Counting Complexity

Miguel Romero

17 de Abril del 2012

Problemas de Conteo

NP = existencia de *certificados* polinomiales.

Dada fórmula proposicional φ, existe asignación que haga verdadera a φ?

A veces queremos *contar* la cantidad de certificados.

Dada fórmula proposicional φ, cuantas asignación hacen verdadera a φ?

Dado grafo bipartito G, cuantos matchings perfectos hay en G?

Problemas de Conteo

NP = existencia de *certificados* polinomiales.

Dada fórmula proposicional φ, existe asignación que haga verdadera a φ?

A veces queremos *contar* la cantidad de certificados.

Dada fórmula proposicional φ, cuantas asignación hacen verdadera a φ?

Dado grafo bipartito G, cuantos matchings perfectos hay en G?

Problemas de Conteo

NP = existencia de *certificados* polinomiales.

Dada fórmula proposicional φ, existe asignación que haga verdadera a φ?

A veces queremos *contar* la cantidad de certificados.

Dada fórmula proposicional φ, cuantas asignación hacen verdadera a φ?

Dado grafo bipartito G, cuantos matchings perfectos hay en G?

La clase #P

Queremos capturar los problemas de conteo asociados a NP, i.e., las funciones que cuentan certificados polinomiales.

Definición (#P, Valiant 79')

Una función $f : \{0,1\}^* \to \mathbb{N}$ *está en* $\#P$ *si existe un polinomio* p *y una MT a tiempo polinomial* M *tal que para todo* $x \in \{0,1\}^*$*:*

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

La clase #P

Como en el caso de NP, podemos caracterizar #P usando MT no deterministas.

Definición (#P, definición alternativa)

Una función $f : \{0,1\}^* \to \mathbb{N}$ *está en* #P *si existe una MT no determinista a tiempo polinomial* M *tal que para todo* $x \in \{0,1\}^*$, $f(x)$ *es igual a la cantidad de ramas de aceptación de* M *sobre la entrada* x.

#P-completitud

f es #P-hard si todo problema en #P se reduce a f y es #P-completo si está en #P y es #P-hard.

Qué reducción escoger en #P?

Definición (parsimonius reduction)

$f \leq_{par} g$ ssi existe una función $\alpha : \{0,1\}^* \to \{0,1\}^*$ computable en tiempo polinomial tal que para todo $x \in \{0,1\}^*$, $f(x) = g(\alpha(x))$.

Es decir, existe un mapeo entre los problemas, tal que preserva el número de certificados/soluciones.

#P-completitud

f es #P-hard si todo problema en #P se reduce a f y es #P-completo si está en #P y es #P-hard.

Qué reducción escoger en #P?

Definición (parsimonius reduction)

$f \leq_{par} g$ ssi existe una función $\alpha : \{0,1\}^* \to \{0,1\}^*$ computable en tiempo polinomial tal que para todo $x \in \{0,1\}^*$, $f(x) = g(\alpha(x))$.

Es decir, existe un mapeo entre los problemas, tal que preserva el número de certificados/soluciones.

#SAT es #P-completo

#SAT:
Dada fórmula proposicional φ, cuantas asignación hacen verdadera a φ?

Usando una modificación del Teorema de Cook-Levin se tiene que

Teorema

#SAT es #P-completo, via parsimonius reductions.

Muchos otros problemas de conteo asociados a problemas NP-completos son #P-completos via este tipo de reducción.

#SAT es #P-completo

#SAT:
Dada fórmula proposicional φ, cuantas asignación hacen verdadera a φ?

Usando una modificación del Teorema de Cook-Levin se tiene que

Teorema

#SAT es #P-completo, via parsimonius reductions.

Muchos otros problemas de conteo asociados a problemas NP-completos son #P-completos via este tipo de reducción.

Otro tipo de reducciones

Las parsimonius reductions son muy restrictivas.

Si f se reduce a g via una parsimonius reduction entonces el problema de decisión asociado a f se reduce al problema de decisión asociado a g via una reducción mucho a uno.

Sólo problemas de conteo cuyos problemas de decisión son NP-completos pueden ser #P-completos via estas reducciones.

Nos gustaría descubrir problemas #P-completos cuyos problemas de decisión sean faciles. Necesitamos otro tipo de reducción.

Otro tipo de reducciones

Las parsimonius reductions son muy restrictivas.

Si f se reduce a g via una parsimonius reduction entonces el problema de decisión asociado a f se reduce al problema de decisión asociado a g via una reducción mucho a uno.

Sólo problemas de conteo cuyos problemas de decisión son NP-completos pueden ser #P-completos via estas reducciones.

Nos gustaría descubrir problemas #P-completos cuyos problemas de decisión sean faciles. Necesitamos otro tipo de reducción.

Otro tipo de reducciones

Las parsimonius reductions son muy restrictivas.

Si f se reduce a g via una parsimonius reduction entonces el problema de decisión asociado a f se reduce al problema de decisión asociado a g via una reducción mucho a uno.

Sólo problemas de conteo cuyos problemas de decisión son NP-completos pueden ser #P-completos via estas reducciones.

Nos gustaría descubrir problemas #P-completos cuyos problemas de decisión sean faciles. Necesitamos otro tipo de reducción.

#P-completitud via Turing reductions

Definición (turing reduction)

$f \leq g$ ssi f se puede resolver con una MT polinomial con acceso a g como oráculo.

Es decir, podemos resolver rapidamente f usando a g como "caja negra".

#P-completitud via Turing reductions

Definición (turing reduction)

$f \leq g$ ssi f se puede resolver con una MT polinomial con acceso a g como oráculo.

Es decir, podemos resolver rapidamente f usando a g como "caja negra".

Calculando el Permanente

Para $n \geq 1$, denotamos por S_n al conjunto de todas las permutaciones de $\{1, ..., n\}$ en $\{1, ..., n\}$.

Definición

Sea A una matriz de $n \times n$ con entradas enteras. Definimos el permanente *de A ($perm(A)$) como:*

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

Cual es la complejidad de computar el permanente?

Calculando el Permanente

Para $n \geq 1$, denotamos por S_n al conjunto de todas las permutaciones de $\{1, ..., n\}$ en $\{1, ..., n\}$.

Definición

Sea A una matriz de $n \times n$ con entradas enteras. Definimos el permanente *de A ($perm(A)$) como:*

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

Cual es la complejidad de computar el permanente?

Calculando el Permanente

Relacionado al permanente se encuentra el *determinante*:

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

El determinante se puede calcular en tiempo polinomial en la representación de A.

Calculando el Permanente

Relacionado al permanente se encuentra el *determinante*:

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

El determinante se puede calcular en tiempo polinomial en la representación de A.

Permanente es #P-completo

El comportamiento para el permanente es totalmente distinto.

Teorema (Valiant's Theorem 79')

Computar el permanente para matrices $0, 1$ *es #P-completo.*

Primer resultado de #P-completitud, donde el problema de decisión asociado es fácil.

Permanente es #P-completo

El comportamiento para el permanente es totalmente distinto.

Teorema (Valiant's Theorem 79')

Computar el permanente para matrices $0, 1$ *es #P-completo.*

Primer resultado de #P-completitud, donde el problema de decisión asociado es fácil.

Permanente $0,1 = \#$Matching Perfectos

Sea $G = (X \cup Y, E)$ un grafo bipartito donde $X = \{x_1, ..., x_n\}$ e $Y = \{y_1, ..., y_n\}$. Un matching perfecto de G es un conjunto de n arcos de G no adyacentes.

Para el grafo G la representación natural es la matriz (de adyacencia) A de $n \times n$ con entradas $0, 1$, donde $A_{i,j} = 1$ ssi $\{x_i, y_j\} \in E$.

Hay una biyección entre matching perfectos de G y permutaciones $\sigma \in S_n$ que cumplen que $A_{i,\sigma(i)} = 1 \quad \forall 1 \leq i \leq n$ (o equivalentemente, $\prod_{i=1}^{n} A_{i,\sigma(i)} = 1$). Luego

$$\#\text{Matching Perfectos}(G) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)} = perm(A)$$

Permanente $0,1$ = #Matching Perfectos

Sea $G = (X \cup Y, E)$ un grafo bipartito donde $X = \{x_1, ..., x_n\}$ e $Y = \{y_1, ..., y_n\}$. Un matching perfecto de G es un conjunto de n arcos de G no adyacentes.

Para el grafo G la representación natural es la matriz (de adyacencia) A de $n \times n$ con entradas $0, 1$, donde $A_{i,j} = 1$ ssi $\{x_i, y_j\} \in E$.

Hay una biyección entre matching perfectos de G y permutaciones $\sigma \in S_n$ que cumplen que $A_{i,\sigma(i)} = 1 \quad \forall 1 \leq i \leq n$ (o equivalentemente, $\prod_{i=1}^{n} A_{i,\sigma(i)} = 1$). Luego

$$\#\text{Matching Perfectos}(G) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)} = perm(A)$$

Otra forma del Teorema de Valiant

Teorema (Valiant's Theorem 79')

#Matching Perfectos es #P-completo.

Verificar la existencia de un matching perefecto en un grafo bipartito se puede hacer en tiempo polinomial.

Luego, efectivamente, el problema de decisión asociado es fácil.

Otra forma del Teorema de Valiant

Teorema (Valiant's Theorem 79')

#Matching Perfectos es #P-completo.

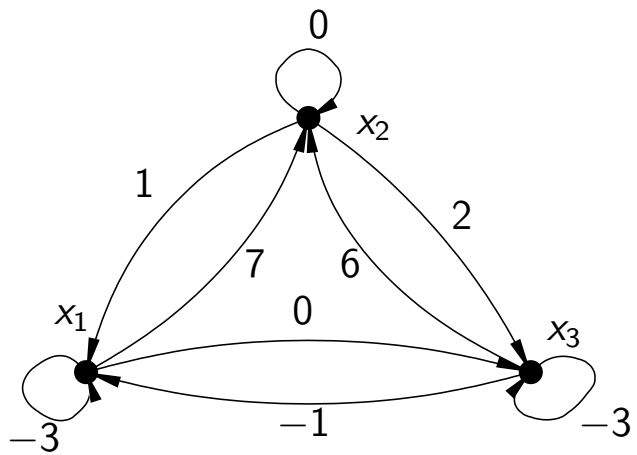
Verificar la existencia de un matching perefecto en un grafo bipartito se puede hacer en tiempo polinomial.

Luego, efectivamente, el problema de decisión asociado es fácil.

Definición combinatorial del Permanente

Sea A una matriz de $n \times n$ con entradas enteras. Consideramos A como la matriz de adyacencia de un grafo dirigido completo G con pesos en los arcos. El peso de (x_i, x_j) es $A_{i,j}$.

Ejemplo: $A = \begin{pmatrix} -3 & 7 & 0 \\ 1 & 0 & 2 \\ -1 & 6 & -3 \end{pmatrix}$



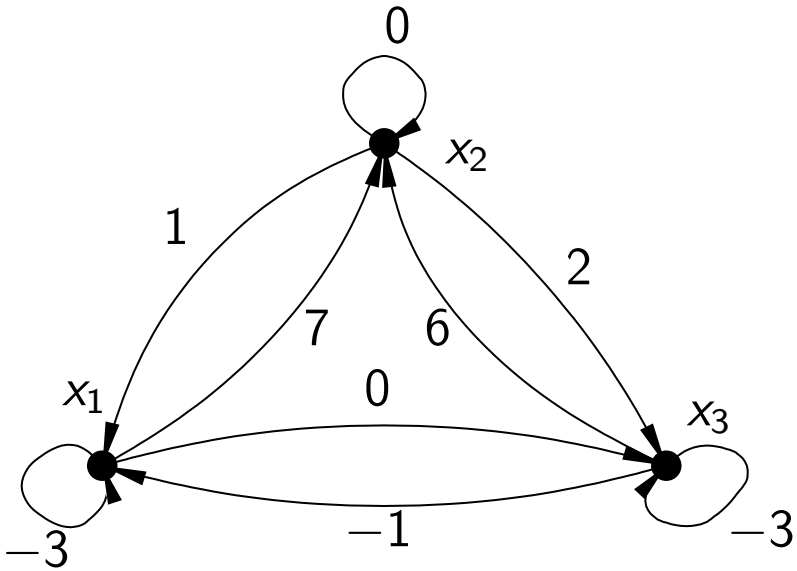
Cycle covers

Definición

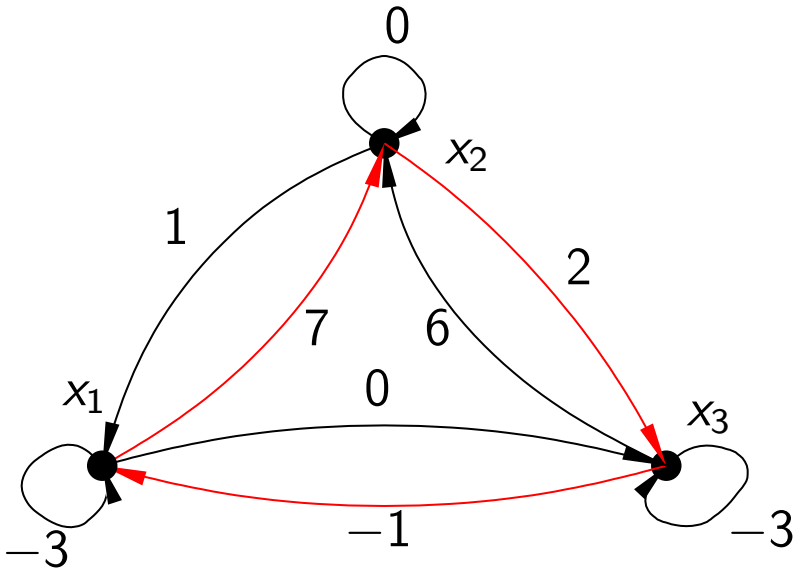
Un cycle cover de G es un subgrafo donde cada nodo tiene grado de entrada y de salida igual a 1, i.e., es un conjunto disjunto de ciclos dirigidos que cubre todos los nodos.
El peso del cycle cover C, denotado $w(C)$ es el producto de los pesos de todos los arcos que participan C. Definimos

$$PesoCycleCovers(G) = \sum_{C \text{ es cycle cover}} w(C)$$

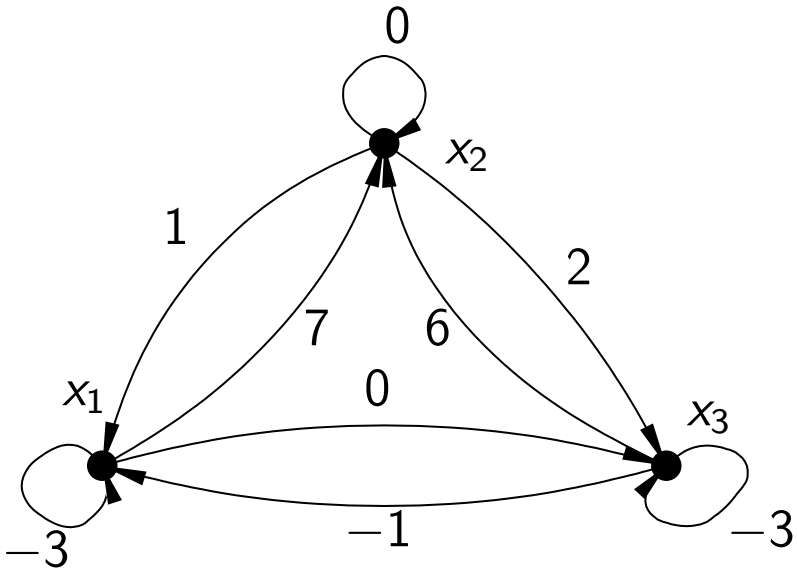
Peso cycle covers



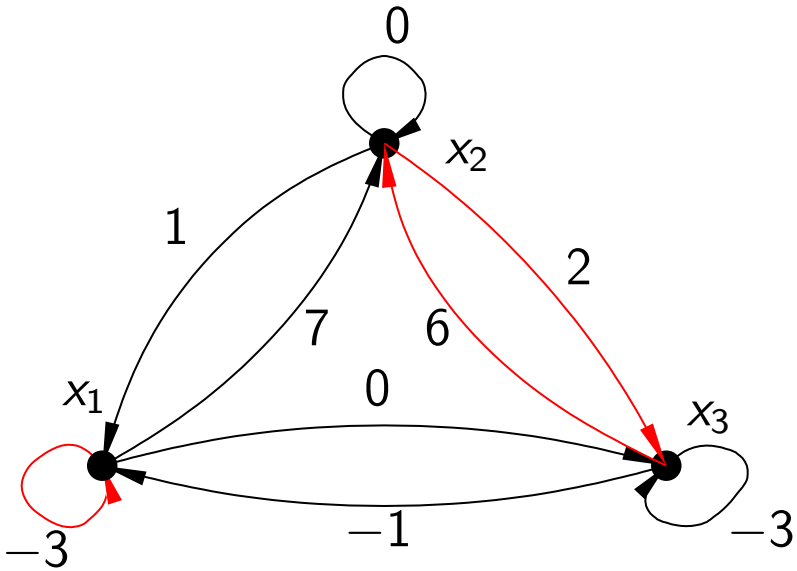
Peso cycle covers



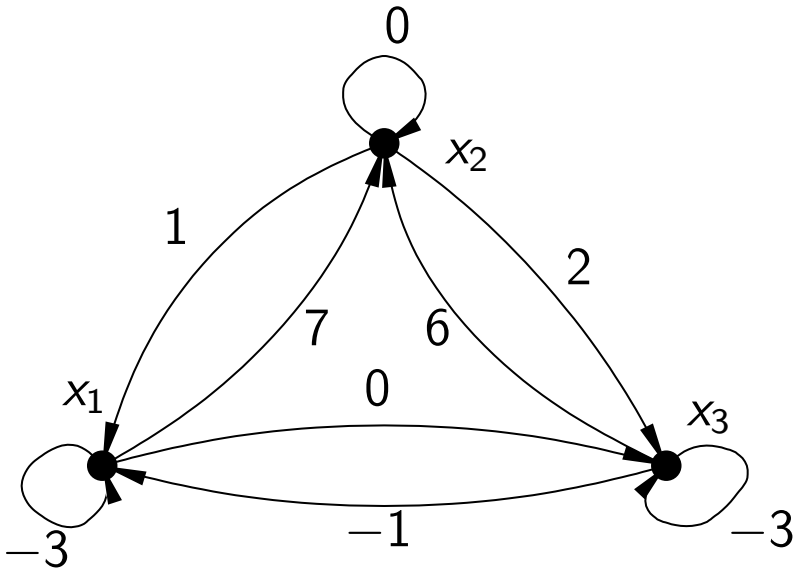
Peso cycle covers



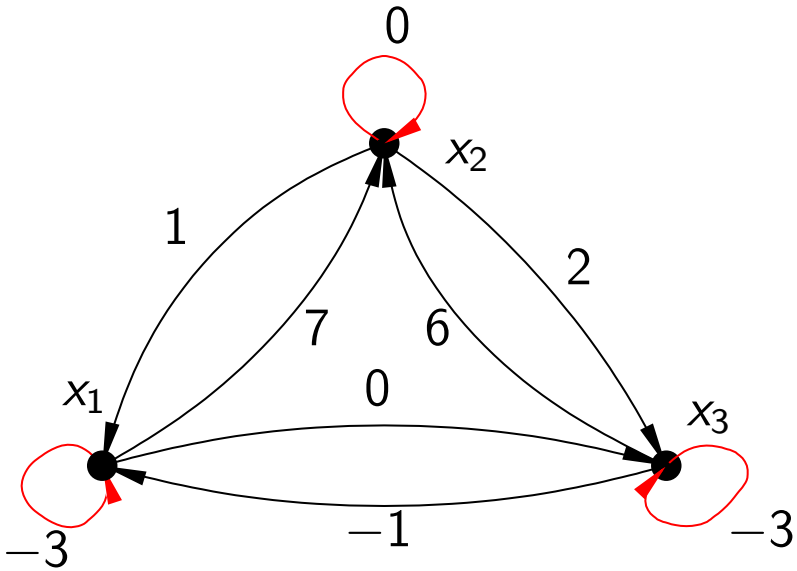
Peso cycle covers



Peso cycle covers



Peso cycle covers



Cycles covers y permutaciones

Sea G un grafo dirigido completo con pesos en los arcos y A la matriz que lo representa. Supongamos que los vertices de G son $\{1, ..., n\}$

Podemos identificar permutaciones en S_n con cycles covers de G.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 4 & 1 & 6 & 2 \end{pmatrix} \rightarrow (\ 1\ \ 3\ \ 4\ )(\ 2\ \ 5\ \ 6\ )$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 1 & 4 & 6 & 2 \end{pmatrix} \rightarrow (\ 1\ \ 3\ )(\ 2\ \ 5\ \ 6\ )(\ 4\ )$$

$$(\ 6\ \ 1\ \ 3\ \ 2\ )(\ 5\ \ 4\ ) \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 2 & 5 & 4 & 1 \end{pmatrix}$$

$$(\ 4\ \ 2\ \ 1\ )(\ 6\ \ 3\ )(\ 5\ ) \rightarrow \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 6 & 2 & 5 & 3 \end{pmatrix}$$

Cycles covers y permutaciones

Sea G un grafo dirigido completo con pesos en los arcos y A la matriz que lo representa. Supongamos que los vertices de G son $\{1, ..., n\}$
Podemos identificar permutaciones en S_n con cycles covers de G.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 4 & 1 & 6 & 2 \end{pmatrix} \quad \rightarrow \quad (\ 1 \quad 3 \quad 4\ )(\ 2 \quad 5 \quad 6\ )$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 1 & 4 & 6 & 2 \end{pmatrix} \quad \rightarrow \quad (\ 1 \quad 3\ )(\ 2 \quad 5 \quad 6\ )(\ 4\ )$$

$$(\ 6 \quad 1 \quad 3 \quad 2\ )(\ 5 \quad 4\ ) \quad \rightarrow \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 2 & 5 & 4 & 1 \end{pmatrix}$$

$$(\ 4 \quad 2 \quad 1\ )(\ 6 \quad 3\ )(\ 5\ ) \quad \rightarrow \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 6 & 2 & 5 & 3 \end{pmatrix}$$

PesoCyclesCovers=Perm

Mas aún si σ es una permutación en S_n y C el cycle cover asociado, entonces

$$\prod_{i=1}^{n} A_{i,\sigma(i)} = w(C)$$

Luego,

$$PesoCycleCovers(G) = perm(A)$$

PesoCyclesCovers=Perm

Mas aún si σ es una permutación en S_n y C el cycle cover asociado, entonces

$$\prod_{i=1}^{n} A_{i,\sigma(i)} = w(C)$$

Luego,

$$PesoCycleCovers(G) = perm(A)$$

PesoCyclesCovers=Perm

En general, podemos considerar el siguiente problema de conteo:

Dado un grafo dirigido G con pesos enteros y no nulos en los arcos, computar $PesoCycleCover(G)$.

Este problema es equivalente al problema de computar $PesoCycleCover(G)$ para un grafo dirigido completo G con pesos enteros, el cual a su vez, como ya vimos, es equivalente a computar el permanente de una matriz entera.

Abuso de Notación:
Para un grafo dirigido con pesos enteros y no nulos, denotaremos $perm(G)$ a $PesoCycleCover(G)$.

PesoCyclesCovers=Perm

En general, podemos considerar el siguiente problema de conteo:

Dado un grafo dirigido G con pesos enteros y no nulos en los arcos, computar $PesoCycleCover(G)$.

Este problema es equivalente al problema de computar $PesoCycleCover(G)$ para un grafo dirigido completo G con pesos enteros, el cual a su vez, como ya vimos, es equivalente a computar el permanente de una matriz entera.

Abuso de Notación:
Para un grafo dirigido con pesos enteros y no nulos, denotaremos $perm(G)$ a $PesoCycleCover(G)$.

PesoCyclesCovers=Perm

En general, podemos considerar el siguiente problema de conteo:

Dado un grafo dirigido G con pesos enteros y no nulos en los arcos, computar $PesoCycleCover(G)$.

Este problema es equivalente al problema de computar $PesoCycleCover(G)$ para un grafo dirigido completo G con pesos enteros, el cual a su vez, como ya vimos, es equivalente a computar el permanente de una matriz entera.

Abuso de Notación:
Para un grafo dirigido con pesos enteros y no nulos, denotaremos $perm(G)$ a $PesoCycleCover(G)$.

Demostración del Teorema de Valiant

1. Reducimos de #3SAT a Perm con entrada enteras.
2. Reducimos Perm con entradas enteras a Perm con entradas en $\{-1, 0, 1\}$.
3. Reducimos Perm con entradas en $\{-1, 0, 1\}$ a Perm con entradas $\{0, 1\}$.

Demostración del Teorema de Valiant

1. Reducimos de #3SAT a Perm con entrada enteras.
2. Reducimos Perm con entradas enteras a Perm con entradas en $\{-1, 0, 1\}$.
3. Reducimos Perm con entradas en $\{-1, 0, 1\}$ a Perm con entradas $\{0, 1\}$.

Demostración del Teorema de Valiant

1. Reducimos de #3SAT a Perm con entrada enteras.
2. Reducimos Perm con entradas enteras a Perm con entradas en $\{-1, 0, 1\}$.
3. Reducimos Perm con entradas en $\{-1, 0, 1\}$ a Perm con entradas $\{0, 1\}$.

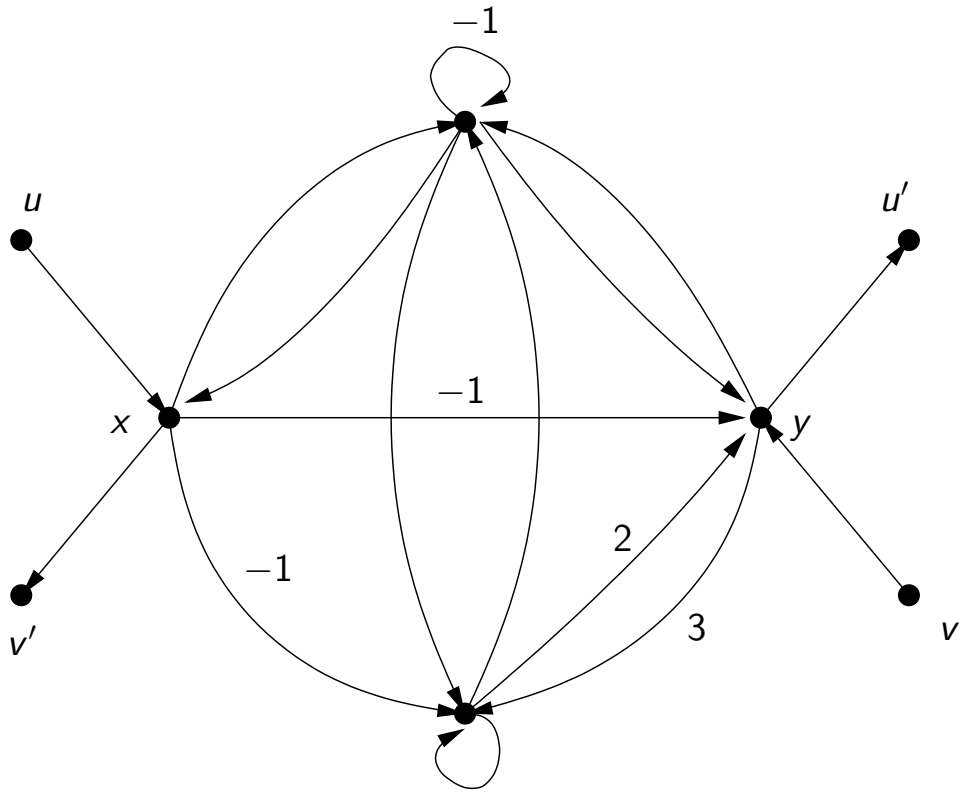
$\#3SAT \leq$ Perm con entrada enteras

Dada una fórmula proposicional φ en 3CNF con n variables y m clausulas construiremos un digrafo con pesos (no nulos) G tal que

$$perm(G) = 4^{3m} \#3SAT(\varphi)$$

Gadget XOR

Sea H un digrafo con peso 1 en todos sus arcos y dos arcos no adyacentes (u, u') y (v, v') en H. Construimos H' en donde los dos arcos son reemplazados por el gadget XOR:



Notación: Los arcos dibujados sin pesos tienen peso 1.

Propiedades del gadget XOR

Todo cycle cover en H' debe caer en algunos de los siguientes casos:

1. Usa el arco (u, x) e (y, u'), y NO usa los arcos (v, y) ni (x, v'), o bien, usa el arco (v, y) y (x, v'), y NO usa los arcos (u, x) ni (y, u'). En estos dos casos decimos que el cycle cover respeta el XOR.
2. Usar los 4 arcos, no usar ninguno, usar (u, x) y (x, v'), y NO usar (v, y) ni (y, u'), o bien, usar (v, y) y (y, u'), y NO usar (u, x) ni (x, v'). En estos 4 casos decimos que el cycle cover no respeta el XOR.

Todos los otros casos no pueden pasar.

Propiedades del gadget XOR

Sea H y H' como antes (recordar que H tiene pesos 1).Tenemos los siguiente:

Proposición

Sea $\mathcal{C}^+$ el conjunto de los cycle covers en H' que respetan el XOR y $\mathcal{C}^-$ los que no lo respetan. Entonces

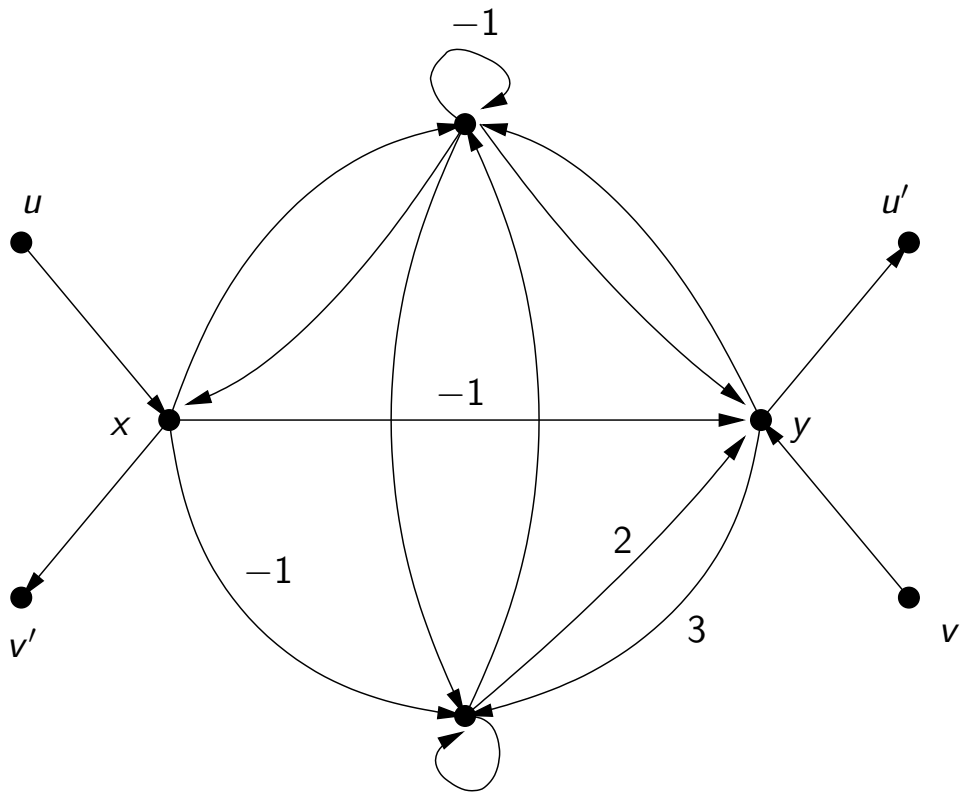
$$\sum_{C \in \mathcal{C}^-} w(C) = 0$$

$$\sum_{C \in \mathcal{C}^+} w(C) = 4M$$

donde M es la cantidad de cycle covers en H que ocupan (u, u') o (v, v'), pero no ambos.

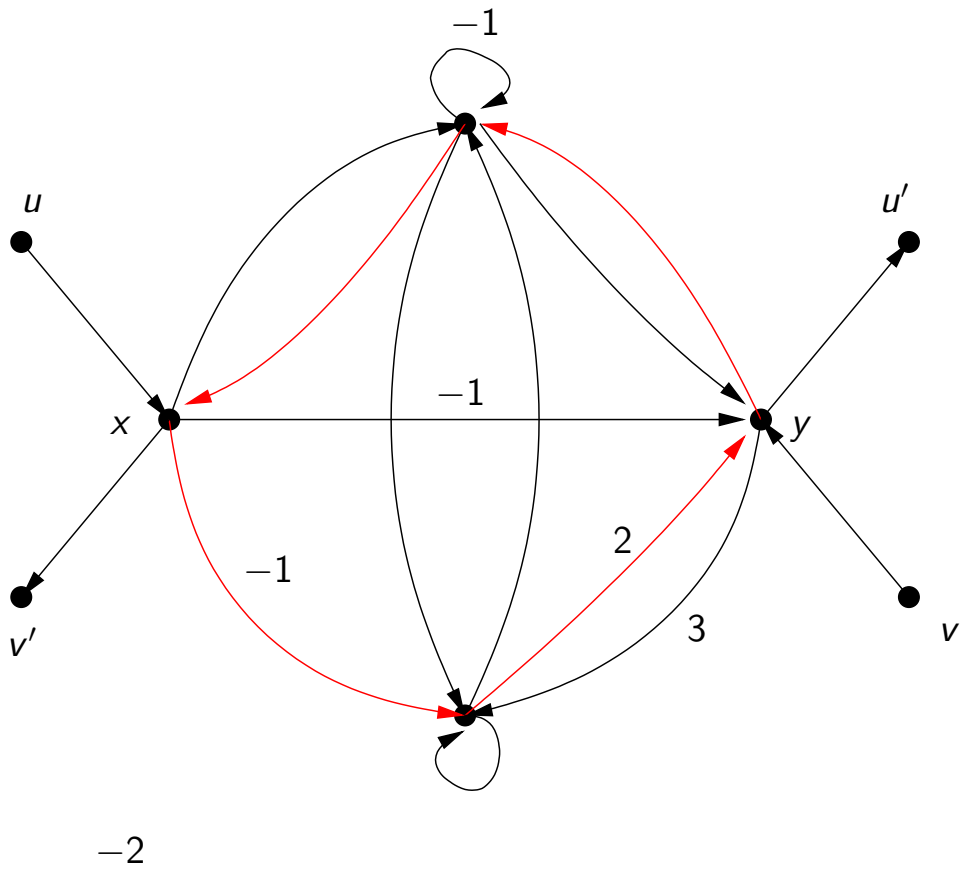
Propiedades del gadget XOR

Ejemplo (no respeta el XOR):



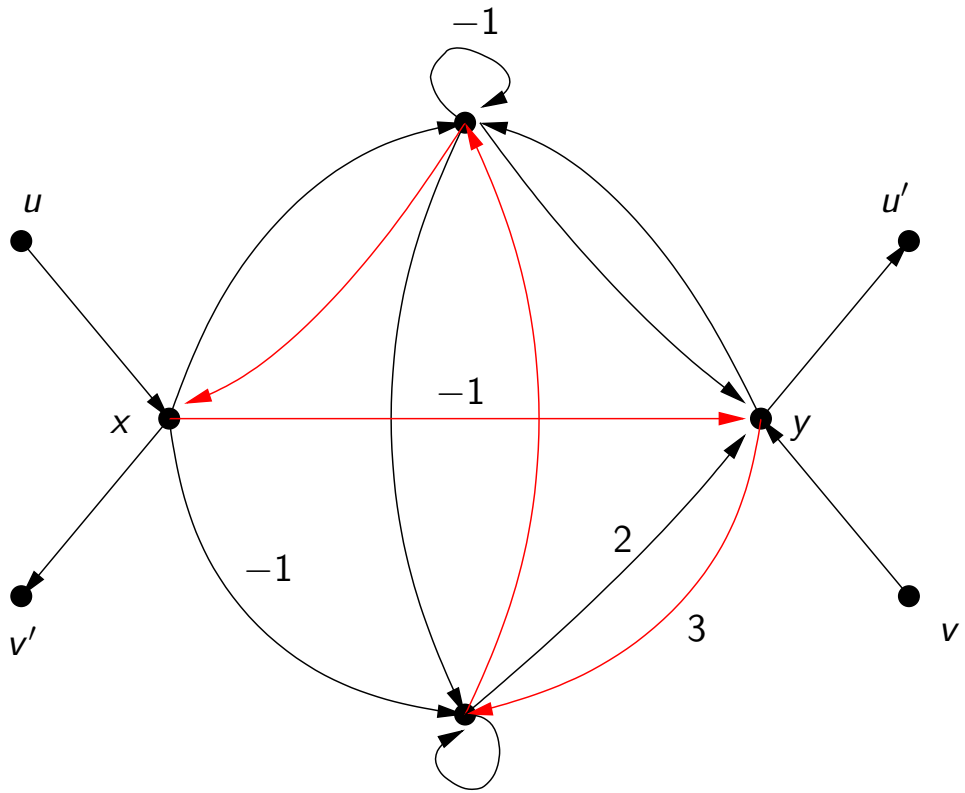
Propiedades del gadget XOR

Ejemplo (no respeta el XOR):



Propiedades del gadget XOR

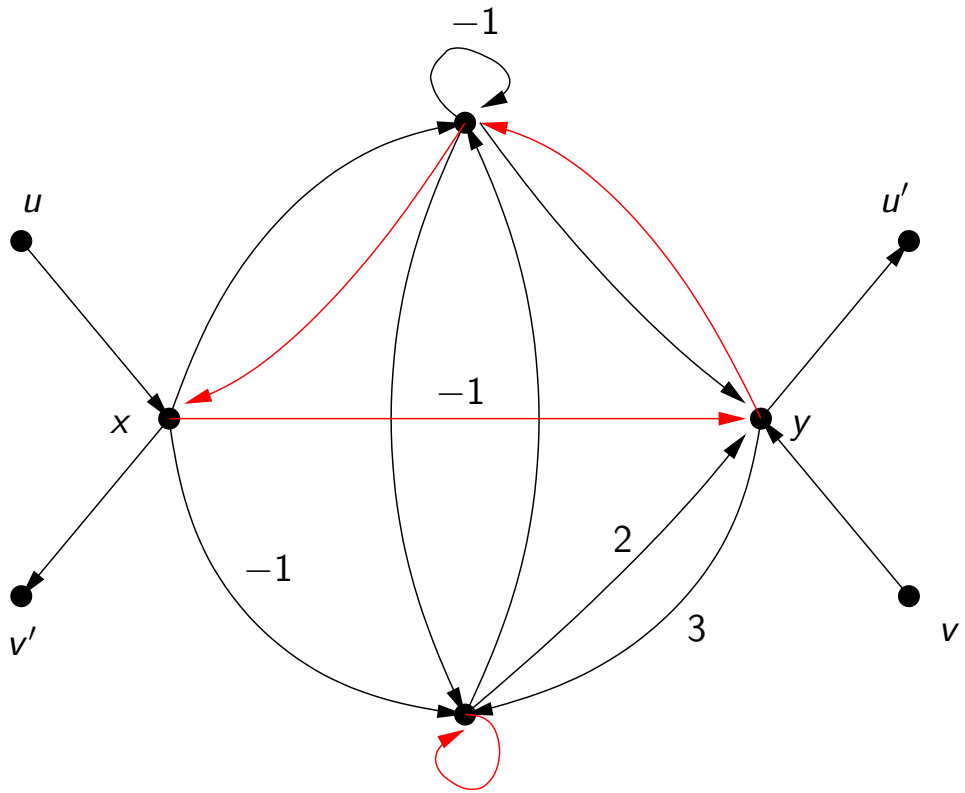
Ejemplo (no respeta el XOR):



$-2 \quad +(-3)$

Propiedades del gadget XOR

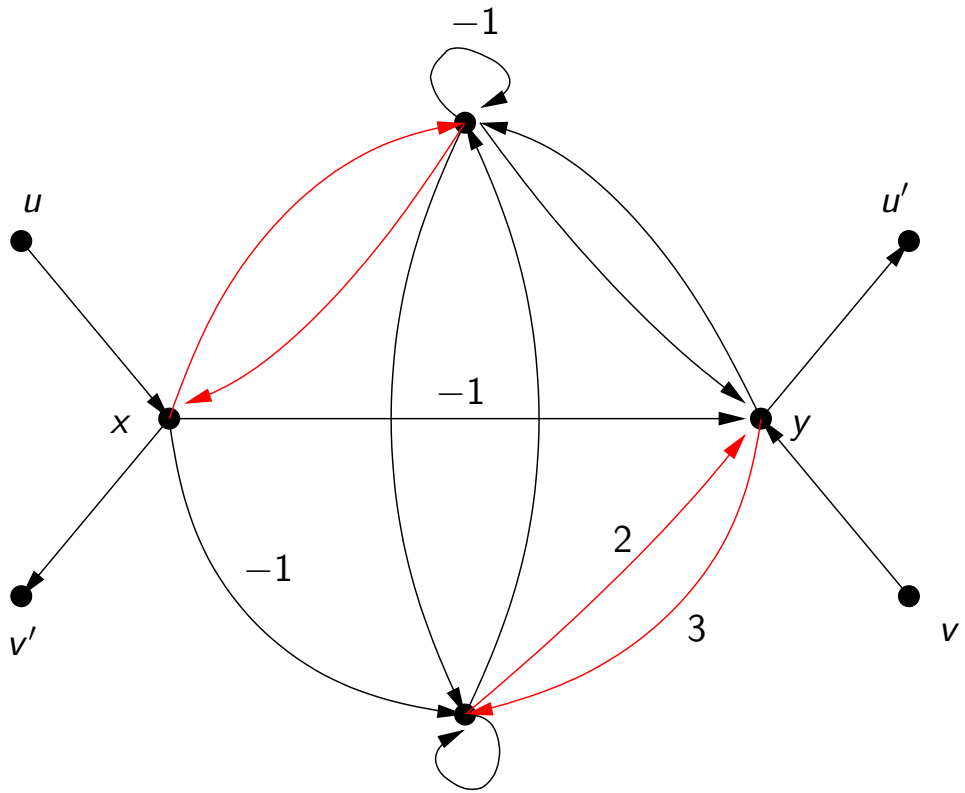
Ejemplo (no respeta el XOR):



$-2 \quad +(-3) +(-1)$

Propiedades del gadget XOR

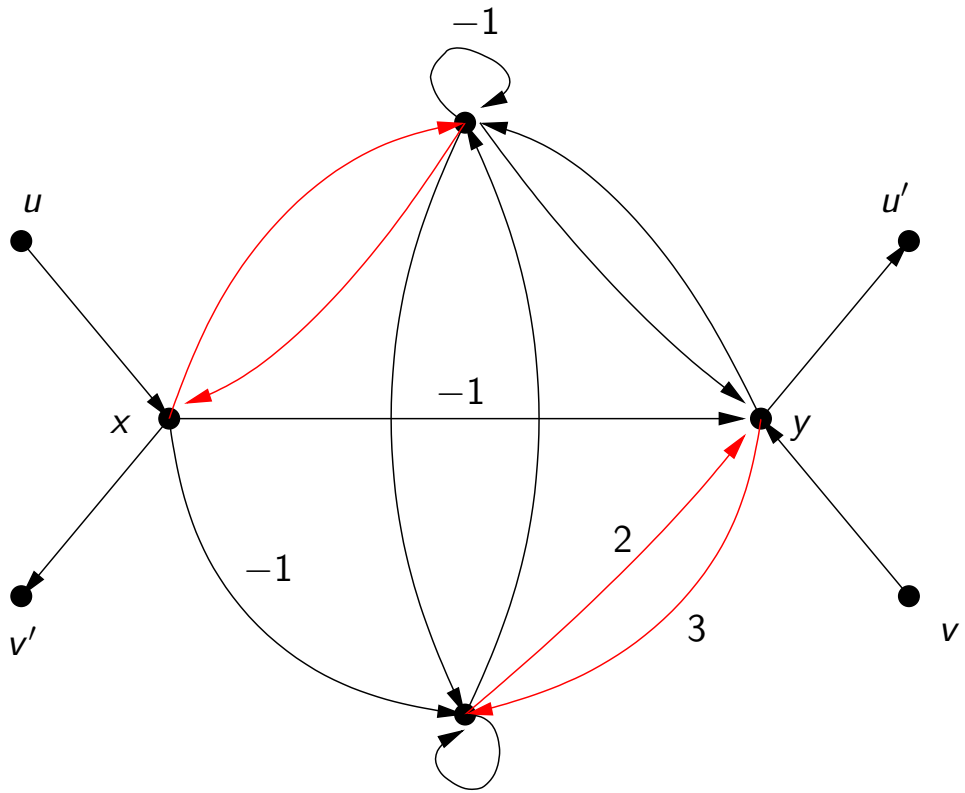
Ejemplo (no respeta el XOR):



$$-2 \;\; +(-3) +(-1) +6$$

Propiedades del gadget XOR

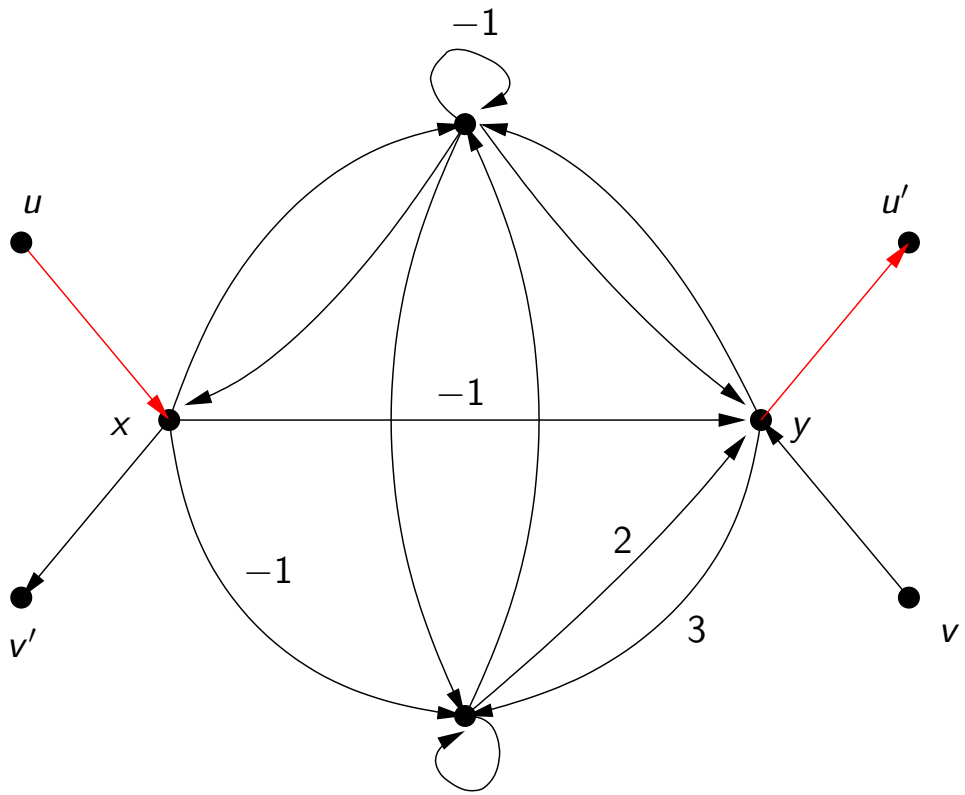
Ejemplo (no respeta el XOR):



$$-2 \ +(-3)+(-1)+6 \ = 0$$

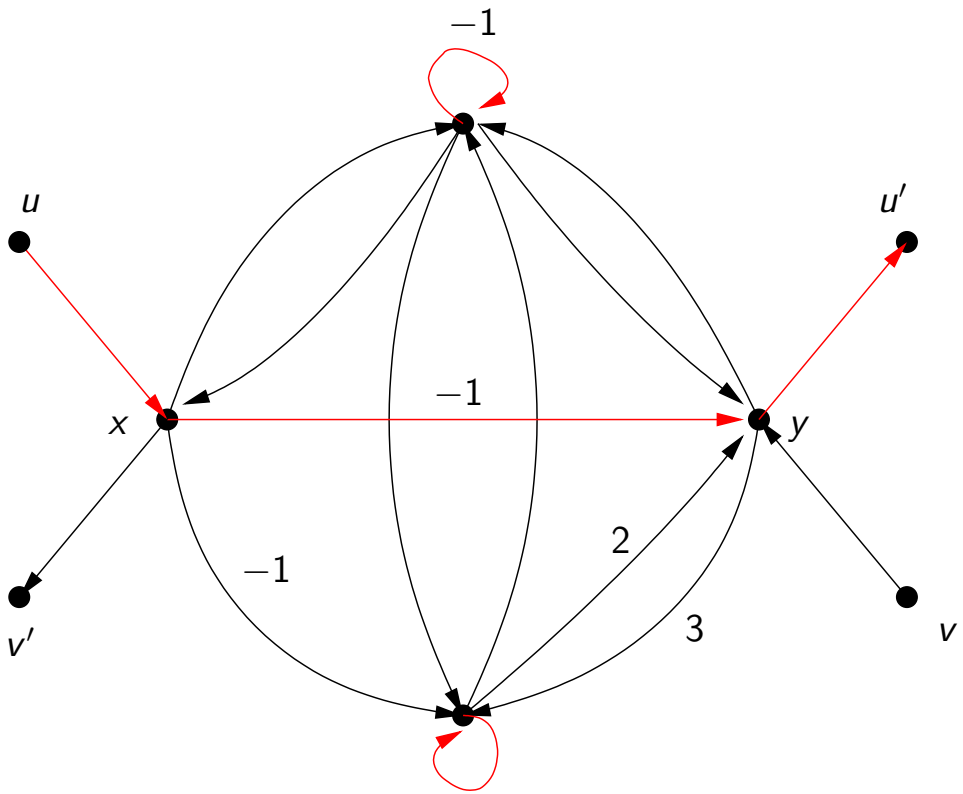
Propiedades del gadget XOR

Ejemplo (respeta el XOR):



Propiedades del gadget XOR

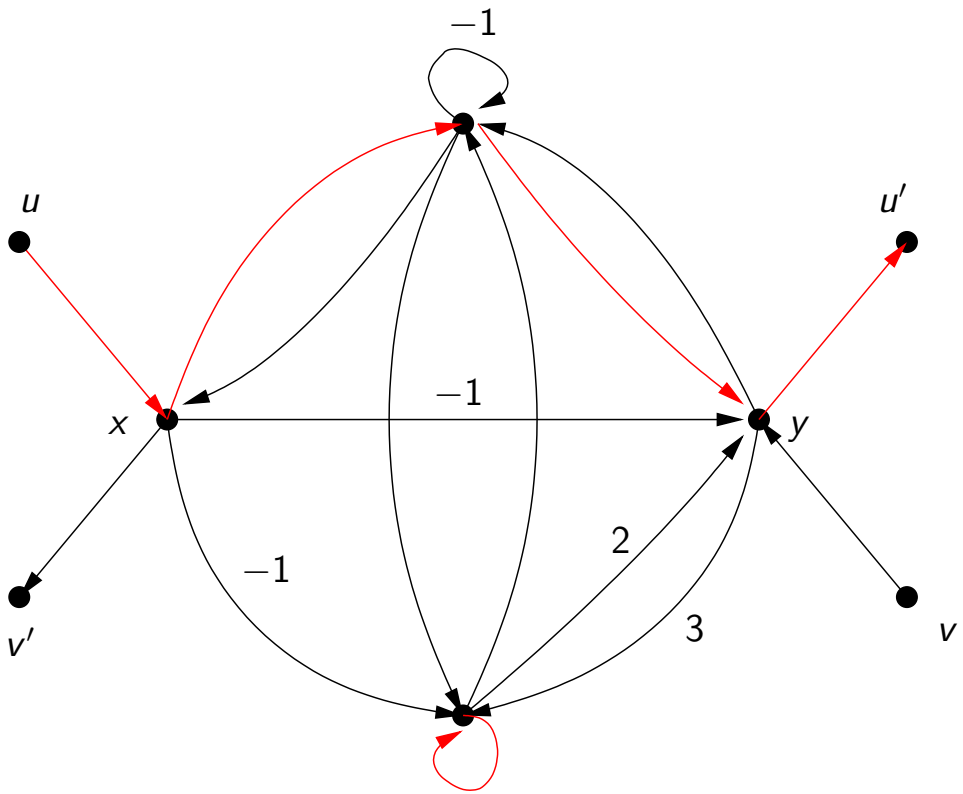
Ejemplo (respeta el XOR):



1

Propiedades del gadget XOR

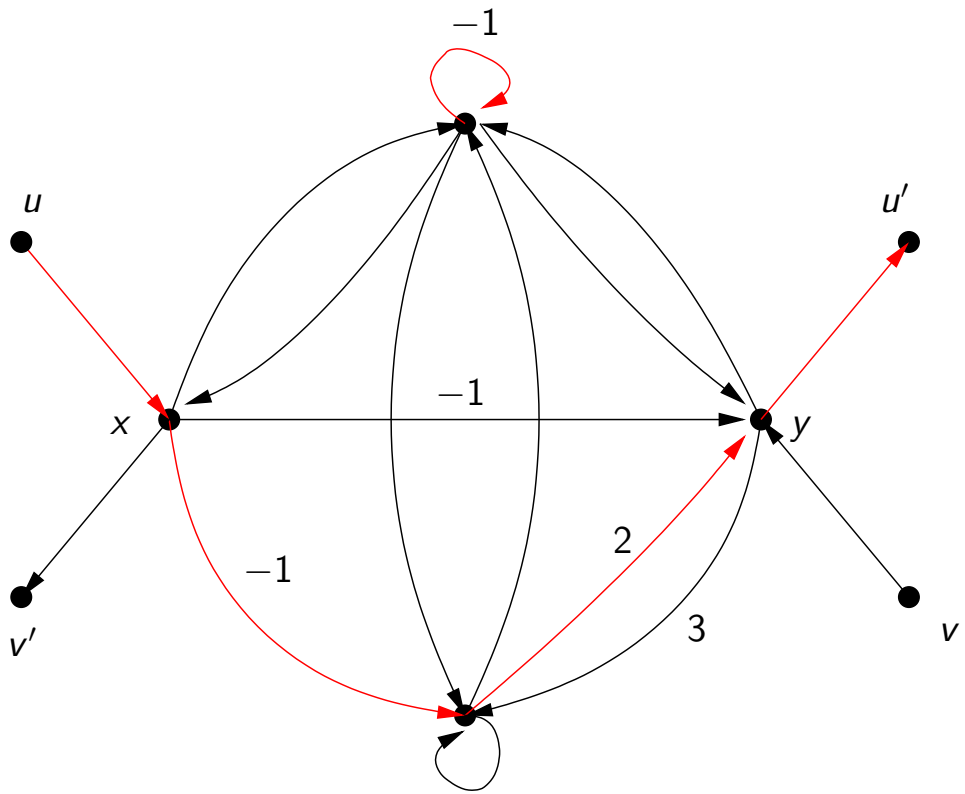
Ejemplo (respeta el XOR):



1 +1

Propiedades del gadget XOR

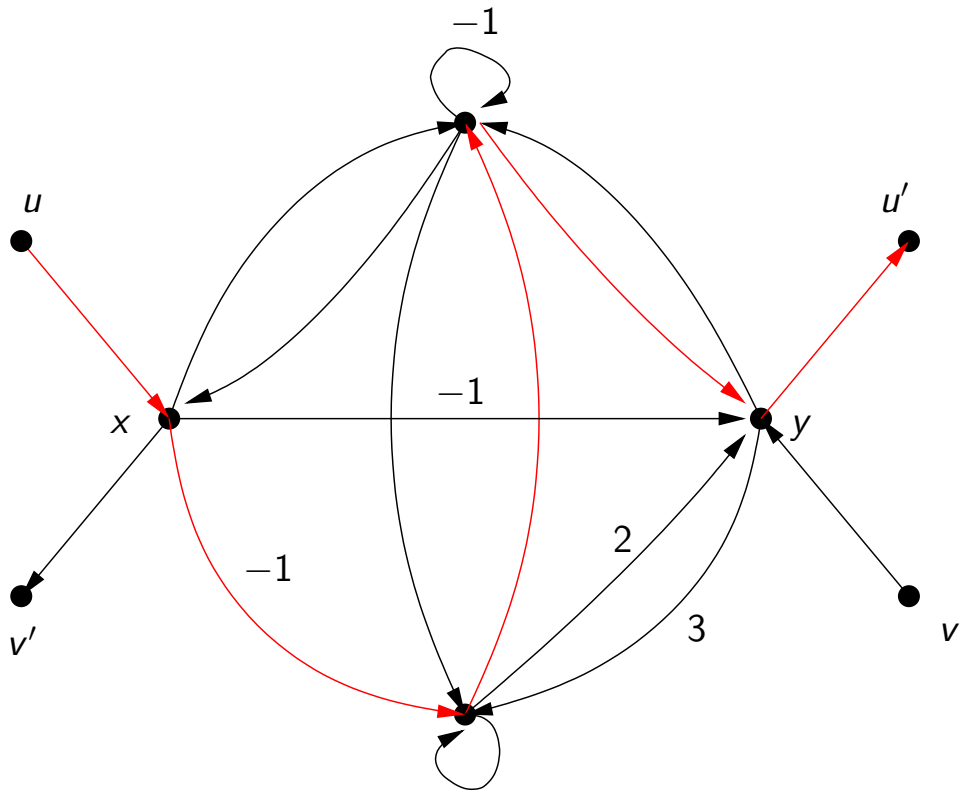
Ejemplo (respeta el XOR):



1 +1 +2

Propiedades del gadget XOR

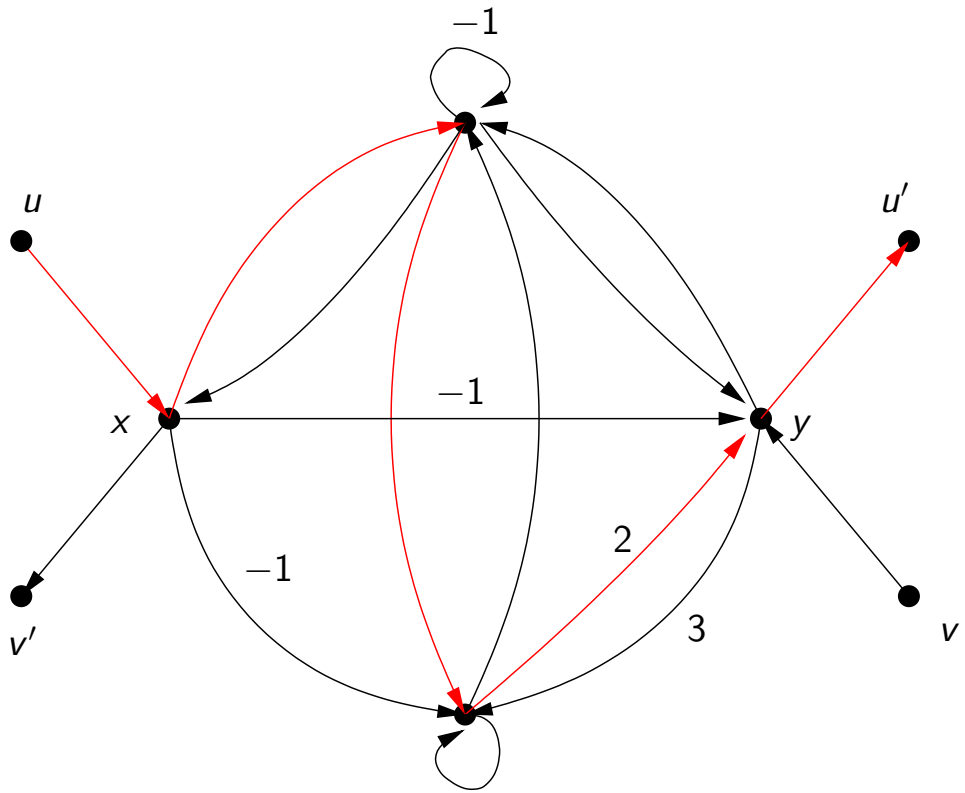
Ejemplo (respeta el XOR):



$1 \quad +1 \quad +2 \quad +(-1)$

Propiedades del gadget XOR

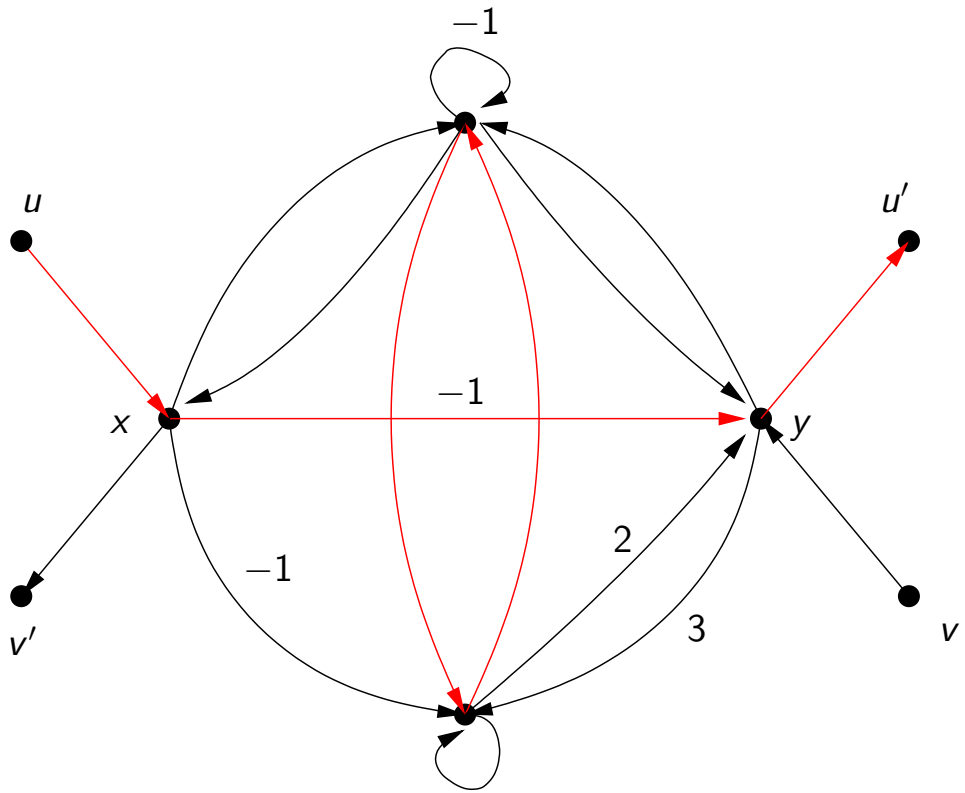
Ejemplo (respeta el XOR):



$1 \quad +1 \quad +2 \quad +(-1) \quad +2$

Propiedades del gadget XOR

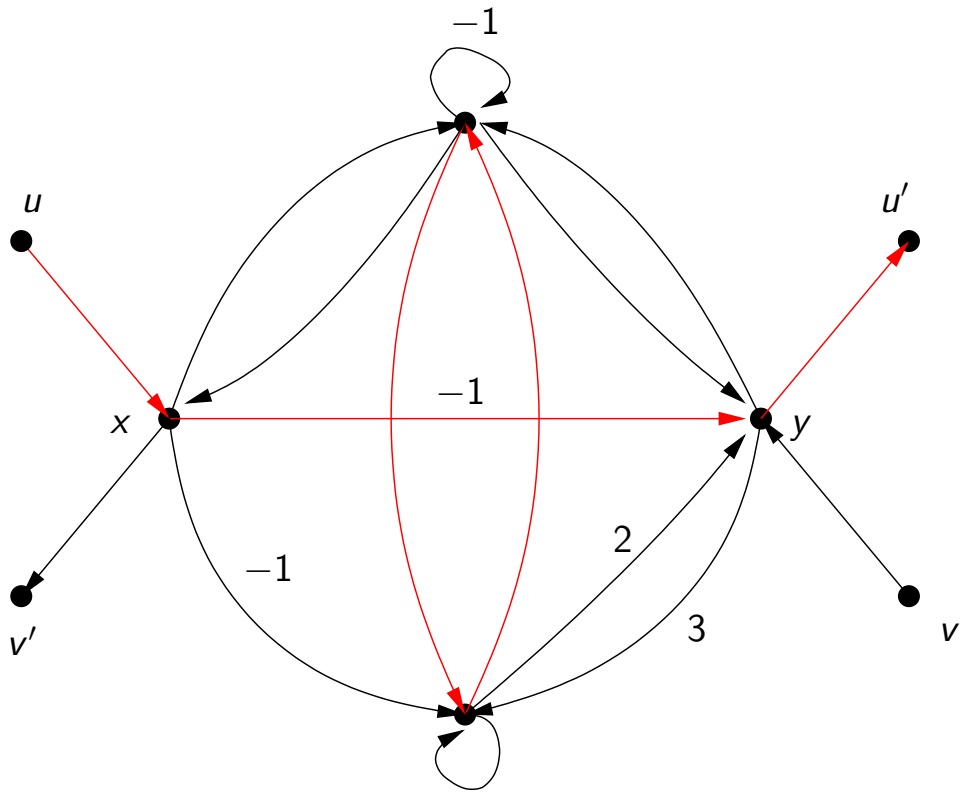
Ejemplo (respeta el XOR):



$1 \quad +1 \quad +2 \quad +(-1) \, +2 \quad +(-1)$

Propiedades del gadget XOR

Ejemplo (respeta el XOR):



$$1 \quad +1 \quad +2 \quad +(-1) \quad +2 \quad +(-1) = 4$$

Propiedades del gadget XOR

Podemos generalizar lo anterior. Sea H con pesos 1 y $2k$ arcos no adyacentes par a par (u_i, u_i') y (v_i, v_i') para $1 \leq i \leq k$. Sea H' el digrafo resultante de hacer el XOR entre (u_i, u_i') y (v_i, v_i') para todo $1 \leq i \leq k$.

Proposición

Sea $\mathcal{C}^+$ el conjunto de los cycle covers en H' que respetan TODOS los XOR y $\mathcal{C}^-$ los que no lo respetan ALGUN XOR. Entonces

$$\sum_{C \in \mathcal{C}^-} w(C) = 0$$

$$\sum_{C \in \mathcal{C}^+} w(C) = 4^k M$$

donde M es la cantidad de cycle covers en H tal que para todo $1 \leq i \leq k$, ocupan (u_i, u_i') o (v_i, v_i'), pero no ambos.

Abusando notación decimos que M es la cantidad de cycle covers en H que respetan todos los XORs.

Propiedades del gadget XOR

Podemos generalizar lo anterior. Sea H con pesos 1 y $2k$ arcos no adyacentes par a par (u_i, u_i') y (v_i, v_i') para $1 \leq i \leq k$. Sea H' el digrafo resultante de hacer el XOR entre (u_i, u_i') y (v_i, v_i') para todo $1 \leq i \leq k$.

Proposición

Sea $\mathcal{C}^+$ el conjunto de los cycle covers en H' que respetan TODOS los XOR y $\mathcal{C}^-$ los que no lo respetan ALGUN XOR. Entonces

$$\sum_{C \in \mathcal{C}^-} w(C) = 0$$

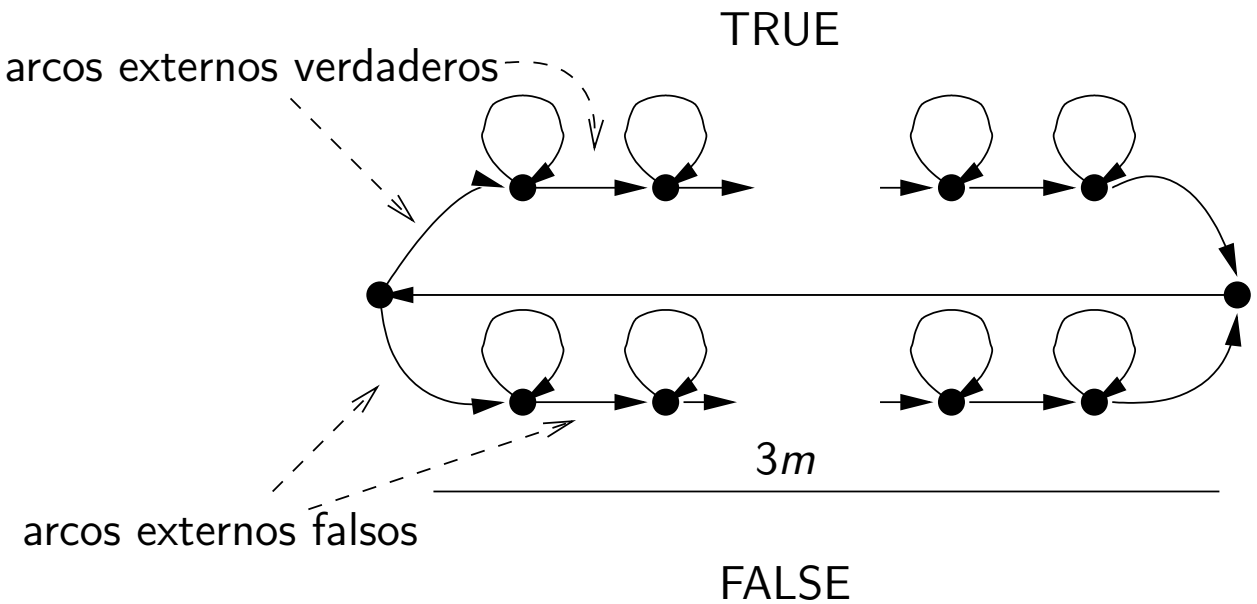
$$\sum_{C \in \mathcal{C}^+} w(C) = 4^k M$$

donde M es la cantidad de cycle covers en H tal que para todo $1 \leq i \leq k$, ocupan (u_i, u_i') o (v_i, v_i'), pero no ambos.

Abusando notación decimos que M es la cantidad de cycle covers en H que respetan todos los XORs.

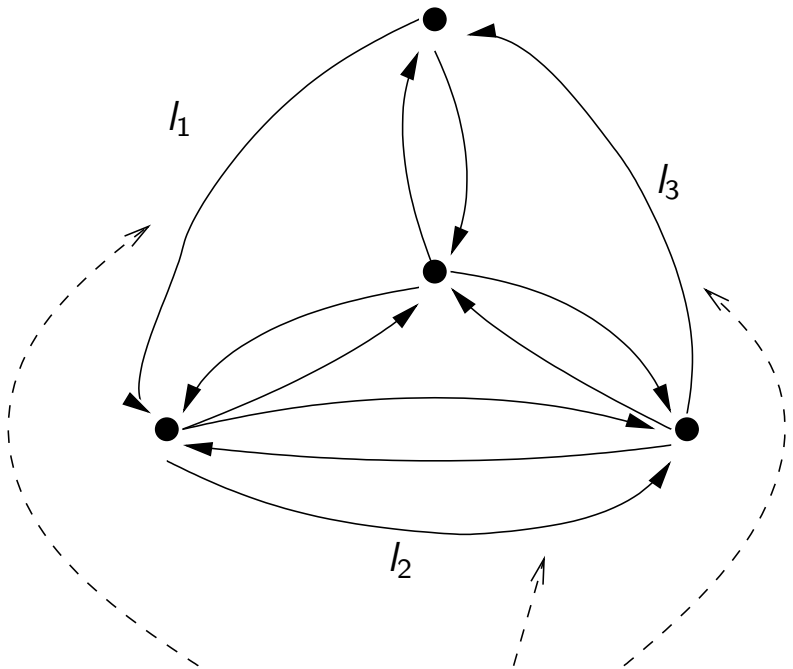
Construcción de G

Tenemos una formula φ en 3CNF con variables $x_1, ..., x_n$ y clausulas $C_1, ... C_m$.
Por cada variable construimos el siguiente gadget de variable:



Construcción de G

Por cada clausula $C = (l_1 \vee l_2 \vee l_3)$ construimos el siguiente gadget de clausula:



Construcción de G

Llamamos G' al grafo construido por los gadgets de variables y clausula. Finalmente, construimos G a partir de G', conectando con un gadget XOR cada arco externo de cada gadget de clausula con algun arco externo disponible en el gadget de variable asociado:

- Si el arco externo en la clausula esta asociado a un literal positivo x_i, entonces conectamos este arco, via un XOR, con algun arco externo verdadero en el gadget de variable asociado a x_i.
- Si esta asociado a un literal negativo $\bar{x}_j$, entonces lo conectamos, via un XOR, con algun arco externo falso en el gadget de variable asociado a x_j.

Construcción de G

Llamamos G' al grafo construido por los gadgets de variables y clausula. Finalmente, construimos G a partir de G', conectando con un gadget XOR cada arco externo de cada gadget de clausula con algun arco externo disponible en el gadget de variable asociado:

- Si el arco externo en la clausula esta asociado a un literal positivo x_i, entonces conectamos este arco, via un XOR, con algun arco externo verdadero en el gadget de variable asociado a x_i.
- Si esta asociado a un literal negativo $\bar{x}_j$, entonces lo conectamos, via un XOR, con algun arco externo falso en el gadget de variable asociado a x_j.

Construcción de G

Como tenemos precisamente $3m$ XOR's tenemos que:

$$perm(G) = 4^{3m} M$$

donde M es la cantidad de cycle covers en G' que respetan todos los XORs.

Propiedades de los gadgets

- Cada gadget de variable tiene 2 posibles cycle covers (TRUE o FALSE).
- No existe un cycle cover en un gadget de clausula que use los 3 arcos externos.
- Para cualquier subconjunto propio de arcos externos en un gadget de clausula, existe un único cycle cover que ocupa precisamente esos arcos externos.

Propiedades de los gadgets

- Cada gadget de variable tiene 2 posibles cycle covers (TRUE o FALSE).
- No existe un cycle cover en un gadget de clausula que use los 3 arcos externos.
- Para cualquier subconjunto propio de arcos externos en un gadget de clausula, existe un único cycle cover que ocupa precisamente esos arcos externos.

Propiedades de los gadgets

- Cada gadget de variable tiene 2 posibles cycle covers (TRUE o FALSE).
- No existe un cycle cover en un gadget de clausula que use los 3 arcos externos.
- Para cualquier subconjunto propio de arcos externos en un gadget de clausula, existe un único cycle cover que ocupa precisamente esos arcos externos.

#3SAT $\leq$ Perm con entrada enteras

Hay una biyeccion de las asignaciones que hacen verdaderas a φ y los cycles cover en G' que respetan los XORs.

Sea $\bar{a}$ una asignacion con $\varphi(\bar{a}) = 1$. Construimos el cycle cover tal que en el gadget de variable x_i escogemos los arcos externos VERDADEROS ssi $a_i = 1$. Para respetar los XORs, si en el gadget de x_i escogimos los arcos VERDADEROS, debemos dejar libre todos los arcos externos de gadget de clausulas asociados a x_i y debes ocupar los asociados $\bar{x}_i$. Si escogimos los arcos FALSOS, debemos debemos dejar libres los arcos externos asociados a $\bar{x}_i$ y ocupar los asociados a x_i.

#3SAT $\leq$ Perm con entrada enteras

Luego debemos dejar libre un arco externo (con literal asociado l) en un gadget de clausula ssi el literal es cierto en la asignación.

Como $\bar{a}$ hace cierto a φ, para construir nuestro cycle cover, por cada gadget de clausula debemos dejar al menos un arco externo libre, lo cual se puede hacer (y de manera única).

Este mapeo es sobreyectivo ya que TODOS los cycles cover que respetan los XORs son generados de esta manera.

#3SAT $\leq$ Perm con entrada enteras

Lo anterior nos dice que

$$M = \#SAT(\varphi)$$

por lo tanto,

$$perm(G) = 4^{3m} \#SAT(\varphi)$$

$\#3SAT \leq$ Perm con entrada enteras

Lo anterior nos dice que

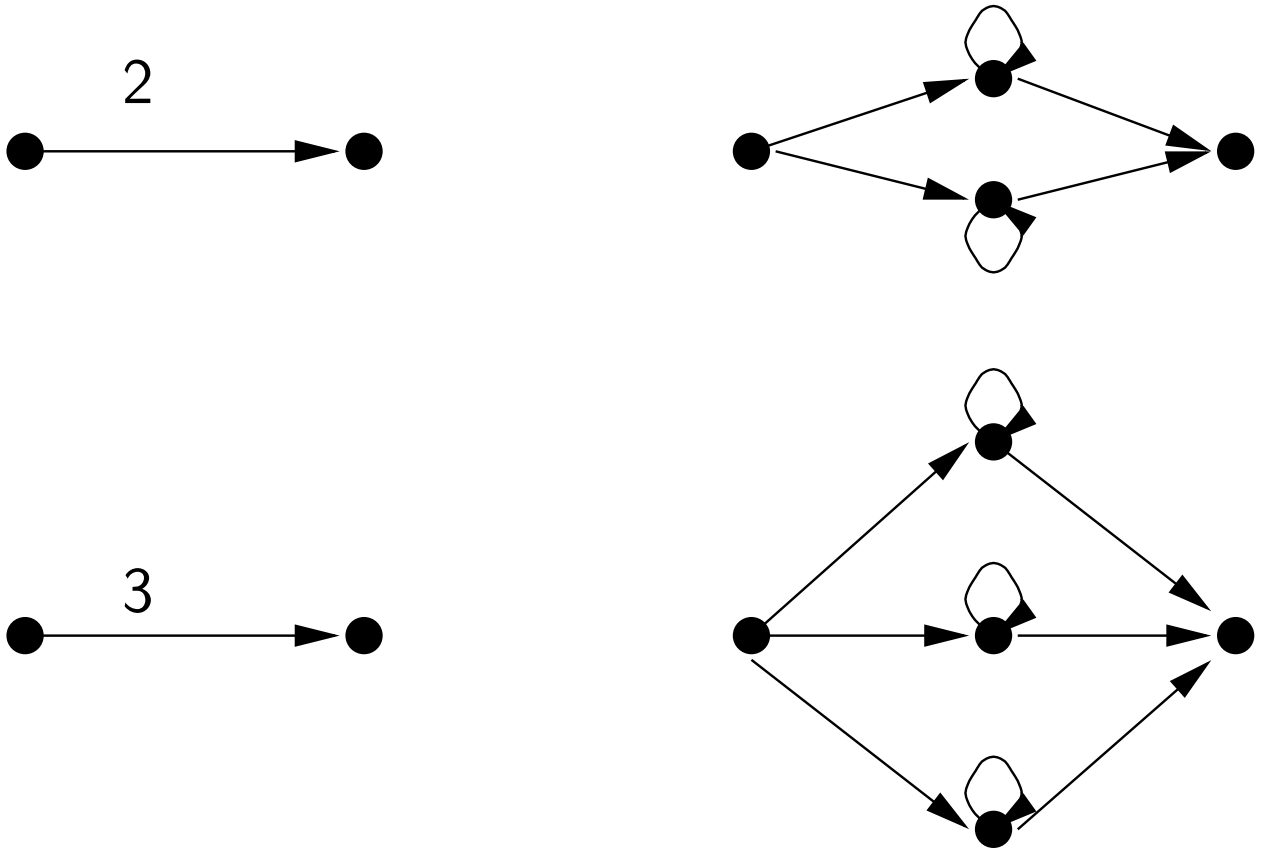
$$M = \#SAT(\varphi)$$

por lo tanto,

$$perm(G) = 4^{3m}\#SAT(\varphi)$$

Perm entradas enteras $\leq$ Perm entradas $\{-1, 0, 1\}$

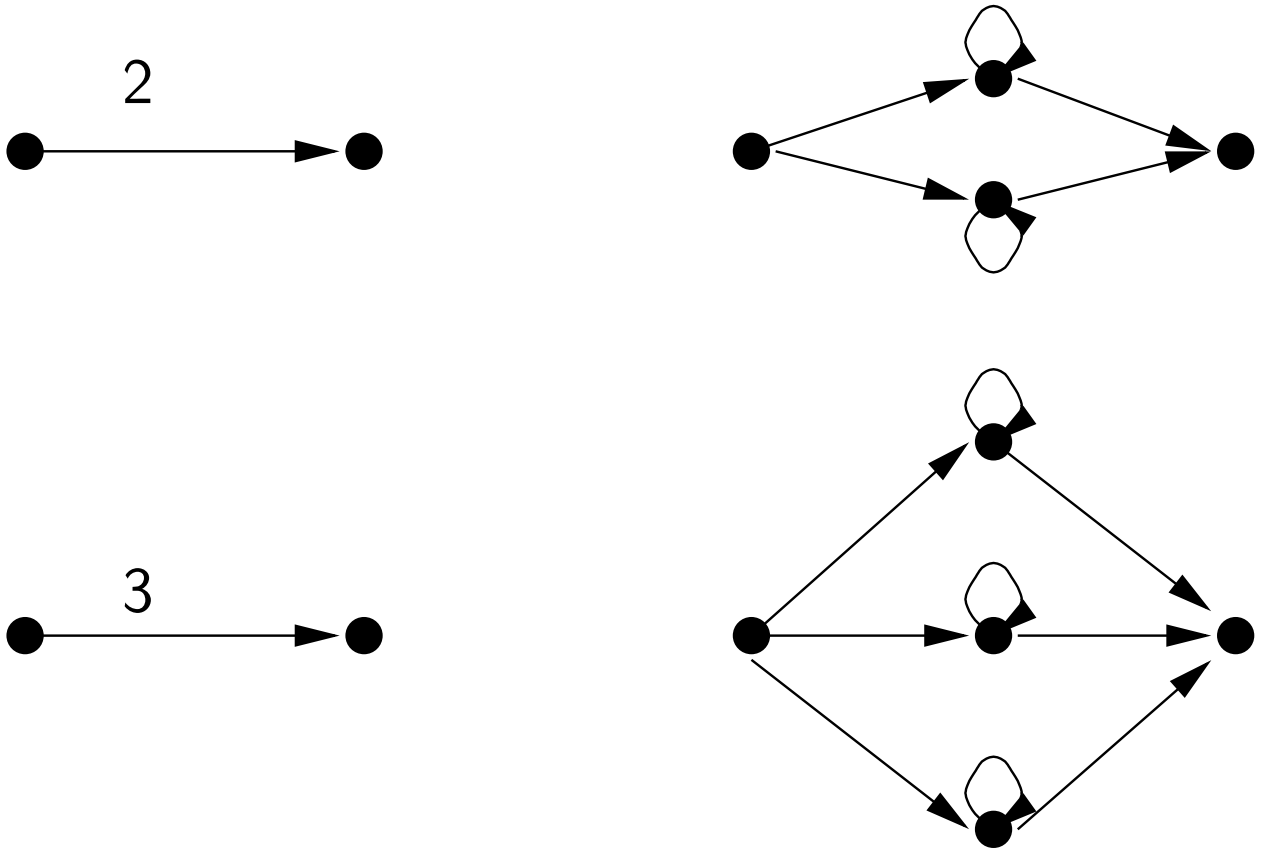
Los siguientes reemplazos no alteran el permantente del digrafo G:



Luego podemos transformar G en un digrafo G'' tal que $perm(G) = perm(G'')$.

Perm entradas enteras $\leq$ Perm entradas $\{-1, 0, 1\}$

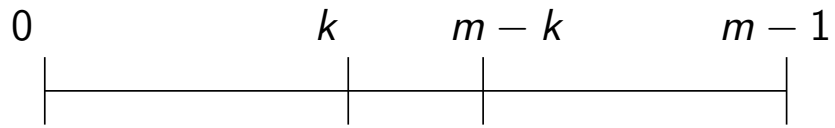
Los siguientes reemplazos no alteran el permantente del digrafo G:



Luego podemos transformar G en un digrafo G'' tal que $perm(G) = perm(G'')$.

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

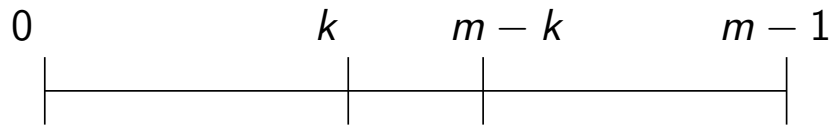
Si $x \in [-k, k]$ y $m > 2k$, entonces si conocemos el valor de x mod m, podemos recuperar el valor de x.



- Si x mod m cae en $[0, ..., k]$ entonces $x = x$ mod m
- Si x mod m cae en $[m - k, ..., m - 1]$ entonces $x = (x \text{ mod } m) - m$

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Si $x \in [-k, k]$ y $m > 2k$, entonces si conocemos el valor de x mod m, podemos recuperar el valor de x.



- Si x mod m cae en $[0, ..., k]$ entonces $x = x$ mod m
- Si x mod m cae en $[m - k, ..., m - 1]$ entonces $x = (x \text{ mod } m) - m$

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Dado G con pesos en $\{-1, 0, 1\}$ construiremos un digrafo G' tal que si conocemos $perm(G')$, podemos computar rapidamente $perm(G)$. Para construir G' hacemos lo siguente:

- Definimos $m = n \log n$, donde n es la cantidad de nodos en G.
- Reemplazamos los pesos -1 por pesos 2^m.
- Simulamos los pesos 2^m con pesos 1 (como?).

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Dado G con pesos en $\{-1, 0, 1\}$ construiremos un digrafo G' tal que si conocemos $perm(G')$, podemos computar rapidamente $perm(G)$. Para construir G' hacemos lo siguente:

- Definimos $m = n \log n$, donde n es la cantidad de nodos en G.
- Reemplazamos los pesos -1 por pesos 2^m.
- Simulamos los pesos 2^m con pesos 1 (como?).

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Dado G con pesos en $\{-1, 0, 1\}$ construiremos un digrafo G' tal que si conocemos $perm(G')$, podemos computar rapidamente $perm(G)$. Para construir G' hacemos lo siguente:

- Definimos $m = n \log n$, donde n es la cantidad de nodos en G.
- Reemplazamos los pesos -1 por pesos 2^m.
- Simulamos los pesos 2^m con pesos 1 (como?).

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Como $-1 \equiv 2^m (\text{mod } 2^m + 1)$ tenemos que $perm(G) \equiv perm(G')(\text{mod } 2^m + 1)$.

Como $perm(G) \in [-n!, n!]$ y $2^m + 1 = n^n + 1 > 2n!$ para $n > 1$, tenemos que si conocemos $perm(G)$ mod $(2^m + 1)$ podemos computar rapidamente $perm(G)$.

Efectivamente, conocemos $perm(G)$ mod $(2^m + 1) = perm(G')$ mod $2^m + 1$.

Perm entradas $\{-1, 0, 1\} \leq$ Perm entradas $\{0, 1\}$

Como $-1 \equiv 2^m (\text{mod } 2^m + 1)$ tenemos que $perm(G) \equiv perm(G')(\text{mod } 2^m + 1)$.

Como $perm(G) \in [-n!, n!]$ y $2^m + 1 = n^n + 1 > 2n!$ para $n > 1$, tenemos que si conocemos $perm(G)$ mod $(2^m + 1)$ podemos computar rapidamente $perm(G)$.

Efectivamente, conocemos $perm(G)$ mod $(2^m + 1) = perm(G')$ mod $2^m + 1$.

Otros resultados

Otro resultado importante en complejidad de conteo.

Teorema (Toda's Theorem 91')

$PH \subseteq P^{\#P}$

Relacionó por primera vez la capacidad de alternar con la capacidad de contar.

Otros resultados

Otro resultado importante en complejidad de conteo.

Teorema (Toda's Theorem 91')

$PH \subseteq P^{\#P}$

Relacionó por primera vez la capacidad de alternar con la capacidad de contar.

Más alla de #P

Hay problemas de conteo que no son capturados por #P.

Dado un grafo G, cuantos subgrafos 3 coloreables tiene G?

Dada una consulta conjuntiva $Q(\bar{x})$ y una base de datos D, cuantas tuplas tiene la respuesta de Q en D?

Existen definiciones de jerarquias de problemas de conteo.

Más alla de #P

Hay problemas de conteo que no son capturados por #P.

Dado un grafo G, cuantos subgrafos 3 coloreables tiene G?

Dada una consulta conjuntiva $Q(\bar{x})$ y una base de datos D, cuantas tuplas tiene la respuesta de Q en D?

Existen definiciones de jerarquias de problemas de conteo.

Counting Complexity

Miguel Romero

17 de Abril del 2012