PROGRAMACIÓN ENTERA

Iker Caballero Bragagnini

Tabla de contenido

La programación entera y sus formulaciones

- Un tipo especial de problemas de programación lineal son los problemas de programación entera, en los cuales hay variables que solo pueden tomar valores en el conjunto de los números enteros

    - Los problemas de programación entera son una extensión de los modelos lineales en donde algunas o todas las variables toman valores enteros

        - Es importante destacar que para que un problema sea entero, las variables deben ser enteras, pero los coeficientes de la función objetivo no tienen por qué serlo, sino que pueden tomar valores reales

        - No obstante, las restricciones de los problemas tienen que tener coeficientes y términos independientes enteros, dado que como los valores de las variables son enteros no se podrían cumplir las restricciones con igualdad en ningún momento

        - Los problemas enteros y mixtos son más difíciles de resolver que los problemas lineales porque no el conjunto de soluciones factibles no es un conjunto convexo y el óptimo de un problema entero no siempre se encuentra en un vértice de la región factible

    - Suponiendo que se tiene un programa lineal $\max\{\boldsymbol{c}'\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$, se pueden hacer modificaciones para presentar diferentes tipos de problemas con números enteros:

        - Si algunas de las variables de decisión toman valores enteros, se tiene un programa lineal entero mixto (MIP), el cual se escribe de la siguiente forma:

$$\max_{\boldsymbol{x},\boldsymbol{y}} \{\boldsymbol{c}'\boldsymbol{x} + \boldsymbol{d}'\boldsymbol{y} : A\boldsymbol{x} + G\boldsymbol{y} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{y} \in \mathbb{Z}^n \geq \boldsymbol{0}\}$$

        - Si todas las variables de decisión toman valores enteros, entonces se tiene un programa lineal entero (IP)

$$\max_{\boldsymbol{x}} \{\boldsymbol{c}'\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^n; \boldsymbol{x} \geq \boldsymbol{0}\}$$

        - Si todas las variables de decisión toman valores 0 o 1, entonces se tiene un programa entero binario (BIP)

$$\max_{\boldsymbol{x}} \{\boldsymbol{c}'\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \{0,1\}^n\}$$

- Otro tipo de problemas a considerar son los problemas de optimización combinatoria (COP). En este tipo de problema se supone un conjunto finito $N = \{1, \dots, n\}$, el peso c_j para cada $j \in N$ y un conjunto de subconjuntos $\mathcal{F}$ factibles de N

$$\min_{S \subseteq \mathcal{N}} \left\{ \sum_{j \in S} c_j : S \in \mathcal{F} \right\}$$

  - Este problema se puede reescribir como un problema BIP, en donde se consideran variables $x_j \in \{0,1\}$ para cada objeto j

- Muchos de los problemas más usuales son de combinatoria, en el sentido de que la solución óptima es un subconjunto de puntos de un conjunto finito. Esto hace que, en principio, estos problemas se puedan resolver por enumeración

  - No obstante, este método no se puede realizar de manera factible cuando el tamaño del problema es grande. En la siguiente tabla se puede observa como las posibles combinaciones crecen cuanto mayor es el tamaño del problema:

n	$\log n$	$n^{0.5}$	n^2	2^n	$n!$
10	3.32	3.16	10^2	1.02×10^3	3.6×10^6
100	6.64	10.00	10^4	1.27×10^{30}	9.33×10^{157}
1000	9.97	31.62	10^6	1.07×10^{301}	4.02×10^{2567}

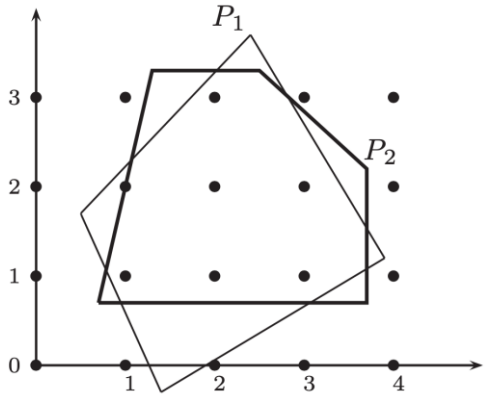
  - Por lo tanto, la enumeración completa solo se puede hacer para valores muy pequeños de n y es necesario encontrar algoritmos más inteligentes

- Un subconjunto de $\mathbb{R}^n$ descrito por un conjunto finito de restricciones lineales $P = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leq \boldsymbol{b}\}$ se denomina poliedro

  - Un poliedro $P \subseteq \mathbb{R}^{n+p}$ es una formulación para un conjunto $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ si, y solo si, $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ (si los puntos $\boldsymbol{x} \in P$ …)

  - El poliedro P, por tanto, se puede entender como el conjunto de restricciones lineales de lo que se denomina relajación lineal del problema (el programa sin considerar valores enteros)

  - Se pueden crear diversas formulaciones a partir de modificar o añadir/eliminar restricciones. Las diferentes formulaciones tienen que contener todos los puntos enteros correspondientes porque $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ y eso hace que las formulaciones sean equivalentes entre ellas

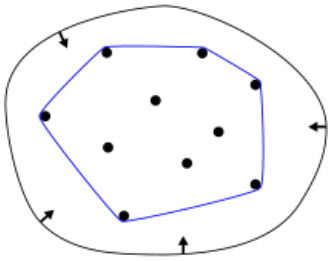


- Geométricamente se puede ver que hay infinitas formulaciones para un mismo problema, por lo que para escoger la mejor formulación se necesitan criterios adicionales

- Para poder saber qué formulación es la mejor, se tienen que utilizar las nociones de casco convexo y de vértices

  - Dado un conjunto $X \subseteq \mathbb{R}^n$, el casco convexo o *convex hull* de X (también llamado envolvente convexa), denotado como $conv(X)$, es el conjunto convexo más pequeño que contiene todos sus puntos, y que se define de la siguiente manera:

$$conv(X) = \left\{ \boldsymbol{x} : \boldsymbol{x} = \sum_{i=1}^{t} \lambda_i \boldsymbol{x^i}, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0 \ \ for\ i = 1,2,\dots,t;\ \boldsymbol{x} \in X \right\}$$



    - En otras palabras, el casco convexo de $X \subseteq \mathbb{R}^n$ es la intersección de todos los conjuntos convexos conteniendo X. Además, si $X = \{\boldsymbol{x} \in \mathbb{Z}^n : A\boldsymbol{x} \leq \boldsymbol{b}\}$, entonces $conv(X)$ es un poliedro

  - Dado un poliedro P, $\boldsymbol{x}$ es un punto extremo de P si $\boldsymbol{x^1}, \boldsymbol{x^2} \in P$ con $\boldsymbol{x} = \lambda \boldsymbol{x^1} + (1-\lambda)\boldsymbol{x^2}$ para $0 < \lambda < 1$ implica que $\boldsymbol{x} = \boldsymbol{x^1} = \boldsymbol{x^2}$. Un punto extremo corresponde a la idea geométrica de vértices

    - Cuando un programa lineal $\max \{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in PX\}$ o de maximización (donde P es un poliedro) tiene un valor óptimo finito, entonces existe un punto extremo de P que es óptimo

- Debido a estos dos resultados, se puede reemplazar un problema de programación entera $\max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X\}$ por un programa lineal equivalente $\max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in conv(X)\}$ (en el caso de minimización ocurre el análogo) y así obtener puntos extremos óptimos enteros

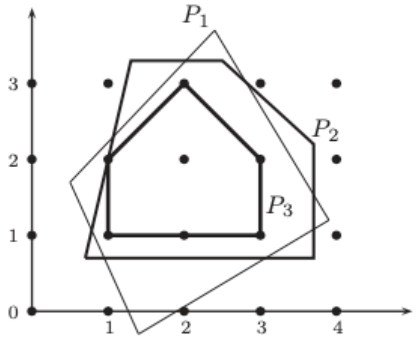
    - Esto ocurre porque, debido a que $conv(X)$ es un poliedro si $\boldsymbol{x} \in \mathbb{Z}^n$ y hay al menos un punto óptimo en los extremos (los cuales son enteros), el óptimo (entero) de este problema se encontrará también en $conv(X)$ porque $X \subseteq conv(X)$ (ya que $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ y $X \subseteq conv(X) \subseteq \mathbb{R}^p$)

    - Esta reducción ideal del programa lineal se puede hacer para conjuntos de números enteros no acotados $X = \{\boldsymbol{x} \in \mathbb{Z}^n : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ y para conjuntos de enteros mixtos $X = \{\boldsymbol{x} \in \mathbb{Z}^n, \boldsymbol{y} \in \mathbb{R}^n : A\boldsymbol{x} + G\boldsymbol{y} \leq \boldsymbol{b}; \boldsymbol{x}, \boldsymbol{y} \geq \boldsymbol{0}\}$ (en donde A, G y $\boldsymbol{b}$ son racionales)

    - Independientemente de que X esté acotado o no, este resultado es solo teórico, dado que muchas veces hay un número exponencial de desigualdades necesarias para describir $conv(X)$ y no hay una caracterización simple de estas

- Debido a que la solución ideal $conv(X)$ tiene la propiedad de que $X \subseteq conv(X) \subseteq P$ (ya que $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ y $X \subseteq conv(X) \subseteq P \subseteq \mathbb{R}^p$) para todas las formulaciones P, es posible establecer un criterio para encontrar la mejor formulación

    - Dado un conjunto $X \in \mathbb{Z}^n$ y dos formulaciones P_1 y P_2 para X, P_1 es una mejor formulación que P_2 si $P_1 \subset P_2$. Esto ocurre porque, si $X \subseteq conv(X) \subseteq P_2$, entonces $X \subseteq conv(X) \subseteq P_1 \subset P_2$ y eso hace que se tengan que considerar menos puntos al estar en una formulación lo más reducida posible. En la imagen, P_3 es la mejor formulación porque sería el casco convexo



    - Es decir, cuanto más cerca se esté del casco convexo, mejor es la formulación

- Para poder ver si $P_1 \subset P_2$, se tiene que demostrar que $P_1 \subseteq P_2$ y que $P_1 \neq P_2$. Para el primer caso, solo basta con demostrar que cualquier punto en P_1 también está en P_2, pero para el segundo, se tiene que demostrar que hay al menos algún punto en P_2 que no está contenido en P_1

- Hay problemas de programación entera para los cuales los puntos extremos de P son enteros, y eso se da porque $conv(X) = P$

  - Esto ocurre en problemas de flujos de redes, en donde $\boldsymbol{b}$ es un vector de enteros y A es una matriz de incidencia de nodos-arcos

  - También hay otras formulaciones del problema, en donde el poliedro cumple que $conv(X) = P$ y el número de restricciones que define P es exponencial

- Asumiendo que todas las variables son enteras, se considera una formulación $\min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ con $P \subseteq \mathbb{R}^n$ y otra formulación $\min\{\boldsymbol{c}'\boldsymbol{x} : (\boldsymbol{x}, \boldsymbol{w}) \in Q \cap (\mathbb{Z}^n \times \mathbb{R}^p)\}$ con $Q \subseteq \mathbb{Z}^n \times \mathbb{R}^p$, entonces se puede dar la siguiente definición:

  - Dado un poliedro $Q \subseteq \mathbb{Z}^n \times \mathbb{R}^p$, la proyección de Q sobre el subespacico $\mathbb{R}^n$, denotada por $proj_x Q$, se define de la siguiente manera:

$$proj_x Q \equiv \{\boldsymbol{x} \in \mathbb{R}^n : (\boldsymbol{x}, \boldsymbol{w}) \in Q \ \ for \ \ some \ \boldsymbol{w} \subseteq \mathbb{R}^p\}$$

  - Entonces, $proj_x Q \subseteq \mathbb{R}^n$ es la formulación obtenida a partir de Q en el espacio $\mathbb{R}^n$ para las variables originales $\boldsymbol{x}$, y permite comparar Q con otras formulaciones $P \subseteq \mathbb{R}^n$ (al estar en el mismo espacio y tener el mismo número de variables)

Los problemas canónicos de programación entera

- El problema de la mochila o *knapsack problem* se basa en que hay un presupuesto disponible b para la inversión en un conjunto de proyectos $\mathcal{N} = \{1,2,\dots,n\}$ para el próximo año, por lo que el objetivo es determinar qué subconjunto de un conjunto de proyectos $\mathcal{N}$ maximiza los beneficios esperados sin superar el presupuesto máximo

  - La formulación del problema es la siguiente:

$$\max \sum_{i=1}^{n} c_i x_i \quad s.t.$$

$$\sum_{i=1}^{n} a_i x_i \leq b, \qquad x_i \in \{0,1\}$$

- En este caso, c_i es el beneficio del proyecto $i \in \mathcal{N}$ y a_i es la inversión inicial en el proyecto $i \in \mathcal{N}$

- Las variables del problema se pueden definir como binarias que toman valores cero y uno

$$x_i = \begin{cases} 1 & if\ selected \\ 0 & otherwise \end{cases}$$

- En este problema, el número de subconjuntos posibles del conjunto de proyectos es 2^n, ya que es la cardinalidad del conjunto de poder

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

  - Además, también se pueden añadir restricciones a este problema con tal de extenderlo

    - Si no se puede invertir en más de k proyectos a la vez, entonces la suma de proyectos no puede superar el número k (porque las variables x_i son binarias y representan la elección de proyectos)

$$\sum_{i=1}^{n} x_i \leq k$$

    - Si al invertir en un proyecto a, también se tiene que invertir en un proyecto b, entonces el valor de x_a tiene que ser menor o igual al de x_b (ya que si $x_a = 1$, entonces $x_a = x_b$, pero si $x_a = 0$, entonces x_b puede ser 0 o 1 porque no es un bicondicional)

$$x_a \leq x_b$$

    - Si dos proyectos a y b no puede ser escogidos de manera simultánea, entonces la suma de sus variables tiene que ser menor o igual a 1 (no puede ser mayor a 1 porque significaría que hay más de uno de los dos proyectos seleccionados)

$$x_a + x_b \leq 1$$

- El problema de la asignación o *assignment problem* se basa en que hay n individuos que tienen que realizar n tareas, y cada individuo solo puede hacer una tarea y cada tarea solo puede ser realizada por un individuo. Hay

información sobre la afinidad de cada individuo a cada tarea, por lo que el objetico es asignar las tareas de modo que se maximice la afinidad total de los individuos

- La formulación del problema es la siguiente:

$$\max \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad s.t.$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad for\ j = 1,2,\dots,n \qquad \sum_{j=1}^{n} x_{ij} = 1 \quad for\ i = 1,2,\dots,n$$

$$x_i \in \{0,1\}$$

  - En este caso, n es el número de individuos y de tareas, mientras que c_{ij} representa la afinidad o interés del individuo i por la tarea j

  - Las variables del problema se pueden definir como binarias que toman valores cero y uno

$$x_{ij} = \begin{cases} 1 & if\ i\ is assigned to\ j \\ 0 & otherwise \end{cases}$$

  - Si cada individuo solo puede ser asignado a una tarea, entonces para una tarea j concreta, la suma de x_{ij} entre los individuos debe ser 1. Con la misma lógica, si cada tarea solo puede ser realizada por un individuo, entonces para un individuo i concreto, la suma de x_{ij} entre las tareas debe ser 1

  - Debido a que este es un problema de permutaciones, se puede ve que el posible número de asignaciones diferentes es $n!$. Además, como el problema tiene un grado asociado, por lo que se puede resolver de manera más fácil que otros

- La relajación lineal de este problema da una solución óptima entera, por lo que $conv(X)$ para$X \equiv P \cap \mathbb{Z}^{n^2}$ es P, la cual se define de la siguiente manera:

$$P \equiv \left\{ x_{ij} \in \{0,1\} : \sum_{i=1}^{n} x_{ij} = 1 \quad for\ j = 1,2,\dots,n\ ; \sum_{j=1}^{n} x_{ij} = 1 \quad for\ i = 1,2,\dots,n \right\}$$

- El problema de cobertura de conjuntos o *set covering problem* se basa en que hay n localizaciones en donde se puede instalar un servicio (con coste conocido) y m ciudades que reciben el servicio solo por algunas localizaciones

(también conocido). El objetivo es determinar en qué localizaciones se deben instalar los servicios para servir a todas las ciudades

- La formulación del problema es la siguiente:

$$\max \sum_{i=1}^{n} c_i x_i \quad s.t.$$

$$\sum_{i=1}^{n} a_{ij} x_i \geq 1 \quad for \;\; i = 1, \dots, m \qquad x_i \in \{0,1\}$$

  - En este caso, c_i es el coste de instalar el servicio en la localización i, mientras que a_{ij} es una variable binaria para $i = 1,2, \dots, n$ (perteneciente a una matriz de incidencia A) y $j = 1,2, \dots, m$ que se define de la siguiente manera:

$$a_{ij} = \begin{cases} 1 & if \;\; j \;\; is \; served \; by \; location \; i \\ 0 & otherwise \end{cases}$$

  - Las variables del problema se pueden definir como binarias que toman valores cero y uno

$$x_i = \begin{cases} 1 & if \;\; service \; installed \; in \; i \\ 0 & otherwise \end{cases}$$

- El problema es de cobertura de conjuntos porque si $\mathcal{M} \equiv \{1,2, \dots, m\}$ (ciudades) y $\mathcal{S}_i \equiv \{j : a_{ij} = 1\}$ (ciudades servidas por la localización i), el propósito es encontrar un subconjunto $T \subseteq N \equiv \{1,2, \dots, n\}$ de localizaciones tal que la unión de $\mathcal{S}_i$ para cada i sea igual a $\mathcal{M}$

$$\min \left\{ \sum_{i \in T} c_j : \bigcup_{i \in T} \mathcal{S}_i = \mathcal{M} \right\}$$

  - De este modo, el conjunto de las ciudades cubiertas por las diferentes localizaciones es igual al conjunto de ciudades (todas están cubiertas), y el objetivo es minimizar los costes totales $\sum_{i \in T} c_j$ escogiendo que localizaciones $i \in T$ para cubrir todo $\mathcal{M}$

  - Este problema de optimización combinatoria se puede expresar en términos de variables binarias, como se ha hecho anteriormente

  - En este problema, el número de subconjuntos posibles del conjunto de proyectos es 2^n, ya que es la cardinalidad del conjunto de poder

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

- Si una ciudad solo puede ser servida por una localización, entonces la restricción tiene que ser de igualdad y se suele renombrar el problema como problema de partición del conjunto

$$\sum_{i=1}^{n} a_{ij} x_i = 1 \quad for \;\; i = 1, \dots, m$$

- El problema del viajante o *traveling salesman problem* se basa en que, dado un conjunto $\mathcal{N} \equiv \{1,2, \dots, n\}$ de ciudades y distancias c_{ij} para $i, j \in \mathcal{N}$ entre ellas, el objetivo es encontrar la ruta más corta de visitar todas las ciudades sin repetir ninguna y de modo que se vuelva al punto de inicio

  - La formulación del problema es la siguiente:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \quad s.t.$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad for \;\; i = 1, \dots, n \qquad \sum_{i=1}^{n} x_{ij} = 1 \quad for \;\; j = 1, \dots, n$$

$$x_{ij} \in \{0,1\}$$

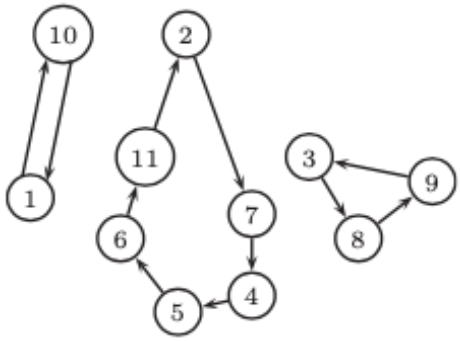
    - Las variables del problema se pueden definir como binarias que toman valores cero y uno

$$x_i = \begin{cases} 1 & if \; city \; j \; is \; visited \; after \; i \\ 0 & otherwise \end{cases}$$

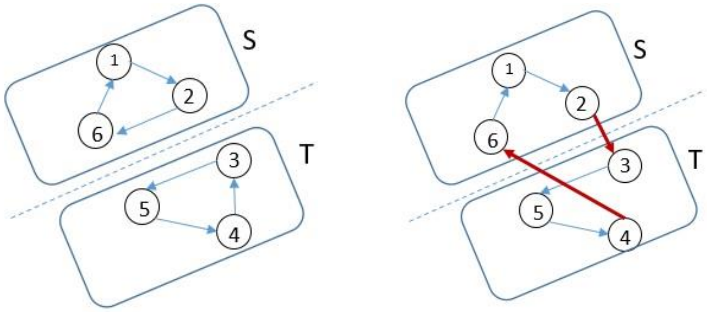
    - La primera restricción expresa que solo una ciudad se visita después de la ciudad i, mientras que la segunda expresa que se llega a la ciudad j después de salir de una sola ciudad

    - Debido a que este es un problema de permutaciones, se puede ve que el posible número de asignaciones diferentes es $n!$

  - Esta formulación es muy similar a la del problema de asignación, pero las restricciones anteriores no son suficientes porque pueden crear subciclos, de modo que se visitarían solo un subconjunto de ciudades sin quebrar ninguna restricción impuesta anteriormente. Por lo tanto, se pueden añadir dos tipos de restricciones

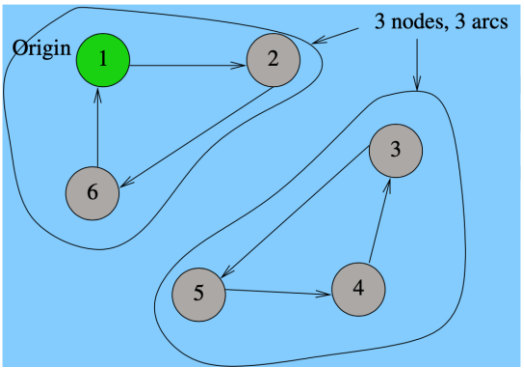


- El primer tipo de restricciones son las restricciones CUT: dada una partición de dos subconjuntos de ciudades $\mathcal{S} \neq \emptyset$ y $\mathcal{N} - \mathcal{S}$, la ruta tiene que pasar de una ciudad de $\mathcal{S}$ y $\mathcal{N} - \mathcal{S}$. Esta restricción se basa en el hecho de que, si hay subciclos, entonces se puede hacer una partición del subconjunto de ciudades de ambos subciclos en las que no se corte el arco de ninguno de los dos grados



$$\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{N}-\mathcal{S}} x_{ij} \geq 1 \quad for \ \forall \mathcal{S} \subset \mathcal{N}, \mathcal{S} \neq 0$$

- El segundo tipo de restricciones son las restricciones SUBTOUR: para cada subconjunto con más de un nodo, pero menos de n, se evita que el número de nodos sea igual al número de arcos. Esta restricción se basa en el hecho de que, si hay subciclos, entonces cada subciclo tiene el mismo número de nodos que de arcos (se le resta 1 a la cardinalidad de $\mathcal{S}$ porque no se tiene en cuenta el último arco que va de la última ciudad a la originaria)



$$\sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{N}-\mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1 \quad for \ \forall \mathcal{S} \subset \mathcal{N}, |\mathcal{S}| \geq 2$$

- El problema de estas dos restricciones es que no suelen ser factibles para problemas grandes debido a que el número de subconjuntos posibles de $\mathcal{N}$ son $2^n - 2$ para el primer tipo, y el número del segundo tipo es $2^n - 2 - n$

$$\sum_{i=1}^{n-1}\binom{n}{i} = 2^n - 2 \qquad \sum_{i=2}^{n-1}\binom{n}{i} = 2^n - 2 - n$$

- El tercer tipo de restricciones son las restricciones TIME: siendo t_i para $i = 1,2,\dots,n$ el orden de la ciudad i dentro del camino seguido por el viajero, entonces se tiene que respetar la siguiente restricción

$$t_j \geq t_i + 1 - (n+1)\left(1 - x_{ij}\right) \quad for\ i = 1,\dots,n \ \&\ j = 2,\dots,n$$

  - Se tiene que garantizar que si $x_{ij} = 1$, entonces $t_j = t_i + 1$, lo cual expresa que primero va la ciudad i (que es de donde se parte), y después la ciudad j (a donde se dirige el viajero). Esta restricción permite observar como se da la condición

$$x_{ij} = 1 \ \Rightarrow \ t_j \geq t_i + 1$$

  - Se tiene que garantizar que si $x_{ij} = 0$, entonces t_j no está restringido lo cual expresa que si primero no se ha visitado la ciudad i (que es de donde se parte), entonces el orden de j (a donde se dirige el viajero) puede ser cualquiera. Esta restricción permite observar como se da la condición

$$x_{ij} = 1 \ \Rightarrow \ t_i - n \leq \ t_j \leq 0$$

  - Además, se fija $t_1 = 1$ (ya que es la ciudad en la que empieza el viajero) y se tiene que garantizar que t_i tome valores entre 1 y n (porque es el número de ciudades totales). Por lo tanto, $j \neq 1$ (porque si no $t_1 \geq t_i + 1$) y $j \neq i$ (porque si no $t_i \geq t_i + 1$)

- Por lo tanto, una formulación final viable (aunque no muy eficiente es la siguiente:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad s.t.$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad for\ i = 1,\dots,n \qquad \sum_{i=1}^{n} x_{ij} = 1 \quad for\ j = 1,\dots,n$$

$$t_j \geq t_i + 1 - (n+1)\left(1 - x_{ij}\right) \quad for\ i = 1,\dots,n\ ;\ j = 2,\dots,n\ \ \&\ \ i \neq j$$

$$t_1 = 1 \qquad x_{ii} = 0 \quad for\ i = 1,2,\dots,n$$

$$x_{ij} \in \{0,1\} \quad for\ i = 1,\dots,n\ \ \&\ \ j = 1,\dots,n$$

$$1 \leq t_i \leq n \quad for\ i = 1,2,\dots,n$$

  - La formulación final hace que haya n^2 variables binarias, n variables continuas y $2n + n(n-1) - (n-1)$ restricciones

- El problema del transporte o *transportation problem* se basa en que una compañía produce bienes en n centros de producción y hay m centros de demanda para estos bienes. El objetivo es transportar bienes de los centros de producción a los diferentes centros de demanda a un coste mínimo

  - La formulación del problema es la siguiente:

$$\max \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \quad s.t.$$

$$\sum_{j=1}^{m} x_{ij} \leq S_i \quad for\ i = 1,2,\dots,n \qquad \sum_{i=1}^{n} x_{ij} \geq D_j \quad for\ j = 1,2,\dots,m$$

$$\sum_{j=1}^{m} D_j \leq \sum_{i=1}^{n} S_i$$

$$x_{ij} \in \mathbb{Z}$$

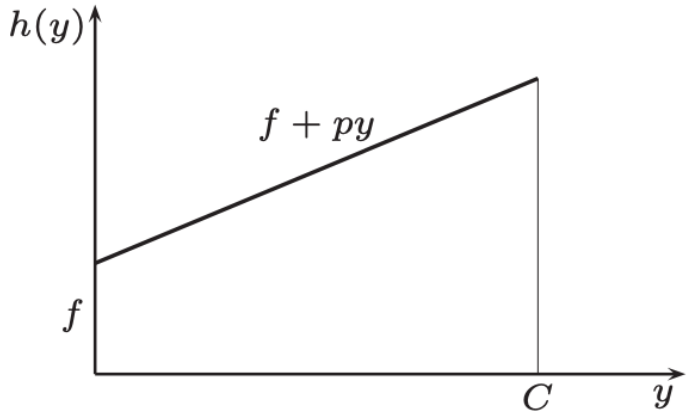
  - En este caso, c_{ij} es el coste de transportar cada unidad desde el centro de producción i al centro de demanda j, S_i es la producción hecha en el centro i y D_j es la demanda en el centro de demanda j

- Las variables del problema se pueden definir como variables que toman valores enteros y representan el número de unidades transportadas desde i a j

- El problema de cargos fijos o *fixed charge problem* se basa en que algunas actividades $i = 1,2,\dots,n$ tienen un coste fijo k_i y otro coste variable $c_i x_i$ cuando se realizan

  - En este caso, la función de costes de las actividades es la siguiente:

$$f(x_i) = \begin{cases} k_i y_i + c_i x_i & if \;\; x_i > 0 \\ 0 & if \;\; x_i = 0 \end{cases}$$



    - Si $k_i = 0$ para cualquier $i = 1,2,\dots,n$, se tiene un problema de programación lineal

  - Si se considera que y_i para $i = 1,2,\dots,n$ son variables binarias que indican que se realiza una actividad i, entonces el problema se puede formular de la siguiente manera:

$$\min \sum_{i=1}^{n} k_i y_i + c_i x_i \quad s.t.+$$

$$x_i \leq M y_i \quad for \;\; i = 1,2,\dots,n$$

    - Las variables x_i del problema representan el número de veces que se realiza la actividad, y x_i puede ser real o puede ser entero

    - Se necesita garantizar que $x_i = 0$ cuando $y_i = 0$ (si no se realiza la actividad, entonces el número de veces debe ser cero), de modo que se escoge una M lo más pequeña posible

Las aplicaciones de la programación entera

- El problema de la mediana óptima de clústeres u *optimal cluster-median problem* (también conocido como k-*medoid problem*) es un problema muy utilizado en *marketing* computacional y con aplicaciones en ciencia de datos para datos en donde haya clústeres

    - Este basa en que hay un matriz de datos $A = [a_{ij}]$ para $i = 1,2,\dots,m$ y $j = 1,2,\dots,n$ de m puntos y n variables y se quiere agruparlos en k clústeres y hacer que la distancia total entre puntos de cada clúster y su mediana sea mínima

        - Debido a que las m filas de la matriz A son los puntos y n son las variables, se puede obtener una matriz cuadrada de distancias $m \times m$ entre un punto y otro

        $$D = [d_{ij}] \quad for \quad i = 1,2,\dots,m \ \& \ j = 1,2,\dots,m$$

        - La matriz cuadrada de distancias suele ser simétrica (debido a que la distancia entre i y j es la misma que de j a i), pero también puede no serlo sin que afecte al problema

        - Con tal de que todos los datos tengan la misma importancia, se suelen normalizar o estandarizar los elementos de la matriz A para que las distancias den un peso justo

    - La mediana (o *medoid*) para un subconjunto de puntos $\mathcal{I} \subseteq \{1,2,\dots,m\}$ es el punto r más cercano a todos los puntos de $\mathcal{I}$

    $$r \ is \ the \ median \ or \ medoid \ of \ \mathcal{I} \ if \ \sum_{i \in \mathcal{I}} d_{ir} = \min_{j \in \mathcal{I}} \sum_{i \in \mathcal{I}} d_{ij}$$

        - El medoide se define como un objeto multidimensional de un grupo cuya disimilaridad media a todos los objetos en el grupo es mínima. No obstante, se utiliza el término mediana para referirse a este objeto multidimensional (aunque sea un concepto bidimensional) y así tener un mejor entendimiento del concepto

    - La formulación de este problema es la siguiente:

    $$\min \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} x_{ij} \quad s.t.$$

    $$\sum_{j=1}^{m} x_{ij} = 1 \quad for \ i = 1,\dots,m \qquad \sum_{j=1}^{m} x_{jj} = k$$

$$x_{jj} \geq x_{ij} \quad for \ i = 1, \dots, m \ \& \ j = 1, \dots, m$$

$$x_{ij} \in \{0,1\} \quad for \ i = 1, \dots, m \ \& \ j = 1, \dots, m$$

- Las variables del problema se pueden definir como binarias que toman valores cero y uno a partir del clúster-j, que es el clúster que tiene como mediana el elemento j de $j = 1,2, \dots, m$

$$x_{ij} = \begin{cases} 1 & if \ element \ i \ belongs \ to \ cluster - j \\ 0 & otherwise \end{cases}$$

- La primera restricción expresa que cada punto i pertenece al menos a un clúster-j; la segunda expresa que existen k clústeres (a través de decir que existen solo k medianas); y la tercera expresa que un punto pertenece a un clúster-j si el clúster-j existe (a través de que si $x_{jj} = 1$, entonces existe un clúster-j con mediana j y cualquier punto x_{ij} que pertenezca al mismo clúster tiene que ser $x_{jj} = x_{ij}$)

- Aunque en las sumatorias para j se consideren m clústeres, la segunda restricción hace que solo haya k clústeres efectivos, dejando así $m - k$ clústeres vacíos

- El grupo de m^2 restricciones (la tercera restricción) se podría formular como $mx_{jj} \geq \sum_{i=1}^{m} x_{ij}$ para cada $i = 1,2, \dots, m$, de modo que habría m restricciones. Sin embargo, eso no quiere decir que la formulación con esta restricción sea mejor

  - Este problema es muy difícil y la solución solo es viable cuando m no es excesivamente grande

    - Existen algunas heurísticas para el problema. El más famoso es el algoritmo *Partitioning Around Medoids* (PAM), que tiene similaridades con la heurística para el problema de k medias

    - Algunas soluciones aproximadas se pueden obtener construyendo un árbol de expansión para los m puntos y después eliminando los $k - 1$ arcos más largos del árbol

- El problema de k medias o *k-means problem* es un problema de clusterización cuyo objetivo consiste en, dados $x_i \in \mathbb{R}^n$ en $i = 1,2, \dots, m$, encontrar una partición $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2 \dots, \mathcal{S}_k\}$ de los m puntos en k clústeres $\mathcal{S}_1, \mathcal{S}_2 \dots, \mathcal{S}_k$ de modo que la suma de cuadrados SSW dentro de los clústeres es minimizada

  - La formulación para este problema es la siguiente:

$$\min \sum_{i=1}^{k} SSW_i = \sum_{i=1}^{k} \sum_{j \in \mathcal{S}_i} \left\| x_j - \bar{x}^i \right\|^2 \quad s.t.$$

$$\bar{x}^i = \frac{1}{|\mathcal{S}_i|} \sum_{k \in \mathcal{S}_i} x_k$$

- En este caso se utiliza el cuadrado de la norma debido a que se está en un espacio vectorial $\mathbb{R}^n$

- El problema de esta formulación es que $\bar{x}^i$ depende de los puntos $x_j \in \mathbb{R}^n$ del clúster $\mathcal{S}_i$, mientras que la función objetivo sirve para asignar los puntos a estos clústeres, con lo cual no se puede calcular porque hay una recursividad. El SSW_i puede escribirse de forma alternativa de la siguiente manera:

$$SSW_i = \sum_{j \in \mathcal{S}_i} \left\| x_j - \bar{x}^i \right\|^2 = \frac{1}{2|\mathcal{S}_i|} \sum_{j,k \in \mathcal{S}_i} \left\| x_j - x_k \right\|^2$$

- Como se puede ver, este es un método de aprendizaje no supervisado, dado que no hay variables dependientes y_i para los puntos dados $x_j \in \mathbb{R}^n$

- Una heurística para el problema de k-medias consiste en seguir los siguientes pasos:

  - Dado un conjunto de datos $x_j \in \mathbb{R}^n$ para $j = 1,2,\dots,m$ y dado un número de centros (clústeres) k, se escogen k centros iniciales $\bar{x}^i = c_i$ para $i = 1,2,\dots,k$ (para cada clúster)

  - Se asigna cada punto al clúster que tenga la media más cercana a este para así obtener una partición $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_k\}$ y se actualizan las medias escogidas con la media de los puntos del clúster k

$$\bar{x}^i = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} x_j$$

  - Este último se repite muchas veces hasta que la partición $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_k\}$ no cambia. Cuando la partición no cambia, quiere decir que cada punto está lo más cerca posible de la media de su respectivo clúster y el algoritmo finaliza

  - Este algoritmo es muy rápido, pero no da un óptimo global, sino uno local. Además, es muy sensible a los centros escogidos en el paso inicial

- El problema de los cuadrados latinos ortogonales o *ortogonal latin squares* es un problema que nace en el diseño de experimentos óptimos bajo condiciones específicas (qué tratamientos se utilizarán dadas unas series de factores)

    - Un cuadrado latino es una matriz $L = [l_{ij}]$ de tamaño $r \times r$ en la que cada casilla está ocupada por uno de los r símbolos de tal modo que cada uno de ellos aparece exactamente una vez en cada columna y en cada fila

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \qquad \begin{bmatrix} a & b & d & c \\ b & c & a & d \\ c & d & b & a \\ d & a & c & b \end{bmatrix}$$

        - Debido a que la matriz es bidimensional, cada una representa a un factor (por lo que se representan dos factores) con r elementos dentro de cada factor

        - Este tipo de matriz se utiliza el diseño de experimentos en donde se quiera eliminar la variación debido a un factor externo diferente a los dos considerados en la matriz

    - Si se quiere considerar más de dos factores (los representados por la matriz), se tiene que utilizar un cuadrado latino que sea ortogonal a todos los otros para cada elemento r de los otros factores

        - Se dice que dos cuadrados latinos L_1 y L_2 son mutuamente ortogonales si los pares ordenados (l_{ij}^1, l_{ij}^2) son diferentes para toda i y j

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{matrix} \quad L_1 \qquad \begin{matrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \end{matrix} \quad L_2 \qquad \begin{matrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \\ 2 & 1 & 4 & 3 \end{matrix} \quad L_3$$

        - Por lo tanto, no quiere decir que en cada misma casilla debe haber un número diferente, sino que no hay dos pares de números (de la misma casilla para cada matriz) que sean iguales

        - Todos los pares ordenados (k, l) para $k, l \in \{1,2, \dots, r\}$ aparecen en (l_{ij}^1, l_{ij}^2)

    - El problema de los cuadrados latinos ortogonales tiene como objetivo encontrar un par de cuadrados latinos los cuales sean mutuamente

ortogonales. La formulación del problema de optimización sin la función objetivo sería la siguiente (para encontrar solo una solución factible, no óptima):

$$\sum_i \sum_j x_{ijkl} = 1 \ \ for \ \ k,l = 1,2,\dots,r$$

$$\sum_k \sum_l x_{ijkl} = 1 \ \ for \ \ i,j = 1,2,\dots,r$$

$$\sum_j \sum_l x_{ijkl} = 1 \ \ for \ \ i,k = 1,2,\dots,r$$

$$\sum_j \sum_k x_{ijkl} = 1 \ \ for \ \ i,l = 1,2,\dots,r$$

$$\sum_i \sum_l x_{ijkl} = 1 \ \ for \ \ j,k = 1,2,\dots,r$$

$$\sum_i \sum_k x_{ijkl} = 1 \ \ for \ \ j,l = 1,2,\dots,r$$

$$x_{ijkl} \in \{0,1\} \ \ for \ \ i,j,k,l = 1,2,\dots,r$$

- Las variables del problema se pueden definir como binarias que toman valores cero y uno, habiendo así r^4 variables binaias

$$x_{ijkl} = \begin{cases} 1 & if \ (k,l) \ is \ in \ the \ position \ (i,l) \\ 0 & otherwise \end{cases}$$

La optimalidad, la relajación y las cotas en programación entera

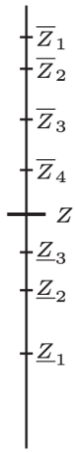
- Dado un problema IP o COP como $z = \min\{c(\boldsymbol{x}) : \boldsymbol{x} \in X \subseteq \mathbb{Z}^n\}$ es interesante saber si un punto $\boldsymbol{x}^*$ es óptimo, de modo que se buscan condiciones de optimalidad para que el algoritmo pare de buscar un óptimo en un problema IP

    - Una respuesta lógica a la pregunta es encontrar cotas inferiores $\underline{z} \leq z$ y superiores $\overline{z} \geq z$ hasta que $\underline{z} = \overline{z} = z$

        - Esto ocurre porque si $\overline{z} = z$, entonces no hay valores superiores a z y tampoco hay valores inferiores si $\underline{z} = z$, por lo que z debe ser el óptimo (ya que las cotas se definen con respecto a esta)

        - De este modo, el algoritmo tiene que encontrar una secuencia creciente $\underline{z}_1 < \underline{z}_2 < \cdots < \underline{z}_t \leq z$ y una secuencia decreciente $\overline{z}_1 > \overline{z}_2 > \cdots > \overline{z}_s \geq z$, parando una vez $\overline{z}_s - \underline{z}_t \leq \varepsilon$, en donde

ε es un número pequeño no negativo llamado parámetro de tolerancia o *optimality gap*



- En consecuencia, lo importante es encontrar o diseñar una manera de poder conseguir estas cotas

- Es posible encontrar una cota inferior o superior para problemas de programación entera en donde se maximice o se minimice la función objetivo, respectivamente

  - Cualquier solución factible $\boldsymbol{x}^* \in X$ proporciona una cota superior $\overline{z} = c(\boldsymbol{x}^*) \geq z$ para un programa $\min\{c(\boldsymbol{x}) : \boldsymbol{x} \in X \subseteq \mathbb{Z}^n\}$, llamada cota primal, dado que si $\boldsymbol{x}^*$ no es el óptimo, entonces no minimiza la función objetivo y es mayor al valor óptimo, pero si $\boldsymbol{x}^*$ es el óptimo, entonces $c(\boldsymbol{x}^*) = z$ y sigue siendo una cota superior (no hay valor menor a este)

  - Cualquier solución factible $\boldsymbol{x}^* \in X$ proporciona una cota inferior $\underline{z} = c(\boldsymbol{x}^*) \leq z$ para un programa $\max\{c(\boldsymbol{x}) : \boldsymbol{x} \in X \subseteq \mathbb{Z}^n\}$, llamada cota primal, dado que si $\boldsymbol{x}^*$ no es el óptimo, entonces no maximiza la función objetivo y es menor al valor óptimo, pero si $\boldsymbol{x}^*$ es el óptimo, entonces $c(\boldsymbol{x}^*) = z$ y sigue siendo una cota inferior (no hay valor mayor a este)

- Encontrar cotas superiores para problemas de maximización o cotas inferiores para problemas de minimización es difícil, por lo que se tiene que recurrir a la dualidad y así obtener cotas duales

  - El enfoque más importante para encontrar cotas duales es el de relajación del problema, en donde la idea es reemplazar un problem IP difícil por un problema de optimización más simple cuyo valor óptimo es mayor o igual (si se maximiza) o menor o igual (si se minimiza) a z

- Existen dos posibilidades para que la relajación del problema tenga esta propiedad: alargar el conjunto de soluciones factibles para que se puede optimizar sobre un conjunto más grande o reemplazar la función objetivo por una función que tenga un valor menor (en caso de minimización) o mayor (en caso de maximización) para todos los puntos

- Un problema $z^R = \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in T \subseteq \mathbb{R}^n\}$ es una relajación (RP) del problema IP $z = \max\{c(\boldsymbol{x}) : \boldsymbol{x} \in X \subseteq \mathbb{R}^n\}$ si $X \subseteq T$ (el conjunto factible de la relajación contiene el del problema original) y $f(\boldsymbol{x}) \geq c(\boldsymbol{x})$ para toda $\boldsymbol{x} \in X$

  - En consecuencia, si el problema RP es una relajación de un problema IP, entonces $z^R \leq z$. Si $\boldsymbol{x}^*$ es una solución óptima del problema IP, entonces $\boldsymbol{x}^* \in X \subseteq T$ y $z = c(\boldsymbol{x}^*) \leq f(\boldsymbol{x}^*)$ (por suposición), y como $\boldsymbol{x}^* \in T$, $f(\boldsymbol{x}^*)$ es una cota inferior en z^R y eso hace que $z \leq f(\boldsymbol{x}^*) \leq z^R$

  - En el caso en donde sea un problema de minimización, la condición $f(\boldsymbol{x}) \geq c(\boldsymbol{x})$ cambia por $f(\boldsymbol{x}) \leq c(\boldsymbol{x})$ para toda $\boldsymbol{x} \in X$ y $z \geq f(\boldsymbol{x}^*) \geq z^R$

  - En todo caso, se puede ver como $\boldsymbol{x}^*$ es el punto óptimo en el que se optimiza la función tanto en la relajación como en el problema IP original, lo único que cambia es el valor de la función

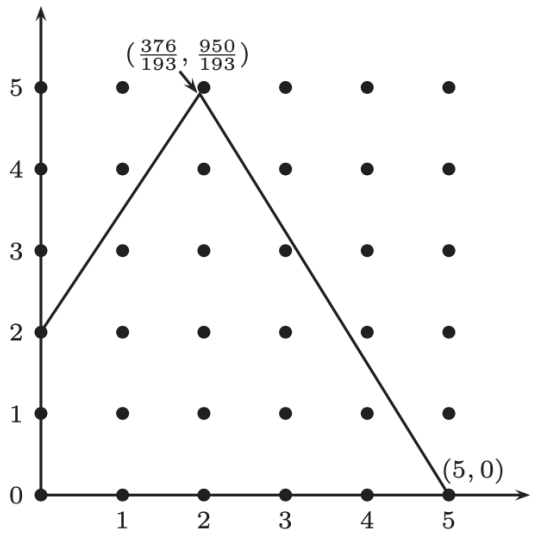
- Tal y como se ha explicado, para encontrar cotas duales en los problemas de programación entera, es necesario hacer una relajación de las restricciones. No obstante, hay diferentes tipos de relajaciones, pero la más importante es la relajación a un programa lineal

  - Para un programa entero $\max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in P \cap \mathbb{Z}^n\}$ con formulación $P = \{\boldsymbol{x} \in \mathbb{R}^n_+ : A\boldsymbol{x} \leq \boldsymbol{b}\}$, la relajación a un programa lineal es el programa lineal $\max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in P\}$

    - Como $P \cap \mathbb{Z}^n \subseteq P$ (el conjunto factible está incluido en el de la relajación) y la función objetivo no ha cambiado, el programa lineal es claramente una relajación como se ha definido anteriormente

    - Como la relajación es un programa lineal, los vértices o puntos extremos de la región factible son las soluciones de este programa. Si la solución es entera, también lo es para el problema IP y se habrá encontrado el óptimo (aunque generalmente no es el caso)

- Truncar o redondear la solución de la relajación no garantiza que ese punto sea el óptimo para el problema IP, dado que esta nueva solución redondeada o truncada puede no ser factible en IP



  - Para poder encontrar los puntos óptimos en programas lineales se suele utilizar el método Simplex visto anteriormente

    - También hay otros métodos como los métodos de puntos interiores, los cuales resuelven el problema LP de forma polinómica

La complejidad y las reducciones de problemas

- Muchos problemas vistos hasta ahora cumplen con la propiedad de optimización eficiente, de modo que hay un algoritmo polinómico para estos problemas, pero hay otros para los que no existen tales algoritmos, por lo que para poder lidiar con estos es necesario hacer distinciones y clasificar los problemas a través de conceptos sobre su complejidad

  - En ciencia computacional, la complejidad temporal es la complejidad computacional que describe la cantidad de tiempo que tarda una computadora en ejecutar un algoritmo

    - Esta cantidad normalmente se estima contando el número de operaciones elementales realizadas por el algoritmo, suponiendo que cada operación elemental tome un tiempo fijo en ser ejecutada

    - Por lo tanto, se considera que la cantidad de tiempo que toma y el número de operaciones elementales realizadas por el algoritmo están relacionadas por un factor constante

- La complejidad temporal se expresa generalmente como una función del tamaño de su insumo, pero como esta función es difícil de calcular exactamente, uno se enfoca en la complejidad cuando el insumo incrementa (el comportamiento asintótico de la complejidad)

    - Por ello, la complejidad temporal se suele expresar utilizando la notación $O\big(g(n)\big)$, donde g es una función de variable real y n es el número de bits necesarios para representar el insumo

    - Siendo f (la función a ser estimada) una función de variable real o compleja y siendo g (la función de comparación) una función de variable real estrictamente positiva para valores suficientemente grandes de n, y definiéndose ambas en un subconjunto no acotado de los reales positivos, se cumple la siguiente proposición:

$$f(n) = O\big(g(n)\big) \ \ as \ \ n \to a \quad if \quad \limsup_{n \to a} \frac{|f(n)|}{g(n)} < \infty$$

- Las complejidades algorítmicas se clasifican acorde al tipo de función g que aparece en la notación $O\big(g(n)\big)$, siendo las clasificaciones más importantes el tiempo constante, el tiempo logarítmico, el tiempo lineal y el tiempo polinómico

    - Un algoritmo es de tiempo constante $O(1)$ si el valor de $T(n)$ (la complejidad del algoritmo) está acotada por un valor que no depende del tamaño del insumo. Si $T(n)$ es $O(a)$, donde a es cualquier valor constante, es equivalente a la notación $O(1)$

    - Un algoritmo es de tiempo logarítmico cuando $T(n) = O(\log n)$. Como $\log_a n$ y $\log_b n$ se relacionan por un multiplicador constante, y este es irrelevante para la notación $O\big(g(n)\big)$ (debido a la definición que se da), se utiliza $O(\log n)$ sin importar cual es la base del logaritmo de $T(n)$

    - Un algoritmo es de tiempo lineal si la complejidad temporal $T(n)$ es $O(n)$, de modo que el tiempo de ejecución incrementa como mucho de manera lineal con el tamaño del insumo

    - Un algoritmo es de tiempo polinómico si el tiempo de ejecución está acotado superiormente por una expresión polinómica del tamaño del insumo del algoritmo, por lo que $T(n) = O(n^k)$ para alguna constante positiva k

- Para poder desarrollar un método de clasificación de problemas, se necesitan los siguientes conceptos básicos:

- Una clase $\mathcal{C}$ de problemas legítimos para la cual la teoría se aplica

- Una subclase no vacía $\mathcal{C}_{\mathcal{A}} \subseteq \mathcal{C}$ de problemas "fáciles" y una subclase no vacía $\mathcal{C}_{\mathcal{B}} \subseteq \mathcal{C}$ de problemas difíciles

- Una relación de "no más difícil que" entre pares de problemas legítimos

  - Esto permite que se pueda establecer el lema de reducción. Suponiendo que P y Q son dos problemas legítimos, se cumplen las proposiciones siguientes:

    - Si Q es "fácil" y P es "no más difícil que" Q, entonces P es fácil

    - Si P es "difícil" y P es "no más difícil que" Q, entonces Q es difícil

    - La primera parte de este lema se ha podido usar en varios problemas, en donde es posible reducirlos a otros problemas "fáciles"

- ...

  - Para poder definir una clase de problemas legítimos, es apropiado considerar problemas de decisión teniendo una respuesta binaria. Por lo tanto, un problema de optimización $\max\{cx : x \in S\}$, para el cual una instancia consiste de $\{c, standard\ representation\ of\ S\}$, se reemplaza por un problema de decisión "¿Hay alguna $x \in S$ con valor $cx \geq k$?", para el cual una instancia consiste de los elementos $\{c, standard\ representation\ of\ S, k \in \mathbb{Z}\}$

    - De este modo, cuando se hace una referencia a un problema de optimización, se tiene en mente que corresponde a un problema de decisión como el planteado

  - El número de variables y restricciones o de nodos y vértices no es lo único que define la longitud de un insumo, sino también el tamaño de los números ocurriendo en los datos

    - Para una instancia X del problema, la longitud del insumo $L = L(X)$ es la longitud de la representación binaria de una representación estándar de la instancia

    - Dado un problema P, un algoritmo A para el problema y una instancia X, se define $f_A(X)$ como el número de cálculos elementales requeridos para ejecutar el algoritmo A en la

instancia X. Además, $f_A^*(l) = \sup_{X}\{f_A(X) : L(X) = l\}$ es el tiempo de ejecución de un algoritmo A

- Un algoritmo A es polinómico para un problema P si $f_A^*(l) = O(l^p)$ para un entero positivo p

- Una vez con estos conceptos, es posible definir clases de problemas legítimos: la clase $\mathcal{NP}$ y la clase $\mathcal{P}$

  - La clase $\mathcal{NP}$ es la clase de problemas de decisión con la propiedad de que para cualquier instancia en la que la respuesta sea 1 o "sí", entonces hay una demostración "corta" o polinómica de ese 1 o "sí"

  - Por lo tanto, si un problema de decisión asociado a un problema de optimización está en $\mathcal{NP}$, entonces el problema de optimización se puede resolver respondiendo el problema de decisión un número polinómico de veces (usando bisección en el valor de la función objetivo)

  - La clase $\mathcal{P}$ es la clase de problemas de decisión en $\mathcal{NP}$ para el cual existe un algoritmo polinómico

- …

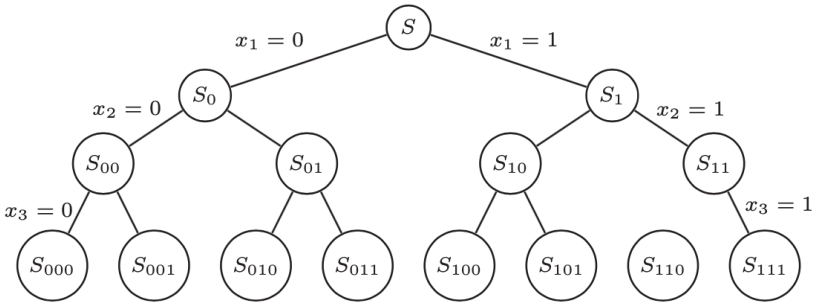
El algoritmo *branch and bound*

- Uno de los algoritmos o sistemas más útiles para poder encontrar los óptimos en problemas IP es el de ramificación y límite o *branch and bound*, el cual se basa en un enfoque de división y conquista y en la enumeración implícita

  - Considerando un problema $z = \max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in S\}$, es posible dividir el problema en partes más pequeñas y fáciles para poder juntar la información posteriormente y resolver el problema original

    - Siendo $S = S_1 \cup \ldots \cup S_K$ una descomposición de S en subconjuntos más pequeños y $z^k = \max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in S_k\}$ para $k = 1,2,\ldots,K$, entonces $z = \max_k z^k$

    - Una manera típica de representar este enfoque de dividir y conquistar es a través de un árbol de enumeración, en donde cada nodo es un subconjunto del conjunto precedente (en el nodo anterior y hay tantos niveles como variables de decisión)

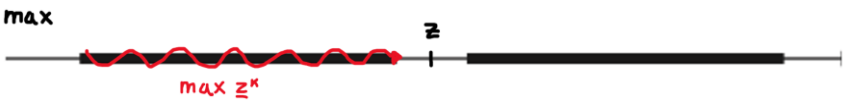


- Obviamente, la solución única y óptima del problema entero solo pertenece a un solo subconjunto de la partición (no puede haber dos soluciones óptimas y no puede haber dos subconjuntos con un mismo valor al ser partición. Además, esta separación no puede descartar soluciones candidatas al óptimo, por lo que todas las soluciones factibles $\boldsymbol{x}$ deben pertenecer a un subconjunto de restricciones S_k

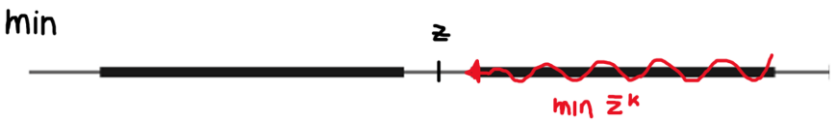
$$S_1 \cap \ldots \cap S_K = \emptyset \qquad \boldsymbol{x} \in S_k$$

- Como se ha visto anteriormente, una enumeración completa es totalmente imposible para la mayoría de problemas cuando el número de variables de decisión es grande en un programa entero. Por ello, es necesario usar cotas en los valores de $\{z^k\}_{k=1}^K$ de manera inteligente

  - Siendo $S = S_1 \cup \ldots \cup S_K$ una descomposición de S en subconjuntos más pequeño $z^k = \max\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in S_k\}$ para $k = 1,2,\ldots,K$, $\overline{z}^k$ una cota superior de z^k y $\underline{z}^k$ una cota inferior de z^k, entonces $\overline{z} = \max_k \overline{z}^k$ es una cota superior de z y $\underline{z} = \max_k \underline{z}^k$ es una cota inferior de z



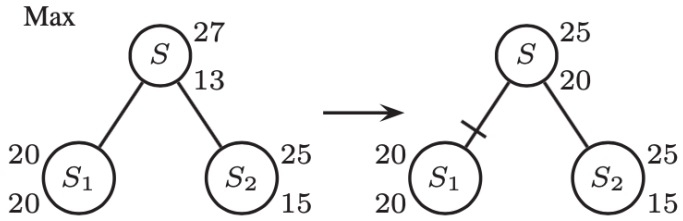
  - Esta proposición aplica de manera análoga para el caso en el que se minimiza, pero cambiando $\overline{z} = \min_k \overline{z}^k$ y $\underline{z} = \min_k \underline{z}^k$



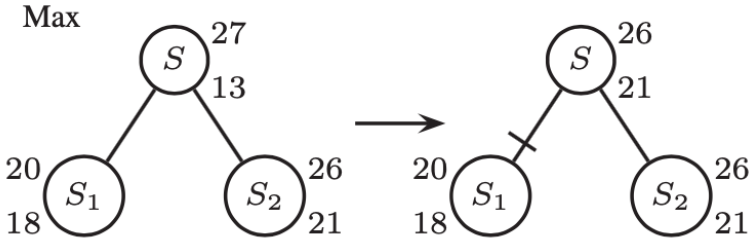
  - De este modo, si se pueden encontrar unas cotas inferiores $\underline{z}$ (para maximización) o unas cotas superiores $\overline{z}$ (para minimización) en una rama pronto, las otras ramificaciones se pueden descartar sin tener que explorarlas en su totalidad (se poda el árbol de decisión)

- Hay tres maneras de poder podar el árbol creado y así enumerar un gran número de soluciones de manera implícita a través del resultado anterior:

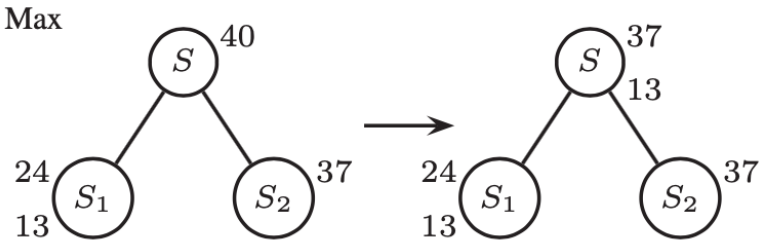
  - Se puede podar por optimalidad, de modo que se eliminan las ramas en donde las cotas inferiores $\underline{z}^k$ y superiores $\overline{z}^k$ (de ese nodo k) sean equivalentes, y, por tanto, no se puede explorar más en esas ramas



  - Se puede podar por cota, de modo que se elimina la rama para la cual la cota $\overline{z}^k < \underline{z}$ (la cota superior en k es menor a la inferior del óptimo z)



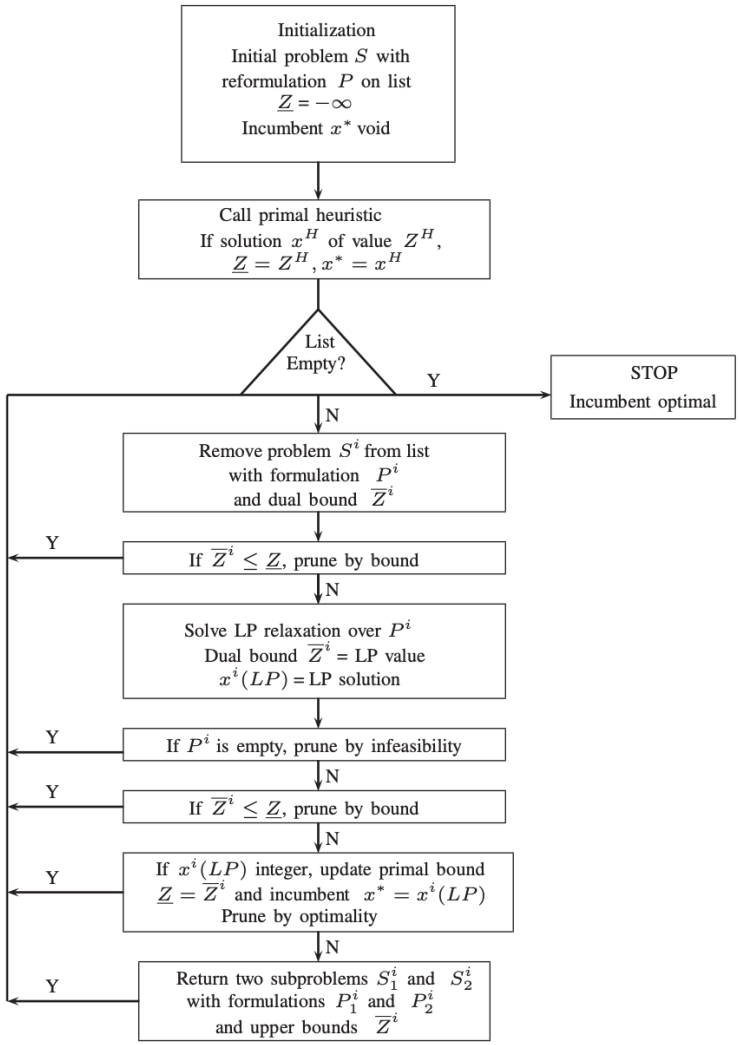
  - Si no se puede aplicar ninguno de los dos métodos anteriores, entonces no se puede podar (es infactible) y se tiene que seguir explorando ambas ramas



- Las cotas se pueden obtener a través de soluciones factibles (cotas primales) y de relajación o dualidad (cotas duales). No obstante, el diseño de un algoritmo así no es tan fácil

  - Una pregunta es qué tipo de relajación o problema dual utilizar, cómo dividir la región factible en varias subregiones, cuántas subregiones crear, el orden de examinación, etc.

- Los pasos para utilizar un algoritmo de *branch and bound* utilizando una relajación a un programa lineal son los siguientes:



  - Primero se hace una relajación a un programa lineal para el IP original (para el conjunto factible original) y se obtiene una solución óptima $\boldsymbol{x_0^*}$ para el problema

$$z = \max\{\boldsymbol{c'x} : \boldsymbol{x} \in S \cap \mathbb{Z}^n\} \Rightarrow z_0 = \max\{\boldsymbol{c'x} : \boldsymbol{x} \in S\} \Rightarrow \boldsymbol{x_0^*}$$

$$where\ S = \{\boldsymbol{x} \in \mathbb{R}_+^n : A\boldsymbol{x} \leq \boldsymbol{b}\}$$

    - En este caso, la cota superior $\overline{z} = \overline{z}_0$ será la solución encontrada (en el caso de minimización, sería una cota inferior), y si no se puede encontrar una solución factible (entera), se supone que $\underline{z} = \infty$ (en el caso de minimización sería $\overline{z} = -\infty$)

$$fix\ \ \overline{z} = \overline{z}_0\ \ \&\ \ \underline{z} = \infty\ \ if\ no\ feas.sol.found\ \ (max.)$$

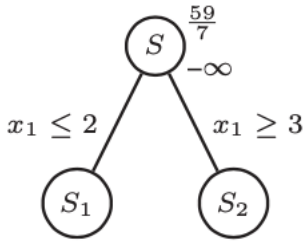
$$fix \;\; \underline{z} = \underline{z}_0 \;\; \& \;\; \overline{z} = \infty \;\; if \; no \; feas.sol.found \;\; (min.)$$

- Si la solución de la relajación lineal es entera, esta es la solución del óptima del problema entero original. Si, en cambio, la relajación lineal no es factible, entonces el problema entero tampoco es factible

- Si $\overline{z} \neq \underline{z} \neq z^*$, es necesario dividir o ramificar el conjunto factible. Para ello, se divide la región factible escogiendo una variable entera que sea fraccional en la solución de la relajación y dividir el problema en dos con respecto al techo y al suelo de este valor fraccional. Por lo tanto, si $x_j = \overline{x_j} \in \mathbb{Z}$, se pueden crear dos subconjuntos de la siguiente forma:

$$S_1 = S \cap \{x : x_j \leq \lfloor \overline{x_j} \rfloor\} \qquad S_2 = S \cap \{x : x_j \geq \lceil \overline{x_j} \rceil\}$$

$$such \; that \;\; S = S_1 \cup S_2 \;\; \& \;\; S_1 \cap S_2 = \emptyset$$



  - Durante este paso, se crea una solución titular o *incumbent solution* $\boldsymbol{x}^*$, la cual es la mejor solución entera hasta el momento para el problema IP original. En el primer paso no se fija ningún valor para esta, pero en los siguientes se puede actualizar

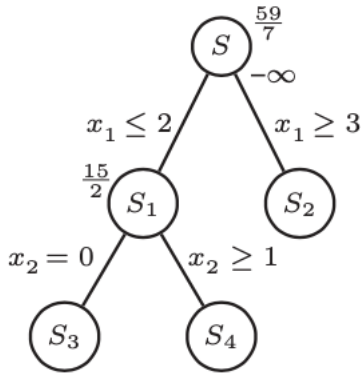
  - Además, el valor de cualquier solución factible del problema entero será una cota inferior al óptimo del problema, de modo que $z^*(\boldsymbol{x}^*) < z(\boldsymbol{x})$

- El tercer paso consiste en escoger arbitrariamente una de las ramificaciones (para comenzar) y resolver el problema relajado a un programa lineal

  - En cada relajación del problema, se añade como restricción el criterio usado para hacer la ramificación en base a la variable con parte fraccional. Si no es la primera ramificación que se hace, entonces se añaden como restricciones los criterios usados para todas las ramas anteriores al nodo

  - Una vez se solucionan la relajación, y se encuentra $\boldsymbol{x}_i^*$ para el problema i, se fija el valor de la función objetivo en ese nodo como la cota superior $\overline{z}_i = \boldsymbol{c}'\boldsymbol{x}_i$ (la cota inferior en el caso de minimización) para i

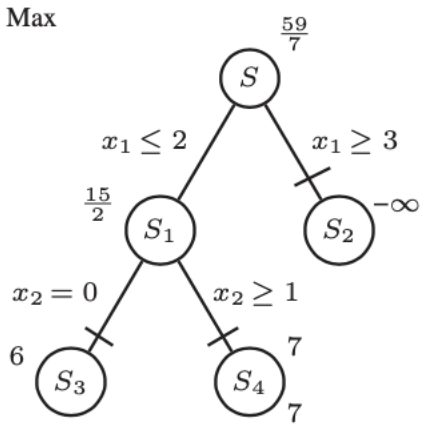
- Después de obtener una cota superior para el problema de maximización (inferior en el caso de minimización), se comprueba si esta rama i se puede podar utilizando los criterios de infactibilidad o de cota



- De no poderse podar, se divide o ramifica este problema en más subconjuntos factibles con respecto a la variable entera con un valor fraccional en la solución $\boldsymbol{x_i^*}$. Las ramificaciones de la ramificación escogida se siguen analizando hasta que se encuentra una solución entera $\boldsymbol{x_i^*}$

- Si la solución $\boldsymbol{x_i^*}$ es entera, se actualiza la solución incumbente $\boldsymbol{x^* = x_i^*}$ y se actualiza la cota inferior $\underline{z} = \max\{\underline{z} = -\infty, \underline{z_i}\}$ (en caso de maximización) o la cota superior $\overline{z} = \min\{\overline{z} = \infty, \overline{z_i}\}$ (en caso de minimización). Una vez hecha esta actualización, esta ramificación se poda por optimalidad

- El cuarto paso consiste en repetir el tercer paso, pero resolviendo las relajaciones de los otros problemas o nodos en el mismo nivel (en la misma ramificación) que no se resolvieron en el paso anterior, podando los resultados con los tres criterios descritos

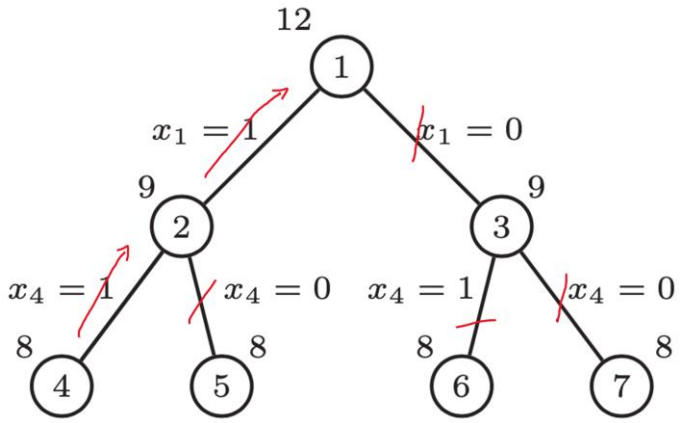


- Esto se hace porque puede ser que se encuentre una solución aún más óptima, que permita obtener una cota superior (o inferior en el caso de la minimización) mejor que la encontrada en el otro nodo

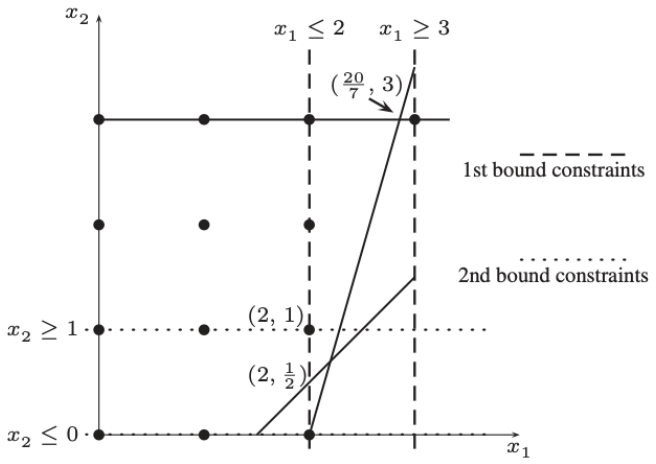
- Si no se puede podar una de las ramificaciones, entonces se tiene que dividir o ramificar ese problema aún más y volver a aplicar las instrucciones del tercer paso

- En el último paso, se comprueba si quedan problemas o nodos por resolver (volviendo hacia atrás en cada nivel el árbol). Si queda algún problema, se vuelve al tercer paso, pero si no queda ninguno, entonces la solución incumbente $\boldsymbol{x}^* = \boldsymbol{x}_i^*$ es el óptimo con un valor de $z = \overline{z}_i$ (de $z = \underline{z}_i$ en caso de minimización)

    - Si no quedan más problemas que resolver, entonces el algoritmo para y da como óptimo la solución incumbente descrita. De manera ilustrativa, se tendría un árbol de enumeración con un solo camino posible y en donde todos los otros arcos que conectan a los otros nodos están podados



- Gráficamente, el algoritmo de *branch and bound* permite restringir más la región hasta encontrar un punto óptimo



Los algoritmos de planos cortantes

- Los algoritmos de planos cortantes se basan en la idea de reformular el problema entero a través del concepto de casco convexo

    - A partir de la definición de casco convexo $conv(X)$ como el conjunto convexo más pequeño que contiene $X \subseteq \mathbb{R}^n$, se puede demostrar que este es un poliedro convexo

        - Para un problema entero, como $X = \{\boldsymbol{x} \in \mathbb{Z}^n : A\boldsymbol{x} \leq \boldsymbol{b}\}$, entonces el casco convexo se puede definir a través de desigualdades del tipo $\tilde{A}\boldsymbol{x} \leq \tilde{\boldsymbol{b}}$ que cumpla las condiciones vistas anteriormente

        - No obstante, el problema de esto es que, en general, el número de desigualdades necesarias para definir la envolvente convexa es muy grande

    - Cuanto más cerca se esté del casco convexo, mejor será la formulación, y los puntos extremos del casco son soluciones óptimas (como se ha visto anteriormente)

        - De este modo, si se soluciona un problema donde las restricciones se definen por el caso convexo, se obtendría directamente una solución entera

        - Siendo x_{RL}^1 y x_{RL}^2 los óptimos de las relajaciones lineales de P_1 y P_2, es obvio que los óptimos enteros son mayores o iguales a los de la relajación lineal, por lo que si $P_1 \subset P_2$ entonces $x_{RL}^2 \leq x_{RL}^1 = x^1 = x^2$. Por lo tanto, se prefiere P_1 a P_2 porque estar más cerca del casco convexo permite que ser esté mas cerca de los puntos enteros

- Los métodos basados en planos de corte consisten en ir secuencialmente aproximándose al casco convexo del problema inicial

    - Para hacer esto se añaden nuevas restricciones, que serán desigualdades válidas, que reducen la región factible de la relajación lineal sin descartar soluciones enteras factibles

    - Una desigualdad $\pi x \leq \pi_0$ es una desigualdad válida para $X \in \mathbb{R}^n$ si $\pi x \leq \pi_0$ es válida para toda $x \in X$. Si $X = \{x \in \mathbb{Z}^n : Ax \leq b\}$ y $conv(X) = \{x \in \mathbb{R}^n : \tilde{A}x \leq \tilde{b}\}$, las restricciones $a^i x \leq b_i$ y $\tilde{a}^i x \leq \tilde{b}_i$ son obviamente desigualdades válidas

- Un algoritmo para poder encontrar desigualdades válidas es el Chvátal-Gomory, el cual se basa en la obtención de desigualdades válidas para el problema entero para poder recortar el área de la región factible

    - Para poder entender como se pueden generar desigualdades válidas para programas enteros, primero es necesario entender como hacerlo para programas lineales

        - Una desigualdad $\pi x \leq \pi_0$ es válida para $P = \{x : Ax \leq b, x \geq 0\} \neq \emptyset$ si, y solo si, existe una $u \geq 0$ y una $v \geq 0$ tal que $uA - v = \pi$ y $ub \leq \pi_0$. Alternativamente, es válida para P si, y solo si, existe una $u \geq 0$ tal que $uA \geq \pi$ y $ub \leq \pi_0$

        - Debido a la dualidad de los programas lineales, $\max\{\pi x : x \in P\} \leq \pi_0$ si, y solo si, $\min\{ub : uA - v = \pi, u \geq 0, v \geq 0\} \leq \pi_0$ o si $\min\{ub : uA \geq \pi, u \geq 0, v \geq 0\} \leq \pi_0$

    - Si se considera una región factible de un programa entero $\{x : Ax \leq b, x \in \mathbb{Z}_+^n\}$, entonces se puede conseguir una desigualdad válida haciendo una simple observación

        - Siendo $X = \{x \in \mathbb{Z}^1 : x \leq b\}$, entonces la desigualdad $x \leq \lfloor b \rfloor$ es válida para X. Esto ocurre debido a que si b es entera, entonces $x \leq \lfloor b \rfloor = b$, y si no lo es, entonces $x \leq \lfloor b \rfloor$ se cumple porque x es entero

    - El procedimiento de Chvátal-Gomory para construir una desigualdad válida para el conjunto $X = P \cap \mathbb{Z}^n$ donde $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$, A es una matriz $m \times n$ con columnas $\{a_1, a_2, \dots, a_n\}$ y $u \in \mathbb{R}_+^m$ es el siguiente:

        - La desigualdad $\sum_{j=1}^n ua_j x_j \leq ub$ es una desigualdad válida para P dado que $u \geq 0$ y $\sum_{j=1}^n a_j x_j \leq b$

- La desigualdad $\sum_{j=1}^{n}\lfloor ua_j \rfloor x_j \leq ub$ es una desigualdad válida para P dado que x es un entero y, por tanto $\sum_{j=1}^{n}\lfloor ua_j \rfloor x_j$ será un entero que cumplirá esta desigualdad

- La desigualdad $\sum_{j=1}^{n}\lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$ es una desigualdad válida para X dado que $\sum_{j=1}^{n}\lfloor ua_j \rfloor x_j$ es un entero y como este era menor a ub, también lo tiene que ser para $\lfloor ub \rfloor$ por la lógica anteriormente mencionada

  - Cualquier desigualdad válida para X se puede obtener aplicando el procedimiento de Chvátal-Gomory un número finito de veces

- Uno de los algoritmos más utilizados para poder generar desigualdades válidas para un programa entero es el algoritmo de planos cortantes fraccionales de Gomory, obteniendo así los conocidos cortes de Gomory

  - Considerando un problema $\min\{c'x : Ax = b, x \in \mathbb{Z}_+^n\}$, la idea es resolver primero la relajación y encontrar la base óptima, escoger una variable básica que no sea entera y generar una desigualdad Chvátal-Gomory sobre la restricción asociada con la variables básica de modo que corte la solución del programa lineal

    - Suponiendo que se tiene una base óptima, el problema se puede reescribir de la siguiente forma:

$$\min \boldsymbol{c_B'x_B} + \boldsymbol{c_D'D} \;\; s.t. \;\; \boldsymbol{Bx_B} + \boldsymbol{Dx_D} = \boldsymbol{b} \;\; \& \;\; \boldsymbol{x} \geq \boldsymbol{0}$$

    - Siendo $\boldsymbol{x}^*$ la solución óptima al problema, y sabiendo que esta se puede dividir en $\boldsymbol{x_B^*} \geq \boldsymbol{0}$ y $\boldsymbol{x_D^*} = \boldsymbol{0}$, es posible obtener la siguiente expresión:

$$\boldsymbol{Bx_B} + \boldsymbol{Dx_D} = \boldsymbol{b} \;\; \Rightarrow \;\; \boldsymbol{x_B} + \boldsymbol{B^{-1}Dx_D} = \boldsymbol{B^{-1}b}$$

    - Si $\boldsymbol{x_B}$ no es entera, significa que para algunas filas i $\boldsymbol{x_B} = \boldsymbol{B^{-1}b} \notin \mathbb{Z}$ (dado que $\boldsymbol{B^{-1}Dx_D} = \boldsymbol{0}$)

    - Cada elemento en $\boldsymbol{Y} = \boldsymbol{B^{-1}D}$ se puede expresar como $Y_{ij} = u_{ij} + f_{ij}$, en donde u_{ij} es un número entero y $f_{ij} \in (0,1)$, y los elementos de $\widetilde{\boldsymbol{b}} = \boldsymbol{B^{-1}b}$ se pueden expresar como $\lfloor \tilde{b}_i \rfloor + f_i$, donde $f_i \in (0,1)$. Por lo tanto, se pueden obtener las siguientes igualdades:

$$x_B^i + \sum_{j \in \boldsymbol{D}} (u_{ij} + f_{ij}) x_N^j = \lfloor \tilde{b}_i \rfloor + f_i$$

$$\Rightarrow\ x_B^i + \sum_{j\in \boldsymbol{D}} u_{ij}x_N^j + \sum_{j\in \boldsymbol{D}} f_{ij}x_N^j = \lfloor \tilde{b}_i \rfloor + f_i$$

$$\Rightarrow\ x_B^i + \sum_{j\in \boldsymbol{D}} u_{ij}x_N^j - \lfloor \tilde{b}_i \rfloor = f_i - \sum_{j\in \boldsymbol{D}} f_{ij}x_N^j$$

- Dado que $\sum_{j\in \boldsymbol{D}} f_{ij}x_N^j > 0$, entonces $1 > f_i > f_i - \sum_{j\in \boldsymbol{D}} f_{ij}x_N^j$ y eso hace que $\sum_{j\in \boldsymbol{D}} f_{ij}x_N^j \geq f_i$ sea una desigualdad válida para cualquier valor de x_N^j, conocida como el corte de Gomory

- Esta también se puede expresar en términos de $\boldsymbol{Y}$, en donde $y_{ij} - \lfloor y_{ij} \rfloor$ es la parte fraccional de y_{ij} y $\tilde{b}_i - \lfloor \tilde{b}_i \rfloor$ es la parte fraccional de :

$$\sum_{j\in \boldsymbol{D}} (y_{ij} - \lfloor y_{ij} \rfloor) x_D^j \geq \tilde{b}_i - \lfloor \tilde{b}_i \rfloor$$

- Las variables de holgura de la restricción que proviene del corte de Gomory (ya que una vez hecho el corte se añade al problema de optimización) deben ser variables enteras

- Un procedimiento fácil a seguir cuando se aplica este algoritmo en un programa entero es el siguiente:

  - Primero se calcula el valor de las variables básicas a través de su expresión matricial

$$\widetilde{\boldsymbol{b}} = \boldsymbol{B}^{-1}\boldsymbol{b}$$

  - El segundo paso consiste en calcular el valor de la matriz $\boldsymbol{D}$ en términos de la base $\boldsymbol{B}$

$$\boldsymbol{Y} = \boldsymbol{B}^{-1}\boldsymbol{D}$$

  - El tercer paso consiste en dividir la parte fraccional de la parte entera de cada matriz para poder construir los cortes de Gomory, los cuales serán expresados por una desigualdad matricial del siguiente tipo, donde $\boldsymbol{x_D}$ son las variables no básicas:

$$\boldsymbol{Y} = \boldsymbol{Y^{int}} + \boldsymbol{Y^f} \ \ \& \ \ \widetilde{\boldsymbol{b}} = \widetilde{\boldsymbol{b}}^{\boldsymbol{int}} + \widetilde{\boldsymbol{b}}^{\boldsymbol{f}}$$

$$\Rightarrow\ \boldsymbol{Y^f x_D} \geq \widetilde{\boldsymbol{b}}^{\boldsymbol{f}}$$

  - Finalmente, el corte de Gomory que se incluye en las restricciones será el de la variable básica i con una parte

fraccional más alta en valor absoluto en $\widetilde{\boldsymbol{b}}$, y una vez añadida al problema original se vuelve a solucionar el programa lineal

- Si se necesita realizar otro corte de Gomory, se hace el procedimiento descrito a partir de la nueva solución

Las heurísticas primales

- Dado que muchos de los problemas prácticos son $\mathcal{NP}$-difíciles, los algoritmos de heurísticas tienen un rol importante a la hora de obtener soluciones factibles buenas para problemas de optimización discreta
    - Existen varias razones por las cuales alguien se puede decantar al uso de la heurística, algunas de estas son las siguientes:
        - Se requiere una solución rápidamente, ya sea en segundos o en minutos
        - Las instancias son tan largas y/o tan complicadas que no se puede formular como un problema de programación entera o de programación entera mixta de tamaño razonable
        - Aunque se puede formular como un programa entero mixto, es imposible encontrar soluciones factibles buenas para el sistema de *branch and bound*
        - Para algunos problemas combinatorios es más fácil encontrar soluciones factibles por inspección o por conocimiento de la estructura del problema
    - Al diseñar y usar una heurística, algunas de las dudas o preguntas más importantes que uno se hace son las siguientes:
        - Poder aceptar cualquier solución factible o preguntarse a posteriori que tan lejos está del óptimo
        - Garantizar a priori que la heurística produce una solución dentro de una ϵ o $\alpha\%$ de optimalidad+
- Las dos heurísticas más utilizadas son la de búsqueda codiciosa o *greedy search* y la de búsqueda local o *local search*, las cuales se pueden formalizar de manera matemática y proporcionan maneras fáciles de obtener cotas primales
    - Para formalizar el algoritmo de la heurística de *greedy search*, se considera un problema combinatorio de la siguiente forma:

$$\min_{\mathcal{S}\subseteq\mathcal{N}}\{c(\mathcal{S}) : v(\mathcal{S}) \geq k\}$$

- Se asume que el conjunto vacío no es factible, de modo que se cumpla la igualdad $v(\emptyset) = c(\emptyset) = 0$ y la función v no es decreciente

- También se asume que $v(\mathcal{N}) \geq k$, ya que de otra manera el problema no es factible

- Los pasos del algoritmo de la heurística de *greedy search* son los siguientes:

  - Se fija $\mathcal{S}^0 = \emptyset$ (se comienza con el conjunto vacío) y $t = 1$

  - Después, se fija j_t al elemento que minimiza el coste adicional (expresado como $c(\mathcal{S}^{t-1} \cup \{j_t\}) - c(\mathcal{S}^{t-1})$) por añadir una unidad adicional (expresado como $v(\mathcal{S}^{t-1} \cup \{j_t\}) - v(\mathcal{S}^{t-1})$)

$$j_t = \arg\min \frac{c(\mathcal{S}^{t-1} \cup \{j_t\}) - c(\mathcal{S}^{t-1})}{v(\mathcal{S}^{t-1} \cup \{j_t\}) - v(\mathcal{S}^{t-1})}$$

  - Si la solución previa $\mathcal{S}^{t-1}$ es factible, entonces la solución factible titular es $\mathcal{S}^G = \mathcal{S}^{t-1}$. De no ser así, se fija $\mathcal{S}^t = \mathcal{S}^{t-1} \cup \{j_t\}$

  - Si $t = n$, entonces el algoritmo se para fijando $\mathcal{S}^G = \mathcal{N}$ como solución factible final del algoritmo, pero si no, entonces se vuelve al segundo paso fijando $t = t + 1$

- Este algoritmo se tiene que adaptar a la estructura del problema particular

  - La heurística del vecino más cercano o *nearest neighbour* es una heurística de *greedy search* y comienza en un nodo arbitrario y construye un camino desde ese nodo de una manera similar a la heurística anterior presentada

  - La heurística de codicioso puro o *pure greedy* escoge un elemento *j* de mínimo coste de modo que $\mathcal{S}^t$ consiste de un conjunto de caminos disjuntos hasta que el último elemento forma un ciclo

- Para formalizar el algoritmo de la heurística de *local search*, se considera un problema combinatorio de la siguiente forma:

$$\min_{\mathcal{S} \subseteq \mathcal{N}} \{c(\mathcal{S}) : g(\mathcal{S}) = 0\}$$

- En este caso $g(\mathcal{S}) \geq 0$ representa una medida de no factibilidad del conjunto $\mathcal{S}$, por lo que la restricción $v(\mathcal{S}) \geq k$ usada se puede representar como $g(\mathcal{S}) = \big(k - v(\mathcal{S})\big)^+$

- Para un algoritmo de búsqueda local, se necesita definir una solución, una vecindad local $Q(\mathcal{S})$ para toda solución $\mathcal{S} \subseteq \mathcal{N}$ y una función $f(\mathcal{S})$ que puede ser igual a $c(\mathcal{S})$ cuando $\mathcal{S}$ es factible e infinita de otro modo, o puede ser una función compuesta de la forma $c(\mathcal{S}) + \alpha g(\mathcal{S})$ consistiendo de una combinación ponderada del valor de la función objetivo y un múltiplo positivo de la medida de no factibilidad para $\mathcal{S}$

- Los pasos del algoritmo de la heurística de *local search* son los siguientes:

  - Se escoge una solución inicial $\mathcal{S}$ y se busca un conjunto $\mathcal{S}' \in Q(\mathcal{S})$ con $f(\mathcal{S}') < f(\mathcal{S})$

  - Si no existe ningún conjunto así, el algoritmo se para y $\mathcal{S}^H = \mathcal{S}$ es el óptimo local. De otro modo, se fija $\mathcal{S} = \mathcal{S}'$ y se vuelve al primer paso