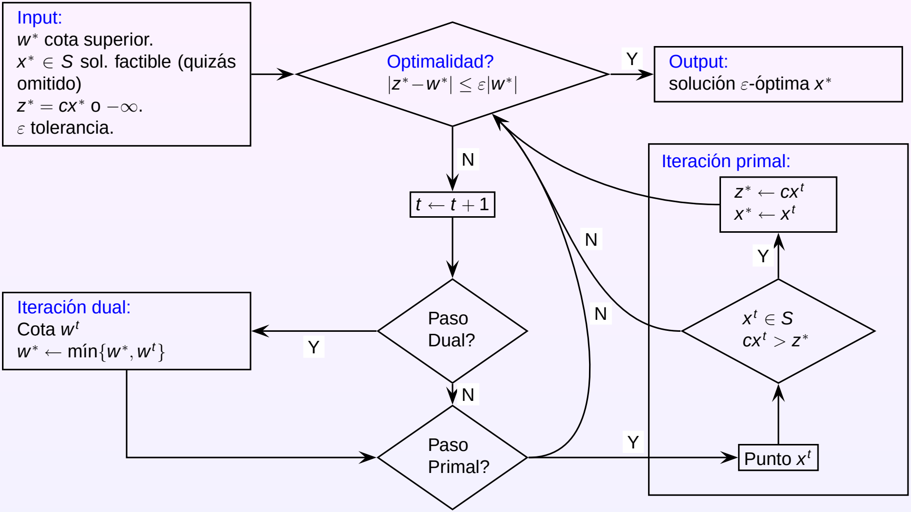
- Para resolver un MIP genérico, necesitamos una solución factible x_o y un número w_o tal que:
  - $w_o \geq Z_{ip}$, $Z_o = cx_o$, $|w_o - Z_o| \leq \varepsilon |w_o|$.
  - Mejoras en el valor de Z_o se llaman pasos primales, mejoras en el valor de w_o se llaman pasos duales.
- ¿Cómo podemos encontrar valores w_o satisfaciendo $Z_{ip} \leq w_o$?

Definición (Relajación)

Dado $S \subseteq \mathbb{R}^n$, decimos que S' es una *relajación* de S si $S \subsetneq S'$.

- Si $w_o = \max\{cx : x \in S'\}$ es fácil de resolver, podemos usar el valor como una cota.

Un Algoritmo genérico



Algoritmo de relajaciones

- La mayoría de los algoritmos se centra en torno a los pasos duales, así que explicaremos en mas detalle este paso y condiciones deseables para el.
  - inicialización: $t \leftarrow 1$, $w^* \leftarrow \infty$, $z^* \leftarrow -\infty$, Seleccionamos S_R^t una relajación de S donde $Z_R^t \geq cx \forall x \in S$.
  - Paso 1: Resolvemos $Z_R^t = \max\{Z_R^t(x) : x \in S_R^t\}$, y sea x^t su solución óptima.
  - Optimalidad: Si $x^t \in S$ y $Z_R^t = cx^t$ entonces $w^* = cx^t = z^*$ y terminamos.
  - Refinamiento: Sea $w^* = Z_R^t$, si $x^t \in S$ guardamos $z^* = cx^t$, $t \leftarrow t+1$, y escogemos S_R^t tal que $S \subseteq S_R^t \subseteq S_R^{t-1}$ y escogemos $Z_R^t(x)$ tal que $cx \leq Z_R^t(x) \leq Z_R^{t-1}$ para $x \in S$.
  - Iterar: Vuelve a Paso 1.

Algoritmo de relajaciones

- Nótese que en general necesitaremos $S_R^t \neq S_R^{t-1}$ o $Z_R^t(x) \neq Z_R^{t-1}(x)$.
- Si escogemos $Z_R^t(x) = cx$ para todo t, necesitamos que $S_R^t \subseteq S_R^{t-1} \setminus \{x^{t-1}\}$.
- Un ejemplo de este tipo de algoritmos es el *Fractional cutting-plane algorithm* (FCPA).
  - En cada iteración se escoge $Z_R^t(x) = cx$, y S_R^1 es la relajación lineal de S
  - Se escoge $S_R^t = S_R^{t-1} \cap \{x : \pi x \leq \pi_o\}$ donde $S \cap \{x : \pi x \leq \pi_o\} = S$ y donde $\pi x^{t-1} > \pi_o$.
  - Podemos usar la solución (dual) de PR^{t-1} para construir una solución factible (dual) de PR^t.
  - Usamos el algoritmo de simplex dual!

Algoritmo de relajaciones

- Otra alternativa es usar un enfoque enumerativo.
  - Sea $\mathcal{L} = \{S^i : i = 1, \ldots, m\}$ tal que $S \subseteq \bigcup(S^i : i = 1, \ldots, m)$, y $S^i \cap S^j = \emptyset$.
  - Definimos $P^i = \max\{cx : x \in S^i\}$, claramente $Z_{ip} \leq \max\{Z_{P^i} : i = 1, \ldots, m\}$.
  - Si $S = \bigcup(S^i : i = 1, \ldots, m\}$ entonces $Z_{ip} = \max\{Z_{P^i} : i = 1, \ldots, m\}$.
  - La idea es usar *dividir para reinar*.
  - En cada paso se subdivide algún S^i en sub conjuntos cuya unión es (o contiene estrictamente) S^i.
  - Esto define un árbol (o jerarquía) de problemas.
  - Esto incluye a enumeración completa.
  - Si no es necesario sub-dividir (explorar) S^i, decimos que S^i puede ser eliminado (pruned).

¿Cuándo podemos eliminar un nodo?

- Si $S^i = \emptyset$.
- Si una solución óptima es conocida para Z_{ip}.
- Si $z^* \geq z_{P^i}$.
- ¿Cómo sabemos que $z_{P^i} \leq z^*$?
  - Basta resolver la relajación lineal PR^i de P^i.
  - Si $z_{P^i} \leq z_{PR^i} \leq z^* \leq z_{ip}$.
  - Si $PR^i = \emptyset$ podemos eliminar S^i.
  - Si $\exists x_R^i \in S^i \cap S$ con $z_{PR^i} = cx_R^i$.
- Sea DP^i un problema (débilmente) dual de P^i, podemos eliminar S^i si
  - Si z_{DP^i} no esta acotado por debajo.
  - si $z_{DP^i} \leq z^*$.

Un Ejemplo

- Consideremos

$$\begin{array}{rrcl} z_{ip} = \text{máx} & -100x_1 + 72x_2 + 36x_3 & & \\ s.t. & -2x_1 + x_2 & \leq & 0 \\ & -4x_1 + x_3 & \leq & 0 \\ & x_1 + x_2 + x_3 & \leq & 2{,}9 \\ & x \in \{0,1\}^3 & & \end{array}$$

- Particionaremos S en x_1, luego en x_2 y finalmente en x_3.
- ¿Cómo podemos usar las reglas anteriores si usamos PR^i?

Branch & Bound

1: Inicialización: $\mathcal{L} = \{IP\}$, $S^o = S$, $w^* = \infty$ y $z^* = -\infty$.
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: Seleccionar $S^i \in \mathcal{L}$, resolvemos PR^i, sean z_{PR^i} y x^i_R su solución y valor, $\mathcal{L} \leftarrow \mathcal{L} \setminus \{S^i\}$.
4: $w^* \leftarrow \max\{z_{PR^i} : P^i \in \mathcal{L}\}$.
5: Si $|w^* - z^*| \leq w^*\varepsilon$ **goto** 11.
6: Si $z_{PR^i} \leq z^*$ **goto** 2.
7: Si $x^i_R \notin S^i$ **goto** 10.
8: Si $cx^i_R \geq z^*$ $z^* \leftarrow cx^i_R$.
9: **goto** 2
10: Sea $\{S^{ij}\}$ una división de S^i, $\mathcal{L} \leftarrow \mathcal{L} \cup \{S^{ij}\}$.
11: **return** solución ε-óptima x^*, z^*, w^*.

Reglas de prunning

1. Si $S^i_{LP} = \emptyset$ (infactibilidad) podemos eliminar nodo P^i.
2. Si $x^i \in S$, guardamos nueva solución factible x^i, definimos $z^* = \max\{z^*, cx^i\}$ y eliminamos el nodo del árbol de búsqueda.
3. Si $z^i_{LP} \leq z^*$ podemos eliminar nodo P^i por dominancia de valores.
   - Si resolvemos LP^i por un algoritmo dual, podemos eliminar P^i antes de resolver P^i_{LP} completamente.
   - Un método dual factible sólo decrece el valor objetivo, y si esta cota baja de z^*, eliminamos P^i.
   - En algunos casos se usan criterios de dominancia relajada, como que $z^i_{LP} \leq z^* + \varepsilon$ para asegurar que la siguiente solución es al menos mejor por una constante ε de la ya encontrada.

Divisiones de P^i

- ¿Cómo dividimos $S = \{x \in \mathbb{R}^n : Ax \leq b\}$?
- La relajación es LP, por lo que división requiere agregar restricciones lineales.
- Una forma común es dado S definir S^1, S^2 tales que $S = S^1 \cup S^2$, donde $S^1 = \{x \in S : dx \leq d_o\}$ y $S^2 = \{x \in S : dx \geq d_o + 1\}$, con $(d, d_o) \in \mathbb{Z}^{n+1}$.
- Si x^0 es la solución de $z^o_{LP} = \max\{x \in \mathbb{R}^n : Ax \leq b\}$, se escoge (d, d_o) tal que $d_o < dx^o < d_o + 1$.
- Esta elección asegura que $x^o \notin S^1 \cup S^2$ y posibilita que $z_{LP^1} < z_{LP}$ y que $z_{LP^2} < z_{LP}$.

Divisiones de P^i

- En la práctica sólo algunas elecciones de (d, d_o) son usadas:
  1. Dicotomía: $d = e_j$ para x_j^o fraccionario, y $d_o = \lfloor x_j^o \rfloor$.
     - Sólo cambiamos cotas superiores/inferiores.
     - Los tamaños de las bases subsecuentes no cambian!
  2. Dicotomía en GUB:
     - GUB: restricción del tipo $\sum x_i : i \in I = 1$, para $I \subset I^{\mathbb{Z}}$.
     - Definimos I^1, I^2 tal que $I^1 \cap I^2 = \emptyset$, $I^1 \cup I^2 = I$ y ambos conjuntos tienen una variable fraccionaria, entonces definimos $S^i = \{x \in S : \sum x_i : i \in I^i = 0\}$ para $i = 1, 2$.
     - Esto asegura que $x^o \notin S^i$.
  3. Si x_j^o es fraccionario, y $l_j \leq x_j \leq u_j$, creamos $S^i = \{x \in S : x_j = i\}$ para $i = l_j, \ldots, u_j$.
     - En la práctica esta división múltiple no se utiliza.
     - Se obtiene como dicotomía en variables repetida.

Finitud de B&B

Teorema

Dado $P = \{x \in \mathbb{R}^n : Ax \leq b, x^i \in \mathbb{Z}, \forall i \in I\}$ acotado. Un árbol basado en dicotomía en variables fraccionarias será finito. De hecho, cada rama tiene un largo acotado por $\sum_{i \in I} w_i$ donde w_i es el ancho (entero) del intervalo para $x_i, \forall i \in I$.

Demostración

Por inducción

- El resultado es cierto incluso para P no acotados, pero la demostración de esto vendrá más adelante.

Importancia de la relajación y soluciones factibles:

Nota:

Si $z_{LP^i} > z^*$, entonces el problema P^i no puede ser eliminado del árbol de B&B.

- Ésto implica que el tamaño del árbol depende fuertemente de la calidad de la formulación.
- Análogamente, en la medida que se tienen mejores soluciones factibles, el proceso de eliminación por dominancia ayuda a reducir el tamaño del árbol de B&B.

Selección de P^i (Node selection)

Dado $\mathcal{L}$ ¿Cuál sub-problema analizamos en detalle?

Los nodos en $\mathcal{L}$ se llaman activos. Las reglas se pueden agrupar en dos grandes clases: reglas a priori (donde las decisiones se toman por adelantado) y reglas adaptivas (donde la selección dependerá del árbol que estamos examinando).

Selección de P^i (Node selection)

- Depth-first search plus backtracking (DFS+BK), o conocida como last in first out (LIFO):
  - En DFS, cuando un nodo no es eliminado, el siguiente nodo a explorar será uno de sus hijos.
  - BK significa que cuando un nodo es eliminado, el siguiente nodo a explorar se encuentra siguiendo el camino hacia la raíz hasta que encontramos un nodo con un hijo sin explorar.
  - Si siempre escogemos el hijo izquierdo, entonces tenemos una regla a priori pura.
  - Cuando pasamos de un nodo padre a un nodo hijo, re-optimización por simplex dual es más eficiente (no se necesita re-construir información ni refactorizar).
  - En muchos casos, las soluciones óptimas se encuentran en lo profundo del árbol.

Selección de P^i (Node selection)

- breath-first search (BFS)
  - Todos los nodos en un mismo *nivel* del árbol de B&B son examinados antes de pasar al siguiente nivel.
- Best Bound:
  - La regla se reduce a escoger $P^i \in \mathcal{L}$ que maximiza z_{LP^i}.
  - Tiene como ventaja que estamos seleccionando un nodo que tenemos que explorar en forma obligada (¿por que?).
  - Muchas veces se usa como una alternativa a backtracking.
  - Permite mejorar la cota dual en la siguiente iteración.

Selección de P^i (Node selection)

- Best Estimate:
  - La regla es escoger el nodo que más probablemente tiene una solución óptima.
  - Si tenemos buenas estimaciones, o buenas cotas, permite que podamos podar más efectivamente el árbol de B&B.
  - Si $\hat{z}_{P^i}$ es una estimación de la mejor solución factible en P^i, escogemos P^i que maximiza $\hat{z}_{P^i} : P^i \in \mathcal{L}$.
- Quick improvement:
  - La idea es mejorar la solución factible rápidamente.
  - Un criterio común es $\max\{\frac{z_{RP^i} - z^*}{z_{RP^i} - \hat{z}_{P^i}} : P^i \in \mathcal{L}\}$.
  - Nótese que preferimos nodos con $z_{RP^i} - \hat{z}_{P^i}$ pequeño.

Selección de ramificación (branching)

Cuál es el Problema?

Dado un problema P^i para el cual conocemos $x^*_{PR^i} \notin P^o$ y $z_{PR^i} > z^*$, aún si nos restringimos a divisiones por dicotomía (en variables o en GUB), la pregunta es cuál de todas tomar.

- Usualmente hay algunas pocas variables claves en los problemas de MIP.
- Escoger branchear en ellas al comienzo del algoritmo tiene impacto profundo en el tiempo de ejecución.
- Software comerciales permiten definir prioridades de brancheo para las variables (basadas en experiencia del usuario y conocimiento del modelo real).

Selección de ramificación (branching)

- ¿Qué esperamos obtener cuando brancheamos?
  - Sólo se realiza cuando nodo no puede ser eliminado.
  - Esperamos obtener una mejora en w^*
  - Esperamos obtener problemas más sencillos.
  - Eliminar alguno de los posibles sub-problemas.
- Enfoques clásicos:
  - First fractional variable: seleccionamos primera variable fraccionaria, es prioridad por orden lexicográfico.
  - Most fractional: seleccionamos variable cuya parte fraccionaria ($x_i = \lfloor x_i \rfloor + f_i$) f_i sea más próxima a $\frac{1}{2}$.
  - Las reglas anteriores son naturales, pero **muy** pobres en la práctica.

Selección de ramificación (branching)

- Otro enfoque:
  - Supongamos que queremos mejorar w^*.
  - Supongamos que tenemos todo el tiempo del mundo y tenemos variables binarias.
  - Llamemos P problema a ramificar, y P_i^o, P_i^1 a los sub-problemas obtenidos por branchear en la variable i a cero y uno respectivamente.
  - Entonces quisiéramos escoger variable x_i minimizando $B_i = \max\{z_{PR_i^o}, z_{PR_i^1}\}$.
  - Problema regla anterior, es que puede no ver que una de las ramas mejora tanto que sobrepasa z^*.
  - Podemos definir $\alpha, \beta \in \mathbb{R}^+$ tales que minimizamos $B_i = \alpha \min\{z_{PR_i^o}, z_{PR_i^1}\} + \beta \max\{z_{PR_i^o}, z_{PR_i^1}\}$.

Strong Branching

- Ideas anteriores se conocen como *Strong Branching*
  - Mejor relajación escogida, mejores valores obtenidos.
  - En general $PR = LP$, pero en algunos casos se usa LP + generación de cortes en cada sub-problema.
  - Basta considerar variables enteras fraccionarias.
  - Simple extender idea a variables enteras generales.
  - Minimiza número de nodos visitados en B&B.
  - Tiempo de ejecución puede ser muy alto.
  - En su forma más pura, se ha utilizado con mucho éxito en el TSP.
  - Disponible en software comerciales (variaciones).
  - ¿Cuáles son las variaciones naturales?
  - Sólo necesitamos *estimar* cuanto cambiará la función objetivo al fijar una variable a cero o uno.

Pseudocost Branching

- Supondremos que conocemos estimadores p_i^+ y p_i^-, de cuanto sube o baja la función objetivo cuando incrementamos o disminuimos (respect.) el valor de una variable entera en una unidad.
- Entonces podemos definir $\hat{B}_i = \alpha \max\{D_i^+, D_i^-\} + \beta \min\{D_i^+, D_i^-\}$, donde $D_i^+ = p_i^+(1 - f_i)$ y $D_i^- = p_i^- f_i$ y escoger x_i minimizando esa cantidad.
- Supongamos que hemos brancheado en la variable x_i una cierta cantidad de veces, y resuelto los nodos hijos de esas ramificaciones.
- Entonces tenemos valores *reales* de D_i^+ y D_i^-, de donde podemos obtener un valor para p_i^+, p_i^-.

Pseudocost Branching

- Esta estrategia funciona muy bien en la práctica.
- Precisión de $\hat{B}_i$ mejora en el tiempo.
- ¿No dijimos que era más crucial al comienzo?
- Inicializamos todos los p_i^+, p_i^- a un valor fijo.
- Primera vez encontramos variable fraccionaria evaluamos cambios por **strong branching** y obtenemos valor exacto de D y p.
- Hacemos evaluación parcial de LP_i^o, LP_i^1, por ejemplo hacemos a lo más k pivotes de simplex en cada sub-problema.

Branching en la Práctica

- Sistemas mezclan **strong** y **pseudocost branching**.
- Al comienzo del árbol se evalúan parcialmente los sub-nodos por simplex de las variables suficientemente fraccionarias.
- A medida que se baja en el árbol, se pasa más a un esquema de promediar la historia.
- Usualmente $\alpha = 2, \beta = 1$, es decir, se le da más prioridad al mínimo, pero también nos importa bastante el máximo de las ramas.
- Ésta es una razón por la que conocer la solución óptima puede ser malo para los solvers (¿por que?).
- Usado para estimar efectos de branching en GUB.

Otros tópicos

- Dada la discusión anterior, ¿cómo computamos $\hat{z}_{Pi}$?
  - Asumiendo independencia de los p_i^+, p_i^-, podemos definir $\hat{z} = z_{PR} - \sum_{i \in I} \min\{D_i^+, D_i^-\}$
- Para GUB, podemos usar una división I^1, I^2 del conjunto I que maximice

$$\alpha \min\left\{\sum_{i \in I^1} D_i^-, \sum_{i \in I^2} D_i^-\right\} + \beta \max\left\{\sum_{i \in I^1} D_i^-, \sum_{i \in I^2} D_i^-\right\}$$

- La idea en GUB es producir ramas o sub-problemas que contengan una cantidad *pareja* de soluciones, y que al mismo tiempo mejore las cotas de los sub-problemas obtenidos.
- GUB es común en modelaciones de funciones lineales por tramos (SOSI).

Gomory para IP

- Consideramos $P = \max\{cx : Ax \leq b, x \in \mathbb{Z}^n_+\}$.
  - Asumiendo que $c \in \mathbb{Z}^n$, entonces $x_o = cx \in \mathbb{Z}$.
  - Reescribimos el problema como $\max\{x_o : x_o = cx, Ax \leq b, x \in \mathbb{Z} \times \mathbb{Z}^n_+\}$.
  - Consideremos una solución básica factible óptima:

$$
\begin{array}{lll}
\max & x_o & \\
 & x_{B_i} + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{a}_{io} & \forall i = 0, \ldots, m \\
 & x_{B_o} = x_o \in \mathbb{Z}, x_{B_i} \in \mathbb{Z}_+ & \forall i = 1, \ldots, m \\
 & x_j \in \mathbb{Z}_+ & \forall j \in N
\end{array}
$$

  - Nótese que por optimalidad $\bar{a}_{oj} \geq 0, \bar{a}_{io} \geq 0$.
  - Si $\bar{a}_{io} \in \mathbb{Z} \forall i$, entonces tenemos el óptimo entero!
  - Si no, consideremos lo siguiente:

Gomory para IP

- Estamos en el caso donde $\exists i$ tal que $\bar{a}_{io} \notin \mathbb{Z}$.
  - Definiendo $x_{n+t} = \lfloor \bar{a}_{io} \rfloor - x_{B_i} - \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j$ obtenemos que $\sum_{j \in N} f_{ij} x_j = f_{io} + x_{n+t}$, además $x_{n+t} \in \mathbb{Z}_+$.
  - Agregando la restricción anterior, y la nueva variable x_{n+t} a P, la solución anterior pasa a ser infactible.
  - Nótese que al agregar la nueva variable y ecuación, y consideramos x_{n+t} como básica, la nueva solución es básica y dual factible.
  - Sólo la restricción $x_{n+t} \geq 0$ es violada.
  - Podemos re-optimizar usando simplex dual.

Gomory para IP

- Esto induce el siguiente algoritmo:

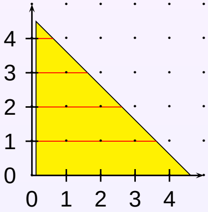
1: Iniciamos $t \leftarrow 0$, P^t el problema original.
2: Definimos S_R^t como la relajación lineal de P^t.
3: Resolvemos $PR^t := \max\{x_o : (x_o, x) \in S_R^t\}$.
4: Si PR^t infactible, **return** Infactible.
5: Si $x^t \in P^o$, **return** Solución óptima x^t.
6: Escogemos i tal que $\bar{a}_{io} \notin \mathbb{Z}$.
7: $\sum\limits_{j \in N} f_{ij} x_j - x_{n+t} = f_{io}, x_{n+t} \in \mathbb{Z}_+$ corte asociado.
8: Definimos $S_R^{t+1} = S_R^t \cap \{x_o \in \mathbb{R}, x \in \mathbb{R}_+^{n+t} : \sum\limits_{j \in N} f_{ij} x_j - x_{n+t} = f_{io}\}$.
9: $t \leftarrow t + 1$, **goto** 3.

Gomory para IP

- ¿Que sabemos de éste algoritmo?
  - Podemos suponer que $P^t \subset \{x \in \mathbb{R}^{n+1} : \sum_{i=o}^{n} x_i \leq d\}$, ¿Por qué?.
  - El método de simplex puede entregar soluciones degeneradas.
  - Eliminamos problema si retornamos solución básica óptima lexicográficamente máxima.
  - Lo podemos hacer por un simplex modificado (Simplex dual lexicográfico).
  - Escogiendo i primer índice fraccionario, y usando simplex lexicográfico podemos probar que el algoritmo de Gomory, termina finitamente.
  - ¿Y en la práctica?

Gomory para MIP

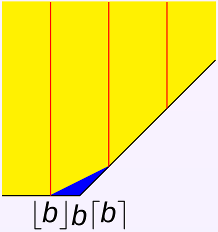
- Gomory-Chvátal cuts (1958).
  - Consideremos una fila fraccionaria del tableau óptimo.
  - Redondeo nos entrega una restricción válida.
  - Para *IP* resuelve cualquier problema.



- $x_2 \in \mathbb{Z}$, $x_1 \in \mathbb{R}^+$
- $P = \{(x_1, x_2) : x_1 + x_2 \leq 4{,}5\}$.
- $x_1 + x_2 \leq 4{,}5, x_1 \geq 0 \Rightarrow x_2 \leq 4{,}5$
- $x_2 \leq 4$.

Gomory para MIP (MIR)

- Consideremos
  $P := \{(x_1, x_2) : x_1 \in \mathbb{Z}, x_2 \geq 0, x_1 - x_2 \leq b, b \notin \mathbb{Z}\}$.



- La formulación de P es incompleta, pues incluye la zona marcada en azul.
- $(b, 0)$ es una solución básica factible de P, ¿Podemos eliminarla?.
- Necesitamos definir la recta que pasa por $(\lfloor b \rfloor, 0)(\lceil b \rceil, 1 - \hat{b})$, donde $\hat{a} := a - \lfloor a \rfloor$.

- La desigualdad que necesitamos es $x_1 - \frac{1}{1-\hat{b}} x_2 \leq \lfloor b \rfloor$.

Gomory para MIP (MIR)

- Obtuvimos la descripción ideal de P.
- ¿Cómo utilizamos esto para un MIP general?
  - Consideramos $MIP : \{x : Ax = b, x \geq 0, x_i \in \mathbb{Z} \forall i \in I\}$.
  - Re-escribimos P como $\{(x, y^+, y^-) : x - y^- + y^+ = b, y \geq 0, x \in \mathbb{Z}\}$.
  - La restricción anterior puede escribirse como $\frac{y^-}{1-\hat{b}} + \frac{y^+}{\hat{b}} \geq 1$ o como $\hat{b}x + y^+ \geq \hat{b}\lceil b \rceil$.
  - ¿Que tenemos en un tableau óptimo para un MIP con soluciones enteras fraccionarias?
  - $x_i + \sum_{j \in N_I} \bar{a}_{ij} x_j + \underbrace{\sum_{j \in N^+ \setminus N_I} \bar{a}_{ij} x_j}_{o^+ \geq 0} - \underbrace{\sum_{j \in N^- \setminus N_I} -\bar{a}_{ij} x_j}_{o^- \geq 0} = b$, donde $b \notin \mathbb{Z}$, $N_I = N \cap I$, $N^+ = \{j : j \in N, \bar{a}_{ij} > 0\}$ y $N^- = \{j : j \in N, \bar{a}_{ij} < 0\}$.

Gomory para MIP (MIR)

- Sólo nos falta decidir que hacer con $x_i + \sum_{j \in N_I} \bar{a}_{ij} x_j$
  - Sea S^1, S^2 partición de N_I, y definimos:
    - $x = x_i + \sum_{j \in S^1} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{j \in S^2} \lceil \bar{a}_{ij} \rceil x_j \in \mathbb{Z}$.
    - $y^+ = o^+ + \sum_{j \in S^1} \hat{a}_{ij} x_j \geq 0$
    - $y^- = o^- + \sum_{j \in S^2} (1 - \hat{a}_{ij}) x_j \geq 0$.
  - Entonces $x + y^+ - y^- = b$, es decir, estamos en P !!
  - El corte podemos escribirlo como
    $\frac{o^+}{\hat{b}} + \sum_{j \in S^1} \frac{\hat{a}_{ij}}{\hat{b}} x_j + \frac{o^-}{1-\hat{b}} + \sum_{j \in S^2} \frac{1-\hat{a}_{ij}}{1-\hat{b}} x_j \geq 1$
  - Escogemos $S^1 := \{j : \hat{a}_{ij} \leq \hat{b}\}$, y de esa forma maximizamos violación del corte resultante!

Cover Inequalities

- Sea $P := \{x \in \{0,1\}^n : ax \leq b\}$, $N = \{1, \ldots, n\}$.
  - Nótese que podemos asumir que $a_i > 0$, si no, definimos $x_i' = 1 - x_i$ y $b' = b - a_i$, $a_i' = -a_i$.
  - $C \subseteq N$ es un cover si $\sum(a_i : i \in C) > b$.
  - $C \subseteq N$ es minimal si $\forall i \in C$, $C \setminus \{i\}$ no es un cover.
  - Dado un cover C, se tiene $\sum(x_i : i \in C) \leq |C| - 1$.
  - ¿Podemos reforzar esta desigualdad?
  - Sea $a_C = \max\{a_i : i \in C\}$, entonces tenemos que $\sum(x_i : i \in C \cup \{i : a_i \geq a_C\}) \leq |C| - 1$.
  - Nótese que P es una sub-estructura común para MIP binarios
  - En la práctica, esta es la segunda desigualdad (versión con sequence independent lifting) más importante en términos computacionales.

Node-Packing

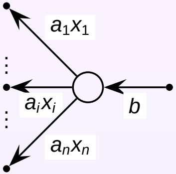
- Consideramos $G = (V, E)$ un grafo no dirigido, y definimos $P = \{x \in \{0,1\}^V : \forall (u,v) \in E, x_u + x_v \leq 1\}$.
  - Sea $C \subseteq \{V\}$ tal que $\forall u, v \in C$, $(u,v) \in E$, entonces $\sum (x_v : v \in C) \leq 1$ es una desigualdad válida.
    - La validez viene de Gomory! Veámoslo?
    - Estas desigualdades se llaman *clique inequalities*.
    - Qué dimensión tienen estas caras?
  - Si definimos $P' = \{x : x \in \{0,1\}^V : \sum (x_v : v \in C) \leq 1, \forall C \text{ clique}\}$, también es una formulación válida de P.
  - En general P' no es una formulación *ideal*, sólo si G es un grafo perfecto.
  - Es una clase de desigualdades importantes para MIP con variables binarias.

Node-Packing

- Consideramos $G = (V, E)$ un grafo no dirigido, y definimos
  $P = \{x \in \{0,1\}^V : \forall (u,v) \in E, x_u + x_v \leq 1\}$.
  - Qué otras desigualdades podemos definir?
  - Consideremos $H \subseteq V$ tal que $H = \{v_1, \dots, v_{2q+1}\}$ y tal que $(v_i, v_{i+1}) \in E$ para todo i.
  - Entonces $\sum (x_v : v \in H) \leq q$ es una desigualdad válida.
    - La validez viene de Gomory! Veámoslo?
    - Estas desigualdades se llaman *Odd hole inequalities*.
    - Qué dimensión tienen estas caras?
  - Usualmente se pide que el ciclo sea *chordless*
  - Podríamos mejorar esta desigualdad?

Fixed Charge Network Flow

- Consideramos $T = \{x \in \{0,1\}^n, y \in \mathbb{R}^n_+ : \sum(y_j : j \in N) \leq b, y_j \leq a_j x_j \, \forall j \in N\}$.



- Nuestra relajación lineal es $P = \{x, y \in \mathbb{R}^n_+ : \sum(y_j : j \in N) \leq b, y_j \leq a_j x_j \, \forall j \in N, x_j \leq 1 \, \forall j \in N\}$.
- Consideremos C un cover de $\sum(a_i x_i : i \in N) \leq b$, y definimos $\lambda = b - \sum(a_i : i \in C) \geq 0$.
- Nótese que para $k \in C$ la capacidad de $C \setminus \{k\}$ es $\min\{b, \sum a_i : i \in C \setminus \{k\}\} = b - (a_k - \lambda)^+$.
- En general para $C' \subset C$ la capacidad de $C \setminus C'$ es $b - (\sum(a_i : i \in C') - \lambda)^+ \leq b - \sum((a_i - \lambda)^+ : i \in C')$.
- $\sum(y_i : i \in C) \leq b - \sum((a_i - \lambda)^+(1 - x_i) : i \in C)$ es válida para T, estas son llamadas *flow cover*.

Fixed Charge Network Flow

- Consideramos $T = \{x \in \{0,1\}^n, y \in \mathbb{R}^n_+ : \sum(y_j : j \in N) \leq b, y_j \leq a_j x_j \; \forall j \in N\}$.
  - ¿Qué sabemos de los flow cover?
  - Son la tercera desigualdad en importancia práctica según CPLEX.
  - ¿Cuáles son los vértices fraccionarios de P?
    - Existe un cover minimal C, $k \in C$ tal que:
    - $y_i = a_i, x_i = 1$ para $i \in C \setminus \{k\}$,
    - $y_k = b - a(C \setminus \{k\}), x_k = \frac{1}{a_k} y_k$,
    - $y_i = 0, x_i = \{0,1\}$ para $i \notin C$.
  - Flow covers elimina todo vértice fraccionario de P.
  - No entregan una descripción completa de T en términos de desigualdades válidas.
  - Podemos generalizar el análisis para considerar arcos *entrantes*.