Aspectos Computacionales de Programación Lineal Entera

Segundo Cuatrimestre de 2019

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico

The Stable Set Problem: Clique and Nodal Inequalities Revisited

Integrante	LU	Correo electrónico
Jessica Singer	177/16	singer.jeess@gmail.com

En este trabajo se incluye una descripción detallada del paper titulado The Stable Set Problem: Clique and Nodal Inequalities Revisited, introduciendo los conceptos de Programación Lineal Entera necesarios para su entendimiento

Linear Integer Programming **Branch and Cut** **Stable Set Problem**

Índice

1. Introducción

El problema que trata el trabajo en cuestión es el de *Conjunto Independiente Máximo*, también llamado Conjunto Estable Máximo.

Como sabemos, este es un problema *NP-Difícil* para el cual se conocen distintos tipos de soluciones, cada cual con sus limitaciones. Existen algoritmos, como el que se verá en este trabajo, cuya limitante puede ser la cantidad de nodos. Existen otros cuya limitante es que funciona únicamente para ciertas familias de grafos, como por ejemplo el caso de los grafos bipartitos.

Según se menciona, los algoritmos más utilizados para este problema al momento de escribir el paper son los combinatorios. Los autores proponen otras posibles resoluciones usando la técnica de programación lineal entera. Este tipo de soluciones ya se han estudiado, sin embargo, hasta el momento dichas técnicas no venían resultando demasiado prácticas o eficientes. (Ver, por ejemplo, [2]).

Se propondrán en el trabajo en cuestión, distintas formulaciones del problema (que como veremos, traerán aparejadas distintas resoluciones). Se introduen a continuación algunas nociones básicas de programación lineal entera para poder comprender las ideas principales del paper:

Definición 1.1: Programación Lineal

La Programación Lineal es una caracterización de problemas que constan de una función objetivo que se quiere maximizar cumpliendo una serie de restricciones lineales. Su formulación es la siguiente:

$$\begin{aligned} &\textit{Encontrar un vector } x \in \mathbb{R}^{nx1} \textit{ tal que} \\ &\textbf{Maximiza: } c^\intercal x_i \textit{ (función objetivo)} \\ &\textbf{Sujeto a: } Ax \leq b \textit{ (restricciones)} \\ &x_i \geq 0 \, \forall \, i \in 1..n \end{aligned}$$

Con c $\in \mathbb{R}^{nx1}$, A $\in \mathbb{R}^{mxn}$, b $\in \mathbb{R}^{mx1}$

Definición 1.2: Región Factible

El conjunto de soluciones de un problema de programación lineal (esto es, los x que cumplan las restricciones) se llama **región factible**.

Lema 1.1

El conjunto de soluciones de un problema de programación lineal entera es convexo.

Demostración. Sean x_1, x_2 soluciones válidas para $Ax \leq b, x_i \geq 0$. Sea $t \in (0, 1)$.
Veamos que $x_1 t + (1-t)x_2$ es solución:

$$\begin{aligned} A(x_1 t + (1-t)x_2) \leq b \iff \\ t.Ax_1 + (1-t).Ax_2 \leq b \iff \\ t.b + (1-t).b \leq b \text{ pues } x_1 \text{ y } x_2 \text{ son soluciones.} \end{aligned}$$

□

Definición 1.3: Cápsula Convexa

Una cápsula convexa de un conjunto S es el conjunto convexo más pequeño que contiene a todos los elementos de S, es decir:

$$\bigcap_{\substack{S \subset C_i \\ C_i convexo}} C_i \tag{1}$$

Naturalmente, nuestro conjunto de soluciones de programación lineal X es una cápsula convexa, ya que es convexo.

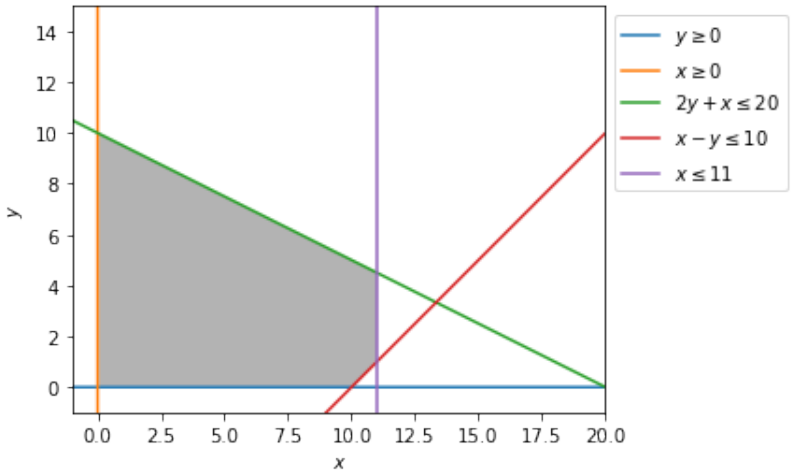

Figura 1: Región factible para la solución de un conjunto de ecuaciones

Definición 1.4: Polítopo Convexo

Un polítopo convexo es una cápsula convexa de un conjunto finito de puntos en un espacio euclideano.

Lema 1.2

El conjunto de soluciones X de programación lineal es un *polítopo convexo*.

Esto no lo demostraré, sólo daré una idea intuitiva; pensemos en nuestro conjunto de soluciones en $\mathbb{R}$:

Podemos tomar las soluciones mínima y máxima, y sabemos que como el conjunto es convexo, lo que esté en medio también es solución.

En el caso general, utilizamos la cápsula convexa de los *puntos extremos*. Intuitivamente, éstos serían los que conocemos como vértices en 2 y 3 dimensiones. La definición formal se incluye a continuación.

Definición 1.5: Puntos extremos

Sea P un polítopo convexo. Un elemento $x \in P$ se dice extremo si no se encuentra en medio de otros dos elementos de P. Es decir, no existe $t \in [0,1]$, $y, z \in P \setminus \{x\}$ tales que $x = ty + (1-t)z$.

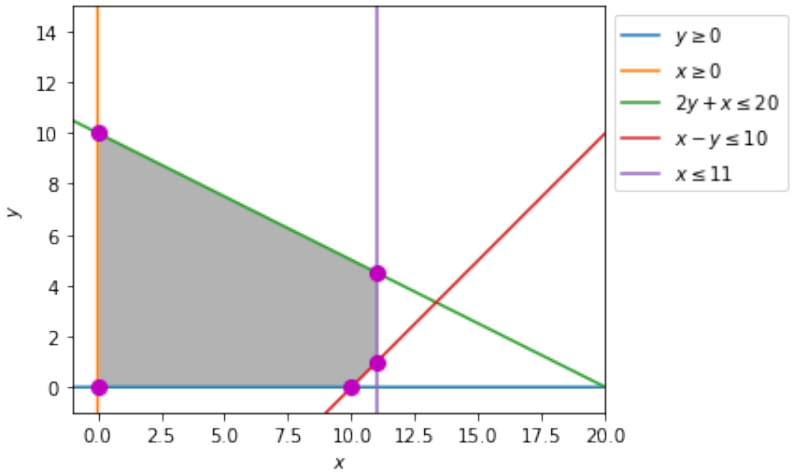

Figura 2: Puntos extremos marcados en la región factible del ejemplo anterior

Notemos que los puntos extremos están en el *borde* del polítopo, pero sin embargo no todo punto en el borde es un punto extremo.

Teorema 1.1

Dado un problema de programación lineal, existe una solución óptima en uno de los puntos extremos de su polítopo de soluciones.

Una de las técnicas más conocidas para resolver problemas de programación lineal es el algoritmo SIMPLEX. La solución que encuentra este método es uno de los extremos.

Definición 1.6: Programación Lineal Entera

Un problema de **Programación Lineal Entera** es un problema de programación lineal donde x $\in \mathbb{Z}^n$, es decir, el vector de soluciones es entero.

Existe una variante de PLE donde sólo consideramos $x \in \{0,1\}^n$. La llamaremos formulación 0-1 PL.

Ejemplo 1.1

Tomemos como ejemplo el siguiente problema:

- $c^t = (0, 1)$ con $c^t x = x_2$ la función lineal a maximizar.
- $Ax \leq b$ sistema lineal definido por:

$$2x_2 - x_1 \leq 1$$
$$x_1 + x_2 \leq 5$$
$$x_1 \leq 3$$

En este caso, podemos marcar a mano las soluciones enteras y así hallar el óptimo.

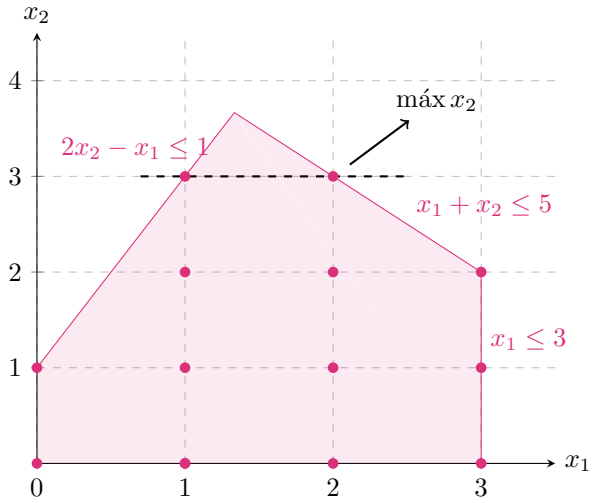

Figura 3: Representación gráfica del problema

1.1. Resolución de Problemas de Programación Lineal Entera

Hay una diferencia sustancial entre problemas de Programación Lineal y Programación Lineal Entera: mientras que los primeros siempre son resolubles en tiempo polinomial, los segundos incluyen conocidas formulaciones de problemas *NP-Difícil*.

Definición 1.7: Relajación Lineal

Dado un problema de PLE, llamamos **relajación lineal** a la solución del problema quitando la restricción de integralidad.

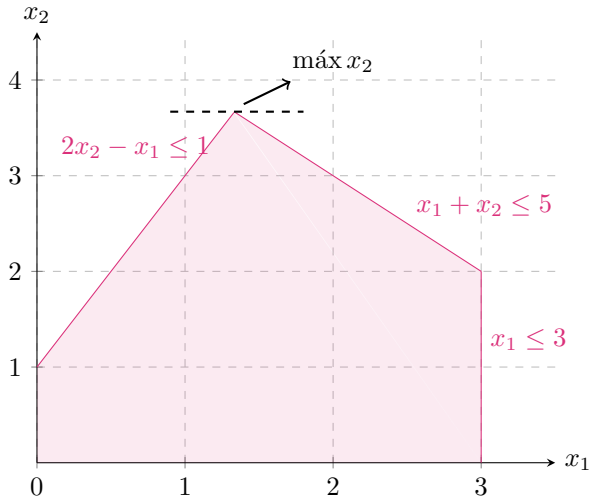

Figura 4: Relajación lineal del problema 1

1.1.1. Branch and Bound

La primera de las técnicas es la de Branch and Bound. No incluiré detalles de este método ya que no es el foco del paper en cuestión. Se puede leer al respecto en [3].

1.1.2. Algoritmos de planos de corte

Este método consta de los siguientes pasos:

1. Primero, corremos la relajación lineal sobre el problema, obteniendo un $\overline{x}$ fraccionario. Esto lo podemos hacer en tiempo polinomial utilizando SIMPLEX.
2. Luego, si el $\overline{x}$ resulta ser entero, ya tenemos la solución. Sino introduciremos nuevas desigualdades que la solución fraccionaria no cumpla, para poder descartarla. Esto va a ser el llamado *plano de corte* y va a depender fuertemente de la naturaleza de cada problema.
3. Repetimos el paso 1

Ejemplo 1.2

Tomemos nuevamente el ejemplo 1.

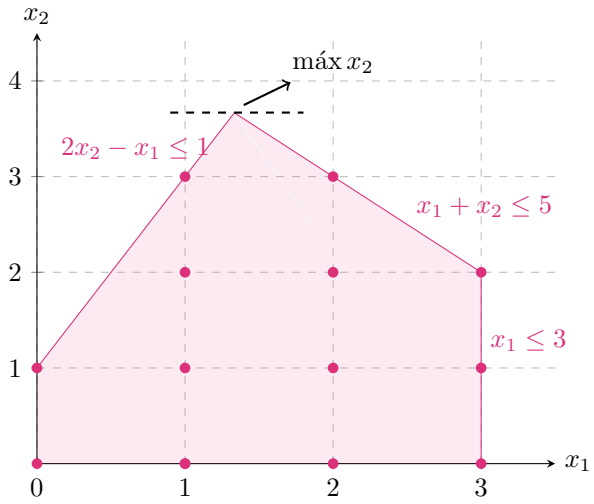

Figura 5: Representación gráfica del problema

Notemos que la solución que nos da la relajación lineal es fraccionaria. Exactamente $3{,}66$.

Lo que hacemos entonces es añadir otra desigualdad al conjunto, que nos restrinja dicha solución. En este caso es fácil ver que añadir la desigualdad $x2 \leq 3$ nos da el resultado esperado. Esta desigualdad es lo que se llama un *corte*.
La mayor parte de los casos no son tan sencillos como este ejemplo. Las desigualdades son condiciones sobre nuestra estructura de soluciones, y para añadir una nueva puede ser necesario estudiar cada problema en particular. Encontrar un corte útil en algunos casos puede ser bastante costoso.

A continuación daré algunas definiciones geométricas que se utilizan fuertemente en el paper. La idea detrás de éstas es generalizar conceptos que nos son familiares en $\mathbb{R}^2$, como por ejemplo las caras de un polígono.

Definición 1.8: Caras de un polítopo

Una *cara* de un polítopo P es un conjunto de puntos $\{x \in P : \ \alpha^\intercal x = \beta\}$, donde $\{x \in P : \ \alpha^\intercal x \leq \beta\}$ es una desigualdad válida en P.

Notemos que la definición anterior difiere de lo que conocemos como cara en la geometría en 2 dimensiones, ya que podemos tener, para una figura de n dimensiones, caras de 1, 2, 3,.., n-1 dimensiones.
A las caras de dimensión 1 las llamaremos los *bordes* del polítopo, mientras que a las de dimensión $n-1$ las llamaremos *facetas*.

Figura 6: En amarillo, faceta de un cubo, viéndolo como un polítopo convexo en $\mathbb{R}^3$

Definición 1.9: Desigualdad dominante

Sean $\gamma, \pi \in \mathbb{R}^{1xn}$, $x \in \mathbb{R}^{nx1}$, $\gamma_0, \pi_0 \in \mathbb{R}$. Una desigualdad $\gamma x \leq \gamma_0$ domina a $\pi x \leq \pi_0$ si y sólo si existe $\mu > 0$ tal que $\mu\pi \leq \gamma$ y $\gamma_0 \leq \mu\pi_0$.

Las *facetas* son un tipo de corte fuertes, ya que no son dominadas por ningún otro corte. Buscaremos cortes que definan facetas ya que éstos, por su dimensión, son de alguna manera los cortes *más grandes* que podemos hacer.

A diferencia de otros problemas, en Programación Lineal Entera la manera de *formular* un problema va a ser determinante en su resolución. Esto es, la elección de las desigualdades a utilizar.

1.2. Formulación clásica de Conjunto Independiente Máximo

Sea $G = (V, E)$. Consideremos la siguiente formulación del problema de Conjunto Independiente máximo:

$$\begin{aligned} \text{Maximizar: } & \sum_{i \in V} x_i \\ \text{Sujeto a: } & x_i + x_j \leq 1, (i,j) \in E \\ & x_i \in \{0,1\}, i \in V \end{aligned}$$

Aquí, $x_i = 1 \iff$ el nodo i está dentro del conjunto independiente.
Notemos que las desigualdades nos dicen que si tenemos un eje entre dos nodos, a lo sumo uno de ellos puede estar en el conjunto independiente.

Tomemos por ejemplo el siguiente grafo:

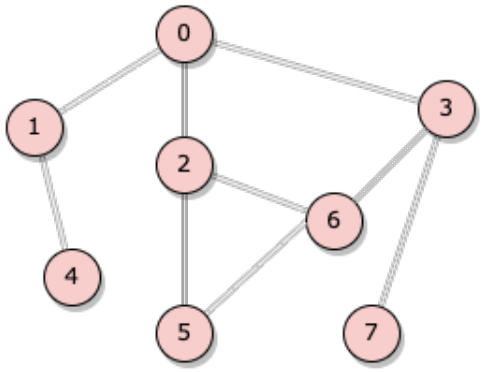

Figura 7: Grafo G

Nos queda el siguiente conjunto de desigualdades:

$$\begin{aligned}
\text{Maximizar: } & \sum_{i \in [0,10]} x_i \\
\text{Sujeto a: } & x_0 + x_1 \leq 1 \\
& x_0 + x_2 \leq 1 \\
& x_0 + x_3 \leq 1 \\
& x_1 + x_4 \leq 1 \\
& x_2 + x_5 \leq 1 \\
& x_2 + x_6 \leq 1 \\
& x_3 + x_6 \leq 1 \\
& x_3 + x_7 \leq 1 \\
& x_5 + x_6 \leq 1 \\
& x_i \in \{0,1\}, i \in V
\end{aligned}$$

Si eligieramos agregar el nodo 0 a nuestro CIM, i.e., $x_0 = 1$, para que se cumplan las restricciones, no podríamos agregar el 1, ni 2, ni 3.

El problema de esta formulación radica en que su relajación lineal no aporta a priori mucha información, ya que es setear todas las variables que aparezcan en las restricciones en $1/2$, y las restantes en 1.

En el paper, se intentará mejorar la formulación del problema y luego se demostrará que la misma tiene un profundo impacto en la eficacia del método, logrando resolver instancias que hasta entonces nunca habían podido ser resueltas utilizando métodos de programación lineal entera.

2. Formulaciones del Problema

En esta sección introduciremos distintas familias de desigualdades utilizadas en el paper, y luego veremos la formulación final del problema en base a las mismas.

2.1. Desigualdades de Clique

La primera familia de desigualdades son las Desigualdades de Clique. La idea detrás de éstas es que si tenemos una clique, definitivamente no vamos a tener más de un nodo de la misma dentro del conjunto independiente.
Notemos que en la formulación clásica se utiliza una idea similar, ya que un eje define una clique entre dos nodos.

Las desigualdades de clique se definen de la siguiente manera:

$$\sum_{i \in C} x_i \leq 1,\ C \text{ clique}$$

Si la clique en cuestión es maximal, se puede probar que esa desigualdad define una faceta.

La primera mejora que se propone es reemplazar las desigualdades de los ejes por desigualdades clique, extendiendo cada par de nodos a una clique maximal, ya que éstas últimas son más fuertes.

Si se toma el caso del grafo anterior 7, podriamos agregar la restricción $x_2 + x_5 + x_6 \leq 1$, la cual es más estricta que tener únicamente desigualdades de los ejes.

2.2. Desigualdades Nodales

Las siguientes desigualdades propuestas son las Nodales.

La idea es, en principio, sumar todas las desigualdades de ejes por cada nodo, de tal manera que reducimos la cantidad de desigualdades de $\|E\|$ a $\|V\|$.

$$\sum_{j \in n(i)} x_j + \|n(i)\| x_i \leq \|n(i)\| \tag{2}$$

Notemos que nos indica, si se incluye el nodo i, no se puede agregar a ninguno de sus vecinos en el conjunto independiente máximo. Sino, se pueden llegar a poner a todos sus vecinos.

Esta desigualdad no aporta mejores resultados que las desigualdades originales, pero se propone una mejora donde en lugar del número de vecinos del nodo i, se utiliza el resultado de CIM para el subgrafo inducido por los vecinos:

Definición 2.1: Desigualdad de Rango

Las desigualdades de rango, se definen de la siguiente manera:

$$\sum_{i \in S} x_i \leq r(S),\ S \subseteq V$$

Donde r(S) es el tamaño del conjunto independiente máximo para el subgrafo inducido por S.

$$\sum_{j \in n(i)} x_j + r(n(i)) x_i \leq r(n(i)) \tag{3}$$

Finalmente las desigualdades nodales van a ser las siguientes:

$$\sum_{j \in S(i)} x_j + r(S(i)) x_i \leq r(S(i)) \qquad \text{Desigualdad Nodal}$$

Con $S(i) \subseteq n(i)$.

Una de las propuestas será combinar desigualdades de clique especialmente seleccionadas, con desigualdades nodales.

2.3. Mejorando las Desigualdades Nodales

2.3.1. Descomposición

La primera propuesta del paper será la descomposición de una desigualdad nodal. En el paper se utiliza el concepto de *dominación* de una manera levemente distinta, porque se habla de conjuntos de desigualdades que dominan a una.

Si tenemos algo como (Desigualdad Nodal), con $S = S(n(i))$, y tenemos los conjuntos $S_1, ..., S_t$ tales que $\sum_{k=1}^{t} r(S_k) = r(S)$, entonces diremos que la desigualdad está *dominada* por las desigualdades:

$$\sum_{j \in S_k} x_j + r(S_k)x_i \leq r(S_k) \quad k = 1,...,t \tag{4}$$

Podemos ver que la suma de esas desigualdades nos da la desigualdad original. Este método nos permite, si logramos descomponer *lo suficiente*, poder resolver problemas pequeños y manejables y con eso resolver el problema original.

El paper nos da dos métodos para descomponer las desigualdades nodales; El primero es simplemente partir S en sus componentes conexas. Me enfocaré en dar una explicación un poco más detallada del segundo método:

Definición 2.2: Vértice Simplicial

Sea $G = (V, E)$, $v \in V$ se dice *simplicial* sii $\{v\} \cup n(v)$ forman una clique.

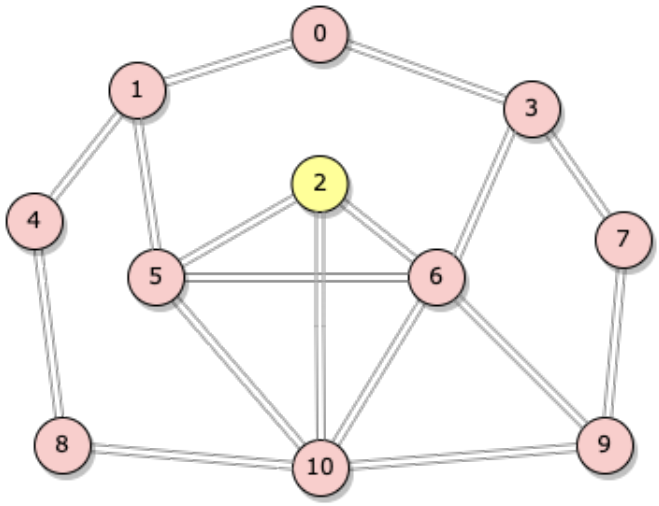

Figura 8: En amarillo, vértice simplicial del grafo.

El segundo método utiliza los vértices simpliciales restringidos a S, es decir, a un vértice $j \in S$ le dice simplicial si $j \cup (S \cap n(j))$ forman una clique. Llamaremos a esa clique $\mathcal{C}$.

Al estar calculando el conjunto independiente, podremos tener a lo sumo, un nodo por cada clique.

Se puede probar que el conjunto independiente máximo va a contener *exactamente* un nodo de esta clique. La demostración no se incluye en el paper, pero es sencilla; si no tuvieramos ningún vértice de

este conjunto, agregamos el vértice simplicial, ya que sabemos que no está ninguno de sus vecinos en la clique máxima.

Con esto tendremos

$$r(S) = r(S \setminus \mathcal{C}) + 1 \tag{5}$$

Si pensamos en S como $S = (S \setminus \mathcal{C}) \cup \mathcal{C}$ podemos tener las siguientes desigualdades:

$$\sum_{j \in S \setminus \mathcal{C}} x_j + r(S \setminus \mathcal{C}) x_i \leq r(S \setminus \mathcal{C}) \tag{6}$$

$$\sum_{j \in \mathcal{C} \cup \{i\}} x_j \leq 1 \tag{7}$$

Más aún, se sugiere agrandar la clique obtenida del vértice simplicial $\mathcal{C} \cup \{i\}$ a una clique maximal en el grafo original. Con esto tenemos otra forma de descomponer una desigualdad nodal por dos desiguales que la dominan.

Luego este proceso puede iterarse, ya que $S \setminus \mathcal{C}$ podría no ser conexo.

La primera propuesta del paper es iterar este proceso para producir desigualdades más fuertes.

2.4. Lifting

El lifting consiste en tomar una desigualdad válida y extenderla a un espacio de más dimensiones, preservando su validez.[1] En este caso, se toma una desigualdad nodal y se extiende a una desigualdad de la forma que describe el paper:

$$\sum_{j \in n(i)} \beta_j x_j + r(S) x_i \leq r(S) \tag{8}$$

Con $\beta_j \geq 1$ para $j \in S \subset n(i)$ y $\beta_j \geq 0$ para $j \notin S$

Una intuición de por qué sería interesante este lifting, puede estar dado por el siguiente lema que se introduce al inicio del paper:

Lema 2.1

Sea un grafo $G = (V, E)$, un subconjunto $S \subseteq V$. Dada una desigualdad de rango, si su lado derecho es mayor a 1 y define una faceta en el polítopo del conjunto estable máximo de G[S] (lo notaremos STAB(G[S]), entonces existe al menos una desigualdad *lifteada* o alzada (lifted inequality) de la forma:

$$\sum_{i \in S} x_i + \sum_{i \in V \setminus S} \beta_i x_i \leq r(S)$$

Que define una faceta de STAB(G).

Ejemplo 2.1

En el paper se incluye un ejemplo utilizando un grafo 5-hole con dos vértices conectados al resto (figura 9).
Si tomamos $i = 7$ y $S = \{1, ..., 6\}$, tenemos que el conjunto independiente máximo de S, $r(S)$ es 2.
Esto se traduce en la desigualdad nodal:

$$\sum_{j=1}^{6} x_j + 2x_7 \leq 2 \tag{9}$$

Lo que se marca en este ejemplo es que si bien $r(S) = 2$, al estar el nodo 6 conectado a todos los otros, no puede estar éste y uno más dentro del conjunto independiente máximo.
Esto se expresa multiplicando al x_6 por $r(S) = 2$.

La idea detrás de este lifting es restringir soluciones inválidas, como por ejemplo tomar como solución $x_1 = x_6 = 1$. Esto se expresa en la siguiente desigualdad *lifteada*:

$$\sum_{j=1}^{5} x_j + 2x_6 + 2x_7 \leq 2 \tag{10}$$

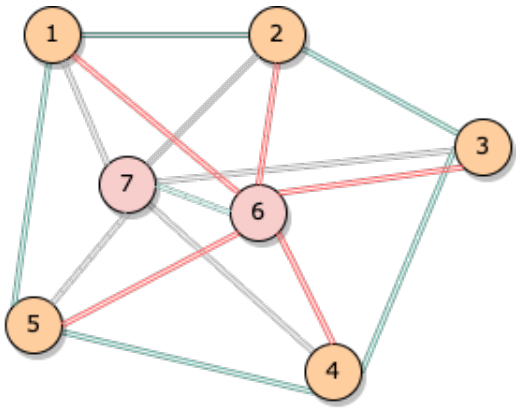

Figura 9: Grafo 5-hole con dos vértices conectados al resto.

El lifting por sí sólo no es un método de resolución, ni una desigualdad válida. Los autores mencionan que no sólo calcular $r(S)$ es $\mathcal{NP}$-difícil, sino que también lo es hallar un β_j lo más grande posible.

Para lo primero, se propone aplicar descomposición de forma tal que tengamos un grafo lo más pequeño posible al momento de calcular el conjunto independiente máximo.

Para lo segundo, una propiedad útil que se menciona es que al tener un nodo $k \in n(i) \setminus S$ que no es vecino de nadie en S, entonces vamos a tener $\beta_k = 0$. Esto porque al no ser vecino de nadie de S, siempre podremos incluirlo al conjunto independiente máximo del subgrafo conformado por los nodos en S.

Se toma la decisión de liftear únicamente las variables de los nodos en S, para reducir el tiempo de cómputo, con lo cual la extensión introducida previamente no se termina utilizando.
No se ahonda en el algoritmo utilizado para el lifting, ni en esta sección ni en las posteriores. Es una lástima ya que es un algoritmo importante dentro del paper, y en la experimentación se podrá ver que tiene un impacto significativo en el tiempo de cómputo para ciertos grafos.

2.5. Formulaciones reducidas

Una limitante que se menciona con respecto al método anterior, es que con grafos muy densos, las desigualdades también son muy densas, y ésto puede traer aparejado que resolver la relajación lineal en cada paso haga más costoso.

A continuación detallaré las distintas alternativas que se proponen:

2.5.1. Primera idea: quitar ejes

Lo primero que se propone es partir de un conjunto de cliques $\mathcal{C}$, luego armar las desigualdades de cliques para ese conjunto, y agregar las desigualdades nodales pero **sin tener en cuenta los ejes que ya utilicé en las cliques**, es decir, tomando el grafo $\hat{G} = (V, E \backslash E(\mathcal{C}))$ como se menciona en el paper.

Ejemplo 2.2

Consideremos el grafo de la figura 10. Los ejes que corresponden a desigualdades de clique están marcados con rosa, mientras que los otros están marcados con gris.

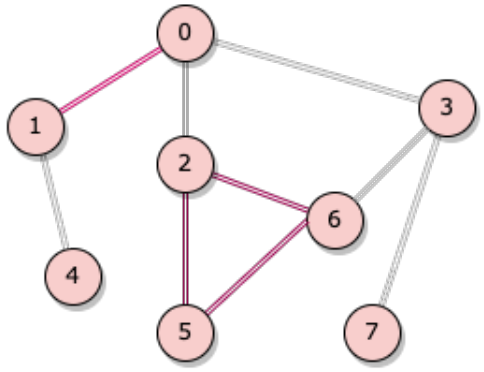

Figura 10: Grafo G. Los ejes pintados con rosa marcan las desigualdades de clique.

Considerando la formulación original sin lifting, y tomando una desigualdad nodal por cada vértice con $S = n(i)$, nos quedan las siguientes restricciones:

$$\begin{aligned}
Nodales: \quad & 3x_0 + x_1 + x_2 + x_3 \leq 3 \\
& 2x_1 + x_0 + x_4 \leq 2 \\
& 2x_2 + x_0 + x_5 + x_6 \leq 2 \\
& 3x_3 + x_0 + x_6 + x_7 \leq 3 \\
& x_4 + x_1 \leq 1 \\
& 2x_6 + x_2 + x_3 + x_5 \leq 2 \\
& x_7 + x_3 \leq 1 \\
Clique: \quad & x_2 + x_5 + x_6 \leq 1 \\
& x_0 + x_1 \leq 1
\end{aligned}$$

Con el método de reducción propuesto, se reducen a las siguientes:

$$\begin{aligned}
Nodales: \quad & 2x_0 + x_2 + x_3 \leq 2 \\
& x_1 + x_4 \leq 1 \\
& x_2 + x_0 \leq 1 \\
& 3x_3 + x_0 + x_6 + x_7 \leq 3 \\
& x_4 + x_1 \leq 1 \\
& x_6 + x_3 \leq 1 \\
& x_7 + x_3 \leq 1 \\
Clique: \quad & x_2 + x_5 + x_6 \leq 1 \\
& x_0 + x_1 \leq 1
\end{aligned}$$

Como se puede observar, el resultado son desigualdades menos densas.

Cuantos menos vértices involucrados haya, más sencillo es calcular $r(S)$, pero hay que tener en cuenta que estamos *debilitando* las desigualdades. Necesitaremos encontrar un punto medio entre tener restricciones fuertes que produzcan soluciones enteras, y tener desigualdades que no ocupen mucho tiempo de

cómputo en generarse y resolver su relajación lineal.
Parte del método constará entonces de la elección correcta de cliques.Se proponen las siguientes alternativas:

1. El primer método es utilizar únicamente desigualdades nodales, es decir, $\mathcal{C} = \emptyset$
2. La segunda idea es utilizar el algoritmo 1 del paper para generar una coleccion de cliques. Luego con las desigualdades de clique, resolver la relajación lineal y quedarnos únicamente con aquellas desigualdades donde se haya cumplido la igualdad (se le llaman *binding inequalities*).
3. La tercera idea consta de varias partes:
   - 3.1 Se construye una formulación con desigualdades de clique como la del ítem anterior,
   - 3.2 Se corre la relajación lineal, obteniendo un $\overline{x}$ fraccionario.
   - 3.3 Se toman 3 conjuntos: $L = \{j \in \|V\| : \overline{x}_j = 0\}$, $F = \{j \in \|V\| : 0 \leq \overline{x}_j \leq 1\}$, $U = \{j \in \|V\| : \overline{x}_j = 1\}$.
   - 3.4 Se define la función de pesos; Sea $i \in V$:

$$f(i) = \begin{cases} \|n(i)\|\overline{x}_i & i \in F \cup L \\ -\infty & i \in U \end{cases}$$

   - 3.5 Se corre el algoritmo de plano de corte siguiendo la heurística descrita en el Algoritmo 2, pasándole G, F y f. Ésta retorna un conjunto de cliques $\mathcal{C}_{sep}$, las agregamos como desigualdades a la formulación de nuestro problema.
   - 3.6 Luego corremos nuevamente la relajación lineal, y elegimos como $\mathcal{C}_{cut}$ aquellas desigualdades donde haya valido la igualdad.

El algoritmo que se utiliza en el paso 3.5 es el siguiente:

Algorithm 1 Algoritmo 2

Datos: Grafo G = (V,E), conjunto ordenado F de variables fraccionarias, función de pesos f
Parámetros: Factor de tolerancia ϵ
Resultado: Una colección $\overline{\mathcal{C}}_{sep} = \{C_1, ..., C_k\}$ de cliques cuya desigualdad es incumplida.

```
Set G'(V', E') = G[F], k := 0          ▷ Tomo el subgrafo nodos cuyas variables son fraccionarias
while E' ≠ ∅ do
    j := argmax_{k∈V'}{f(k)}           ▷ Nodo con peso máximo.
    Computo clique máxima con pesos C, que contenga a (i,j)   ▷ Se hace una heurística greedy
    E' := E' \ E'(C)                   ▷ Quito los ejes de C
    if x̄(C) > 1 + ε then   ▷ Si los valores de los nodos de C suman más de 1, quiere decir que la desigualdad de clique inducida es un corte útil
        k := k + 1
        C_k := C                       ▷ La agrego al conjunto resultante
    end if
end while
f(j) := −1
```

2.5.2. Segunda idea: reducción de vértices

La segunda idea para reducir la densidad de las desigualdades, es que al elegir un vértice i para agregar una desigualdad nodal, luego se trabaje con $G' = (V \setminus \{i\}, E)$. Para maximizar esa reducción, se puede ordenar los vértices en orden decreciente por su cantidad de vecinos.

Ejemplo 2.3

Consideremos nuevamente el ejemplo 2.5.1. Al aplicarle ambas reducciones tomando los nodos en el orden numérico, tendríamos las siguientes restricciones, aún más reducidas:

$$\begin{aligned} Nodales: \quad & 2x_0 + x_2 + x_3 \leq 2 \\ & x_1 + x_4 \leq 1 \\ & 2x_3 + x_6 + x_7 \leq 3 \\ Clique: \quad & x_2 + x_5 + x_6 \leq 1 \\ & x_0 + x_1 \leq 1 \end{aligned}$$

3. Resultados

El objetivo de esta sección es no sólo dar una breve explicación de los resultados de la experimentación, sino también dar una noción de cómo se integraron las múltiples ideas introducidas anteriormente.

3.1. Generando la formulación

Lo siguiente que se describe en el trabajo es un algoritmo con el cual dado un grafo y ciertos parámetros pertinentes, se generan las desigualdades de la formulación de conjunto independiente máximo. Esto es, la formulación final.

Intentaré esclarecer un poco la idea detrás del mismo. Para no incurrir en señalar dentro del algoritmo (me gustaría mantener al mínimo la contaminación visual de este informe), escribiré los pasos en lenguaje coloquial y citaré las líneas de código del paper en caso que sea necesario:

1. Comenzamos construyendo un conjunto $\mathcal{C}$ de cliques de G.
2. Luego tomamos el vector $\overline{x}$ de la relajación lineal utilizando únicamente las desigualdades de clique con el conjunto anterior.
3. Tomamos $\mathcal{C}^* \subseteq \mathcal{C}$, el subconjunto de cliques tal que al tomar sus desigualdades de clique inducidas, el $\overline{x}$ genera la igualdad. ($\sum_{i \in \mathcal{C}} \overline{x}_i = 1$).
4. Fijamos E^*, el conjunto de ejes que pertenecen a las cliques de $\mathcal{C}^*$, y agregamos a P (el conjunto de desigualdades resultante, inicialmente vacío) esas desigualdades de clique.
5. Fijamos también H, como el mismo grafo pero quitándole esos ejes. Esta fue la primera estrategia de reducción contada en la sección anterior.
6. Una vez ordenados los vértices en orden descendente por su grado, tomamos un vértice i y aplicamos lo siguiente:
   I. Fijamos F como los vecinos de i, sujetos a los ejes de H
   II. Tomamos una componente conexa de F, $S = (V_S, E_S)$ (Este paso es el primer método de descomposición propuesto).
   III. Para cada vértice simplicial en S, tomamos una clique maximal extendida considerando todos los nodos de G. Agregamos a P dicha desigualdad de clique, y quitamos todos los nodos de la clique de S. (Este paso corresponde a la otra descomposición propuesta).
   IV. Luego de quitar estos vértices, puede que S sea $\emptyset$ en cuyo caso seguimos con otra componente conexa de F, o puede que ya no sea conexo, en cuyo caso agregamos las componentes conexas que quedan a F.
   V. Si S sigue siendo conexo, entonces computamos el conjunto independiente máximo en S (este paso no es polinomial, dependeremos fuertemente de que S sea un grafo pequeño).
   VI. Luego armamos la desigualdad nodal para el conjunto S, y la lifteamos si se indica en un parámetro de entrada. Agregamos dicha desigualdad al conjunto P.
   VII. Repetimos lo mismo para las demás componentes conexas de F.
   VIII. Por último, si tenemos habilitada la reducción descrita en la sección anterior, quitamos a i del conjunto de vértices del grafo G.
7. Repetimos el proceso para los otros vértices.

Como podemos ver, este proceso varía según que conjunto inicial de cliques escojamos. Éste será el punto fuerte de experimentación en el que se concentrarán los autores, intentando encontrar para distintos tipos de grafo un conjunto inicial que minimice el tiempo de cómputo de la solución.

3.2. Construcción de Instancias

Se utiliza una clase de grafos aleatorios, Erdős–Rényi, con distintos niveles de densidad fijados por una probabilidad d de que un eje aparezca o no.

Las otras familias de grafos son más extensas, se menciona que son un estándar en proveer benchmarks para el algoritmo de clique máxima. Se puede leer más acerca de estas instancias en la página oficial de DIMACS. Este conjunto de casos de prueba es consistente con lo que utilizaron otros papers, por ejemplo [4]. Cabe destacar que en ésta última referencia se incluyó el código de la implementación, lo cual hace mucho más claros los resultados.

3.2.1. Grafos Aleatorios

Para los grafos aleatorios, los autores notan que en el caso de grafos no muy ralos ($d \geq 0{,}3$), si no comenzamos calculando ciertas desigualdades de clique con alguno de los métodos propuestos en la sección 2.5.1, ya las desigualdades nodales no se pueden descomponer mucho.
Esto puede ser un problema ya que las desigualdades nodales incluyen el cálculo de un subproblema de CIM, y para que sea razonable calcular cada desigualdad, los grafos deberían ser pequeños. Es extraño que para grafos más bien densos no hayan podido descomponer utilizando vértices simpliciales.

Teniendo esto último en cuenta, se concluye en que no toma demasiado tiempo construir los grafos aleatorios.

3.2.2. Grafos DIMACS

Para estos grafos[1] es interesante notar primero que no se utilizan todas las familias del benchmark, (las más grandes no se corren siquiera). También que con ciertas instancias con densidad de alrededor 0.7 y 700 y 400 nodos respectivamente, el tiempo requerido en la formulación del problema es muy grande si no se utilizan desigualdades de clique.

Se menciona que el lifting de las desigualdades consume mucho tiempo. Recordemos que en ese paso hay que calcular un conjunto independiente máximo con pesos de manera exacta, con lo cual si la descomposición no es muy eficiente ese paso puede ser un enorme cuello de botella. Lamentablemente no se incluye el algoritmo utilizado para el lifting.

Parece ser que, al menos para generar las instancias, incluir algún conjunto de desigualdades de clique hace una diferencia sustancial en el tiempo de cómputo. Al ser esta únicamente la primera etapa, hay que ver en los resultados si este tiempo se compensa con una reducción del tiempo de resolución del problema.

3.2.3. Resultados para grafos aleatorios

Se realizaron varias comparaciones entre las distintas formulaciones; por ejemplo, en grafos más densos ($d \geq 0{,}4$) funcionan mejor los modelos que únicamente tienen desigualdades nodales que los que incluyen desigualdades de clique.
También se descubre que en las instancias con $d \geq 0{,}6$, la formulación clásica del problema ya resuelve fácilmente el problema debido a que se encuentra rápidamente una relajación lineal entera.

Para no reiterar los resultados que bien resume el paper en el último párrafo de dicha sección, destacaré dos detalles:

El primero es la aclaración de que al utilizar CPLEX para resolver las instancias (software de resolución para programación lineal entera), se agregan otro tipo de planos de corte que exceden lo que trata el paper.

[1] Un detalle presente en las tablas de resultados para este conjunto de grafos es que en los grafos SAN400_0.7, la densidad que aparece en las tablas es 0.3. Al citar las fuentes que menciona el paper, la densidad de dichas 3 instancias ronda los 0.7.

Siendo que CPLEX es un software utilizado muy ampliamente para este tipo de problemas, es razonable que se utilice para la experimentación, pero a la vez, dependiendo de la versión del mismo, podrían llegar a variar los planos de corte que se agregan.

El segundo punto es que la comparación se centra en comparar sus distintas propuestas. Sería muy interesante poder comparar con algún otro método que no hayan propuesto ellos, en particular relacionando con los algoritmos combinatorios.

3.2.4. Resultados para grafos DIMACS

Los resultados en este conjunto de grafos se muestran en algunos casos similares a los grafos aleatorios, conjuntos más densos responden mejor a utilizar únicamente desigualdades nodales. Esto nos indica que se compensa el tiempo que toma formar las instancias.
Hay una excepción en dos tipos de grafos, C-FAT y SAN donde las desigualdades de clique funcionan mejor que utilizar únicamente nodales.
Asimismo se destaca que en un tercio de las instancias, la reducción de las desigualdades, que a priori uno pensaría que daría un resultado más débil, funciona mejor. Estiman que la reducción facilita utilizar descomposiciones, y así lograr un resultado más veloz.

Se concluye nombrando instancias que no habían sido resueltas antes por ningún algoritmo de programación lineal entera, lo cual otorga un gran valor a este trabajo. También se pudieron resolver otras 3 instancias *difíciles* pero ya utilizando paralelización y configuraciones más adhoc.

Nuevamente no se incluyen muchas comparaciones con algoritmos combinatorios, se menciona que funcionan mejor en la mayoría de los casos, resolviendo más instancias. En algunos grafos puntuales con una configuración específica, pudieron alcanzar mejores resultados que los algoritmos combinatorios, pero este último punto es algo escueto, ya que no se menciona contra qué algoritmo están comparando.

4. Conclusiones

La idea de los autores parece ser volver a poner sobre la mesa los algoritmos de programación entera para resolver el problema de conjunto independiente máximo. Según mencionan, están algo desterrados en pos de algoritmos combinatorios. Notemos que éstos últimos pueden incluir heurísticas o soluciones específicas a ciertas familias de grafos, mientras que los métodos propuestos en este trabajo son exactos y generalizables a cualquier tipo de grafo. También se encuentran propiedades que vale la pena considerar a futuro, como la relación entre la densidad del grafo y el rendimiento de utilizar desigualdades de clique.

Las desigualdades que se proponen no involucran una matemática muy compleja, y se mejora la performance enormemente respecto a métodos anteriores de programación lineal entera, incluso resolviendo instancias que hasta entonces no se habían podido resolver.

Nuevamente es una lástima que no se incluya el algoritmo de lifting utilizado. Al momento de escribir esto, envié un mail aún sin respuesta a los autores consultando este punto.

Mi intención con este trabajo es que se haya podido comprender el paper, introduciendo los conceptos pertinentes de programación lineal entera, haciendo hincapié en demostraciones no muy detalladas, e intentando desglosar algoritmos muy extensos. Espero sinceramente que este trabajo sirva como guía para cualquier persona que quiera introducirse en el tema.

Referencias

[1] Gu, Zonghao & Nemhauser, George & Savelsbergh, Martin. (2000). Sequence Independent Lifting in Mixed Integer Programming. Journal of Combinatorial Optimization. 4. 109-129. 10.1023/A:1009841107478.

[2] S. Rebennack, "Stable Set Problem: Branch & Cut Algorithms". In Encyclopedia of Optimization (C. Floudas y P. Pardalos, editores). Springer (2008) 3676-3688.

[3] G. Cornuéjols, M. Trick y M. Saltzman, .A tutorial on integer programming". CMU Technical Report (1995).

[4] Prosser, Patrick. (2012). Exact Algorithms for Maximum Clique: A Computational Study. Algorithms. 5. 10.3390/a5040545