Algoritmos de Aproximación

M. Andrea Rodríguez-Tastets
Ayudante: Erick Elejalde

Universidad de Concepción,Chile
www.inf.udec.cl\ ∼andrea
andrea@udec.cl

I Semestre - 2013

Introducción

- La mayoría de los algoritmos que se han estudio hasta ahora son de tiempo polinomial $O(n^k)$ para una entrada de tamaño n y una constante k. Estos algoritmos resuelven problemas que se llaman problemas de clase P.
- Generalmente se considera que problemas que pueden resolverse en tiempo polinomial son tratables y que problemas que requieren un tiempo superpolinomial son intratables.
- NP es una clase de problemas que son verificables en tiempo polinomial, donde $P \subseteq NP$. Es una pregunta aún abierta si P es o no un subconjunto propio de NP.
- NP-Complete es una clase de problemas que es al menos tan *hard* como cualquier problema NP.
- No existen pruebas conclusivas de que problemas NP-Complete puedan ser resueltos por algoritmos polinomiales. Sin embargo, el que un problema sea NP-Complete da buena evidencia de su intratabilidad.

P versus NP-Complete

En los siguientes casos se muestran problemas que si bien es cierto parecen similares, uno es en P y el otro en NP-Complete.

P	NP
Shortest simple path: Dado un grafo con arcos con pesos $G = (V, E)$, uno puede encontrar el camino más corto con $O((\|V\| + \|E\|)$	**Longest simple path**: Encontrar el camino más largo
Euler tour: es un ciclo en un grafo dirigido conectado $G = (V, E)$ que atraviesa cada arco una sola vez, aunque los vértices se visiten más de una vez $(O(\|E\|))$	**Hamiltonian cycle**: de un grafo dirigido es un ciclo simple que contiene cada vértice de V

La prueba de cuándo un problema es NP-Complete está fuera del alcance de este curso. Sin embargo, abordaremos los algoritmos de aproximación en la sección que sigue para abordar este tipo de problemas.

Introducción (cont.)

Existen muchos problemas de optimización que siendo NP-Complete se necesitan resolver. Entonces, no se puede renunciar a tratar de dar alguna solución. Las alternativas para lidear con estos problema son :

- Uso de búsqueda de fuerza bruta. Incluso para computadores paralelos rápidos, este enfoque es viable solo para instancias pequeñas del problema.
- Heurísticas: Una heurística es una estrategia para producir un solución válida, pero que no garantiza qué tan cerca es del óptimo. Esto es viable si todo lo demás falla y si no es un problema tan grande el hecho de no encontrar un óptimo.
- Métodos de búsqueda general: Estos métodos han sido desarrollados en el área de IA y investigación de operaciones. Ejemplos de estos métodos son algoritmos genéticos, simulated annealing, A*-search, y branch-and-bound. El rendimiento de estos métodos varía de un problema a otro.
- Algoritmos de aproximación: Es un algoritmo que corre en tiempo polinomial (idealmente) y que produce una solución que está dentro de un factor garantizado de la solución óptima.

Razón de Rendimiento para Algoritmos de Aproximación

- La mayoría de los problemas NP-Complete han sido establecidos como problemas de decisión por razones teóricas; sin embargo, encubren en muchos casos problemas de optimización. Por ejemplo, el Vertex Cover (el subconjunto de vértices tal que cada arco del grafo es incidente a al menos uno de los vértices en el set) es un problema de optimización que encuentra una cobertura de vértices de tamaño menor y el problema de optimización clique es el problema (encontrar el subconjunto de vértices de un grafo que están completamente conectados) de encontrar el clique de tamaño máximo. En algunos casos se maximiza y en otros se minimiza.
- ¿Cómo se mide qué tan bueno es un algoritmo de aproximación? Se define una razón de aproximación de un algoritmo de aproximación como sigue.
- Se asumen: (1) cada solución potencial tiene un costo positivo asociado y (2) el problema puede ser de minimización o maximización del costo.

Razón de Rendimiento para Algoritmos de Aproximación (cont.)

- Dada una instance I de un problema, sea $C(I)$ el costo de la solución producida por el algoritmo de aproximación y sea $C^*(I)$ el costo de la solución óptima. Se asumo que los costos son estrictamente positivos.
- Para problemas de minimización, se quiere que $C(I)/C^*(I)$ sea chico, y para problemas de maximización se quiere que $C^*(I)/C(I)$ sea chico.
- Se dice que para un tamaño de entrada n, el algoritmo de aproximación alcanza una razón de aproximación $\rho(n)$, si para todo I, $|I| = n$ tenemos

$$max\left(\frac{C(I)}{C^*(I)}, \frac{C^*(I)}{C(I)}\right) \leq \rho(n)$$

- Si se cumple la condición anterior, se dice que el algoritmo es un algoritmo de $\rho(n)-$approximación.

Razón de Rendimiento para Algoritmos de Aproximación (cont.)

- En problemas de maximización: $0 \leq C \leq C^*$ y $\rho(n) = C^*/C$.
- En problemas de minimización: $0 \leq C^* \leq C$ y $\rho(n) = C/C^*$.
- $\rho(n)$ es siempre positivo.
- Un algoritmo 1-aproximación es una solución óptima.
- El objetivo es encontrar un algoritmo en tiempo polinomial con la menor constante de razón de rendimiento.

Esquema de Aproximación

- Un esquema de aproximación en un algoritmo de aproximación que toma $\epsilon > 0$ como entrada tal que para cualquier $\epsilon > 0$ dado, el esquema es un algoritmo $(1 + \epsilon)-$aproximación.
- Se llaman un esquema de aproximación de tiempo polinomial a un algoritmo que corre en tiempo polinomial de la entrada.
- El tiempo de un algoritmo puede aumentar rápidamente en la medida que ϵ aumenta.
- Se tiene un esquema de aproximación “fully polynomial-time” cuando su tiempo de ejecución es polinomial no solo en n sino en $1/\epsilon$. Por ejemplo, $O((1/\epsilon)^3 n^2)$

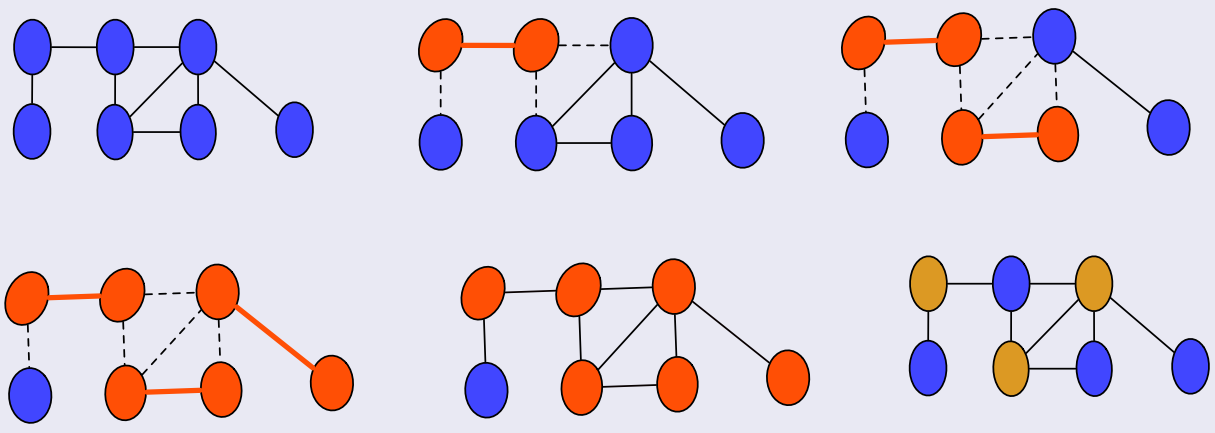
Ejemplo 1: Vertex-Cover

- El vertex cover de un grafo $G = (V, E)$ es un subset $V' \subseteq V$ tal que si (u, v) es un arco de G, entonces o $v \in V'$ o $u \in V'$ (o ambos). El tamaño del vertex-cover es el número de vértice en él.
- El problema del vertex-cover es encontrar un vertex cover de tamaño mínimo.
- Este es un problema NP-Complete.
- Encontrar la solución óptima es difícil, pero encontrar una aproximación no lo es. Hay un algoritmo 2-aproximación, es decir, un algoritmo que entrega no más que el doble del tamaño de la solución óptima.

Ejemplo 1: Vertex-Cover (cont.)

AproxVertexCover(G)
$C \leftarrow \emptyset$
$E' \leftarrow E[G]$
while $E' \neq \emptyset$ **do**
Sea (u,v) un arco arbitrario de E'
$C \leftarrow C \cup \{u, v\}$
remueva desde E' cada arco incidente en u o v
endwhile
return C

Ejemplo 1: Vertex-Cover (cont.)



Ejemplo 1: Vertex-Cover (cont.)

- Este algoritmo es polinomial ya que el loop del algoritmo corre hasta cuando cada vértice ha sido cubierto por algún vértice en C.
- El algoritmo es 2-aproximación por lo siguiente. Sea A el conjunto de arcos que son tomados en la primera sentencia dentro del *while* (línea 4). Para cubrir los arcos en A, cualquier vértice, y en particular en el óptimo C^*, debe incluir al menos un vértice extremo de cada arco en A. Los arcos en A no comparten vértices extremos, ya que luego de tomar un arco, todos los arcos que son incidentes a sus vértices extremos son eliminados desde E'. Así , dos arcos en A no pueden ser cubiertos por el mismo vértice en C^*. Consecuentemente, se tiene la cota inferior: $|C^*| \geq |A|$.

  Cada ejecución de la línea (primera después del *while*) toma un arco para el cual ninguno de sus vértices extremos es en C, alcanzando una cosa superior en el tamaño del vertex cover de $|C| = 2|A|$. Luego combiando las dos cotas, se tiene que $|C| = 2|A| \leq 2|C^*|$.

Ejemplo 2: Vendedor viajero

- Dado un grafo G no dirigido con pesos $c(u, v)$ asociados a cada arco $(u, v) \in E$, el vendedor viajero es el problema de encontrar el costo mínimo del **Hamiltonian cycle**. Este problem es NP-Complete.
- Denotemos $c(A)$ el costo total de los arcos en el subconjunto $A \in E$. Entonces $c(A) = \sum_{(u.v)\in A} c(u, v)$.
- Por desigualdad triangular, se tiene que $c(u, v) \leq c(u, w) + c(w, v)$.

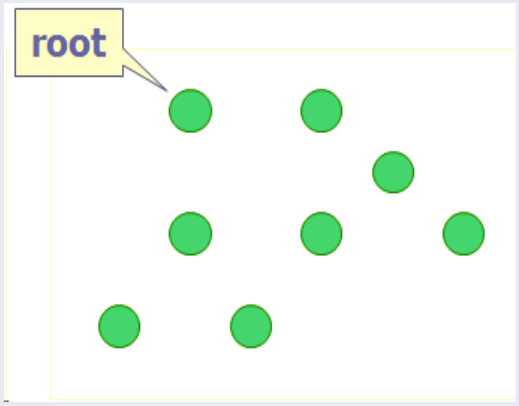
Vendedor viajero: Con desigualdad triangular

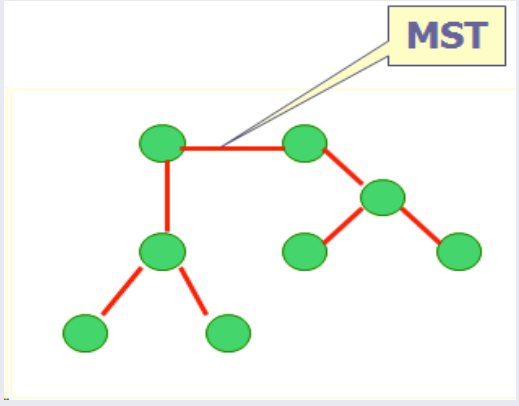
- Aplicando la desigualdad triangular, se puede calcular el mínimo árbol de cobertura (Mínimo Spanning Tree - MST) cuyos pesos sean una cota inferior en el largo del tour del vendedor viajero.
- Se usa el árbol de cobertura mínimo para crear un tour cuyo costo es no más que el doble del peso del MST, mientras el costo de la función satisfaga la desigualdad triangular.

Vendedor viajero: Con desigualdad triangular (cont.)

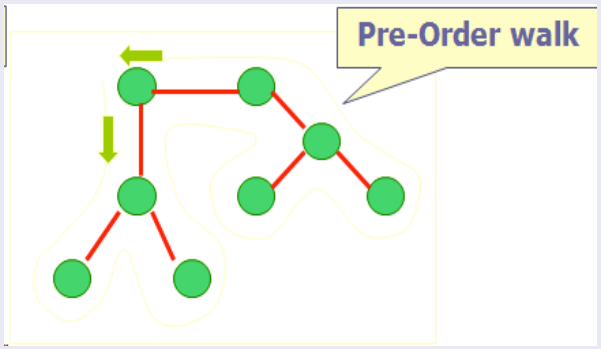
AproxTSPTour(G)
seleccione un vértice $r \in V[G]$ para la raíz del vértice
determine el MST T para G desde la raíz r (use MST-Prim(G) en el libro)
sea L la lista de vértices visitados en un árbol en preorden de T
return retorne el ciclo Hamiltoniano H que visita los vértices en el orden L

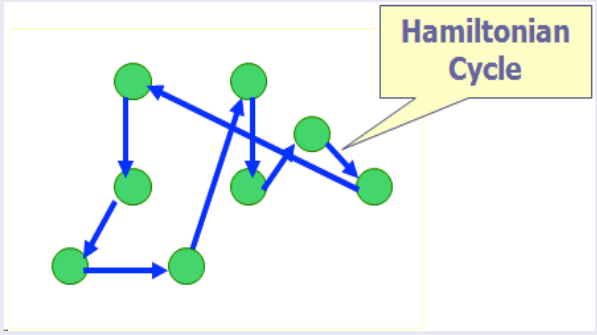
Vendedor viajero: Con desigualdad triangular (cont.)





Vendedor viajero: Con desigualdad triangular (cont.)





Vendedor viajero: Con desigualdad triangular (cont.)

- Este algoritmo es polinomial ya que el algoritmo para obtener el MST es polinomial al igual que el recorrido en preorder.
- El algoritmo es 2-aproximación por lo siguiente. Sea H^* el tour óptimo para el conjunto de vértices dados. Ya que se obtiene un spanning tree borrando cualquier elemento desde el tour, el peso del MST T es menor que el del costo del tour óptimo, esto es $c(T) \leq c(H^*)$.

  Cada caminata completa de T visita los vértices dos veces. Sea esta caminata W, entonces $c(W) = 2c(T)$, lo que implica que $c(W) \leq 2c(H^*)$. Lamentablemente W no es generalmente un tour ya que algunos vértices se vistan más de una vez. Por desigualdad triangular, sin embargo, se puede borrar cualquier visita a cualquier vértice en W y el costo no aumenta. Aplicando en forma repetida esto, se puede remover todo menos la primera visita a un vértice. Por lo tanto $c(H) \leq c(W)$ y $c(H) \leq 2c(H^*)$.
- Existen otros algoritmos que funcionan mejor y que podrían ser considerados.

Vendedor viajero: general

Si $P \neq NP$, entonces para cualquier constante $\rho \geq 1$, no existe un algoritmo de aproximación de tiempo polinomial con razón de aproximación ρ para el problema general del vendedor viajero (el problema donde se saca la condición de que la función de costo satisface la desigualdad triangular).
Esto se puede probar por contradicción.

- Asuma que existe para un entero $\rho \geq 1$ un algoritmo de aproximación polinomial A.
- Sea $G = (V, E)$ la instancia del problema del ciclo Hamiltoniano. Se transforma G en una instancia del problema del vendedor viajero en tiempo polinomial como sigue. Sea $G(V, E')$ el grafo completo en V de manera que $E' = \{(u, v) : u, v \in V \wedge u \neq v\}$. Se asigna un costo a cada arco de E' de la siguiente forma:

$$c(u, v) = \begin{cases} 1 & \text{si } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise} \end{cases}$$

Vendedor viajero: general (cont.)

- Considere ahora el problema del vendedor viajero (G', c). Si el grafo original G tiene un ciclo Hamiltoniano H, entonces, la función de costo c asigna a cada vértice de H un costo 1, y así (G', c) contiene un tour de costo $|V|$. Por lo contrario, si G no contiene un ciclo Hamiltoniano, entonces cualquier tour debe usar algún arco no en E, lo que implica que el tour tiene un costo de al menos:

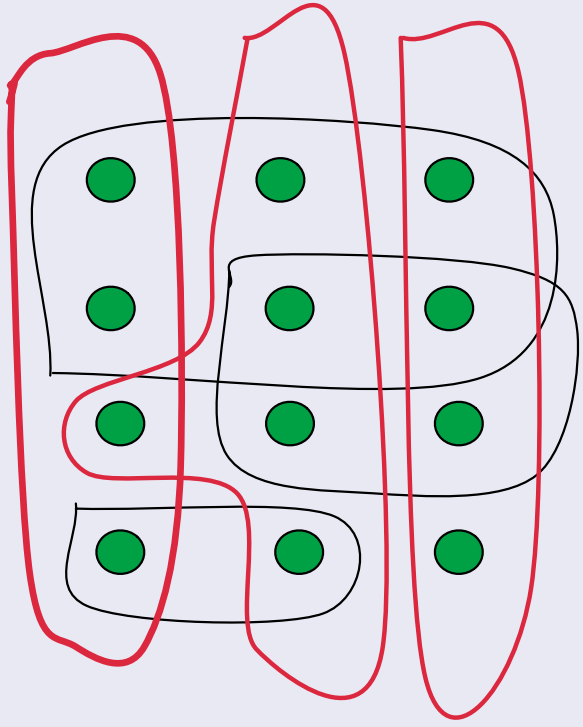
$$(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$$

- Ya que los arcos que no están en G son más costosos, existe un gap entre el costo de un ciclo Hamiltoniano en G y cualquier otro. Entonces si usamos el algoritmo A, puede que no podamos obtener una solución al problema del vendedor viajero general que corra en tiempo polinomial.

Ejemplo 3: set-covering

- Es una generalización del problema de vertex-cover.
- Una instancia (X, F) de un problema de set-covering consiste de un conjunto finito X y un familia F de subconjunto de X, tal que cada elemento X pertenece al menos a un subconjunto de F.
- El problema es encontrar un subconjunto de F de menor tamaño tal que cubra todos los elementos de X.

Set-covering



En esta figura, los subconjuntos de elementos encerrados por líneas rojas forman un set-covering del conjunto de elementos totales.

Set-covering (cont.)

GreedySetCover(X, F)
$U \leftarrow X$
$C \leftarrow \emptyset$
while $U \neq \emptyset$ **do**
seleccione un $S \in F$ que maximice $|S \cap U|$
$U \leftarrow U - S$
$C \leftarrow C \cup \{S\}$
endwhile
return C

Set-covering (cont.)

GreedySetCover es un algoritmo de $\rho(n)-$aproximación de tiempo polinomial, donde $\rho(n) = H(max\{|S| : S \in F\})$, donde $H(d)$ representa el número armónico $H_d = \sum_{i=1}^{d} 1/i$ y $H(0) = 0$.

Para demostrarlo, se asigna un costo de 1 a cada conjunto seleccionado del algoritmo, distribuye el costo sobre todos los elementos cubiertos por primera vez, y entonces se usan estos costos para derivar la relación deseada entre el tamaño de la solución óptima y el tamaño de la solución encontrada por el algoritmo.

Set-covering (cont.)

- Sea c_x el costo del elemento $x \in X$ que es cubierto por primera vez por S_i. Entonces

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})|}$$

- A cada paso el algoritmo agrega un costo 1 y se tiene que $|C| = \sum_{x \in X} c_x$
- El costo de un cover óptimo es la suma sobre todos los elementos de cada subset, y debido a que la solución óptima contiene a un elemento en X al menos una vez, se tiene:

$$\begin{aligned} \sum_{S \in C^*} \sum_{x \in S} c_x &\geq \sum_{x \in X} c_x \\ \sum_{S \in C^*} \sum_{x \in S} c_x &\geq |C| \end{aligned}$$

- Si se tiene que $\sum_{x \in S} c_x \leq H(|S|)$, entonces $|C| \leq \sum_{S \in C^*} H(|S|)$, lo que es lo mismo que $|C| \leq |C^*| H(max\{|S| : S \in F\})$.
- Solo queda demostrar que $\sum_{x \in S} c_x \leq \sum_{S \in C^*} H(|S|)$.

Set-covering (cont.)

- Considere cualquier conjunto $S \in F$ y cualquier $i = 1, 2, \ldots, |C|$, y sea $u_i = |S - (S_1 \cup S_2 \cup \cdots \cup S_i)|$ el número de elementos en S que quedan sin cubrir después de que el algoritmo ha seleccionado $S_1, S_2, \ldots, S_i$. Se define $u_0 = |S|$ como el número de elementos de S, el cual están inicialmente sin cubrir.
- Sea k el último índice tal que $u_k = 0$, tal que cada elemento de S está cubierto por algún conjunto en $S_1, \ldots, S_k$ pero existen algunos elementos en S no cubiertos por $S_1, \ldots, S_{k-1}$. Entonces $u_{i-1} \geq u_i$ y $u_{i-1} - u_i$ elementos son cubiertos por primera vez por S_i. Entonces:

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})|}$$

- Observe que $|S_i - (S_1 \cup S_2 \cdots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cdots \cup S_{i-1})| = u_{i-1}$, ya que la elección greedy garantiza que S no puede cubrir más nuevos elementos que S_i. Entonces

$$\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{u_{i-1}}$$

Set-covering (cont.)

- Finalmente

$$
\begin{aligned}
\sum_{x \in S} c_x &\leq \sum_{i=1}^{k} (u_{i-1} - u_i) \frac{1}{u_{i-1}} \\
&\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \text{ (porque } j \leq u_{i-1}) \\
&= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) \\
&= \sum_{i=1}^{k} \left(H(u_{i-1}) - H(u_i) \right) = H(u_0) - H(u_k) = H(u_0) - H(0) = H(u_0) \\
&= H(|S|)
\end{aligned}
$$

Ejercicio

Considere cada una de las siguientes palabras como un conjunto de letras: $\{arid, dash, drain, heard, lost, nose, shun, slate, snare, thread\}$. Muestre las coberturas que el algoritmo *GreedySetCover* produce cuando se elige en un empate por la palabra que aparece primero en el diccionario.

Aproximación aleatorizada: MAX-3-CNF Satisfiability

- Tal como algunos algoritmos aleatorizados determinan soluciones exactas, algunos algoritmos aleatorizados determinan soluciones aproximadas.
- Se dice que un algoritmo aletaorizado tiene una razón de aproximación $\rho(n)$ si, para cualquier entrada n, el costo esperado C de la solución producida por el algoritmo aleatorizado es en el factor de $\rho(n)$ del costo de C^* de una solución óptima:

$$max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Un algoritmos de aproximación aleatorizado es como un algoritmo de aproximación determinista, excepto que la razón de aproximación es para un costo aproximado.

Problema 3-CNF Satisfiability

Un *literal* en una fórmula booleana es una ocurrencia de una variable o su negación. Una *fórmula booleana* es en *forma normal conjuntiva or CNF* si es expresada como un *AND* de *cláusulas*, cada una de las cuales es el *OR* de uno o más literales. Una fórmula booleana es en 3-CNF si cada fórmula tiene exactamente 3 literales. Por ejemplo:

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

El problema es encontrar una asignación de las variables que haga que la fórmula sea verdadera. Este problema 3-CNF es NP-Complete.

Problema MAX 3-CNF Satisfiability

- El problema de aproximación asociado a 3-CNF es el problema de encontrar una asignación de variables que satisfaga tantas cláusulas como sean posible. A este problema de maximización se le conoce como **MAX-3-CNF**.
- Se muestra a continuación que una asignación aletaoria de 1 o 0 a cada variable con probabilidad 1/2 para ambos casos, lleva a un algoritmo aleatorizado de 8/7-aproximación.
- Se asume que ninguna cláusula contiene una variables y su negación ya que estas cláusulas son siempre verdaderas.

Problema MAX 3-CNF Satisfiability (cont.)

- Considere la variable aleatoria $Y_i = I\{$ cláusula i es satisfecha $\}$, de manera que $Y_i = 1$ cuando tenemos al menos un literal en la $i-$ésima cláusula que es 1.
- Ya que no existen dos literales en la misma cláusula que sean iguales, las asignaciones de las tres variables en una cláusula son independientes.
- Una cláusula no se satisface, si las tres variables son 0 con una probablidad de $Pr\{$ cláusula i es falsa $\} = (1/2)^3 = 1/8$.
- $Pr\{$ cláusula i es verdadera $\} = 1 - (1/2)^3 = 1 - 1/8 = 7/8$, lo que nos da que $E[Y_i] = 7/8$.

Problema MAX 3-CNF Satisfiability (cont.)

- Sea Y el número de cláusulas que se satisfacen en general, entonces $Y = Y_1 + Y_2 + \cdots + Y_m$. Entonces

$$\begin{aligned} E[Y] &= E\left[\sum_{i=1}^{m} Y_i\right] \\ &= \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} 7/8 = 7m/8 \end{aligned}$$

- m es una cota superior en el número de cláusula que se satisfacen y la razón de aproximación es a lo más $m/(7m/8) = 8/7$.

Programación lineal

- Muchos problemas de optimización toman la forma de maximización o minimización de un objetivo, dado un conjunto limitado de recursos y restricciones de competencia. Si se especifica el objetivo como una función lineal de ciertas variables, y si nosotros podemos especificar las restricciones en recursos como igualdades o desigualdades en estas variables, entonces se tiene un problema de programación lineal.

Programación lineal: weighted vertex cover

- En el problema de *minimum weighted vertex cover* (MWVC), se tiene un grafo no dirigido $G = (V, E)$, en el cual cada vértice $v \in V$ tiene asociado un peso positivo $w(v)$. Para cualquier vértice cover $V' \subseteq V$, se define el peso de la cobertura como $w(C') = \sum_{v \in V'} w(v)$. El objetivo es encontrar el vértice cover con mínimo peso.
- Este problema no puede usar el algoritmo que usamos para una cobertura sin pesos, ni una solución aletaorizada, ya que entregan soluciones que están muy lejos de la óptima.
- Se usa un cota inferior en el peso del MWVC usando programación lineal y se redondea esta solución para obtener una cobertura de los vértices.

Programación lineal: weighted vertex cover (cont.)

- Asuma que se asocia una variable $x(v)$ para cada $v \in V$, y que se requiere que $x(v)$ es 0 o 1 para cada $v \in V$. Se pone v en el vertex si y solo si $x(v) = 1$. Entonces se escribe una restricción que para cualquier arco (u, v), al menos uno de los u y v deben estar en el vertex cover tal que $x(u) + x(v) \geq 1$.
- La formulación del problema, conocido como *0-1 integer program* es la siguiente:

  Minimize $\sum_{v \in V} w(v)x(v)$ sujeto a que

$$\begin{array}{ll} x(u) + x(v) \geq 1 & \text{para cada } (u, v) \in E \\ x(v) \in \{0, 1\} & \text{para cada } v \in V \end{array}$$

Programación lineal: weighted vertex cover (cont.)

- Si se remueve la restricciones de que $x(v) \in \{0, 1\}$ y se reemplaza por $0 \leq x(v) \leq 1$, entonces se tiene le siguiente programa de lineal que se conoce como *linear-programming relaxation*: Minimize $\sum_{v \in V} w(v)x(v)$ sujeto a que

$$\begin{array}{ll} x(u) + x(v) \geq 1 & \text{para cada } (u, v) \in E \\ x(v) \leq 1 & \text{para cada } v \in V \\ x(v) \geq 0 & \text{para cada } v \in V \end{array}$$

- El valor de una solución óptima a un programa lineal da una cota inferior al valor de la solución óptima a el 0-1 integer program, y por lo tanto, es una cota inferior en el peso óptimo en el problema del *minimum − weight* vertex cover.

Programación lineal: weighted vertex cover (cont.)

- El procedimiento que sigue usa la solución del linear-programming relaxation para construir una solución aproximada al problema del minimum-weight vertex cover.

  ApproxMinWeightVC(G, w)
  $C \leftarrow \emptyset$
  calcule $\bar{x}$, una solución óptima al linear-programming relaxation
  for each $v \in V$ **do**
  if $\bar{x}(v) \geq 1/2$ **then**
  $C = C \cup \{v\}$
  endif
  endfor
  return C

- La solución óptima al programa de optimización da a cada vértice v un valor asociado $0 \leq \bar{x} \leq 1$. Se usan estos valores entonces para elegir qué vértices incorporar al vertex cover.

Programación lineal: weighted vertex cover (cont.)

- El algoritmo *ApproxMinWeightVC*(G, w) es un algoritmo 2-aproximación de tiempo polinomial al problema del minimum-weighted vertex cover.
- Es polinomial porque el programa de programación lineal en polinomial.
- Para demostrar que es 2-aproximación, considere C^* la solución óptima y sea z^* el valor de una solución óptima al problema de linear-programming relaxation. Ya que el vertex cover óptimo es una solución posible a la solución lineal, z^* debe ser una cota inferior en $w(C^*)$, esto es $z^* \leq w(C^*)$.
- Sea C el resultado del algoritmo, y considere cualquier arco $(u, v) \in E$. Se tiene que $x(u) + x(v) \geq 1$, lo cual implica que al menos uno de los $x(u)$ y $x(v)$ es al menos $1/2$. Entonces al menos uno de los u y v es incluido en el vertex cover y así cada vértice es cubierto.

Programación lineal: weighted vertex cover (cont.)

- Si se consideran los pesos del cover, se tiene:

$$\begin{aligned} z^* &= \sum_{v \in V} w(v)\bar{x}(v) \\ &\geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v)\bar{x}(v) \geq \sum_{v \in V : \bar{x}(v) \geq 1/2} w(v)1/2 \\ &= 1/2 \sum_{v \in C} w(v) = 1/2 w(C) \end{aligned}$$

- Entonces se tiene que $W(C) \leq 2z^* \leq 2w(C^*)$.