

Notas de Curso

Introducción a los Problemas NP-Completos (Ejemplos)

Dra. María De Luz Gasca Soto

Licenciatura en Ciencias de la Computación

Facultad de Ciencias, UNAM.

Revisión, Enero 2021

5. Ejemplos de Problemas NP-Complejos.

En esta sección, probaremos que los siguientes cinco problemas son NP-Complejos:

1. Cubierta de vértices, *Vertex Cover*;
2. Conjunto Dominante, *Dominating Set*;
3. 3-SAT
4. 3 Coloración, *3-Coloring*;
5. Clan, *Clique*.

Cada uno de ellos se describe con detalle más adelante. Se demuestra que son NP-completos con técnicas clásicas. Para probar que un nuevo problema es NP-Complejo debemos demostrar primero que pertenece a la clase NP, lo cual usualmente es fácil – pero no siempre. Después reducimos un problema conocido como NP-Complejo a nuestro nuevo problema en tiempo polinomial.

5.1 El Problema de la Cubierta de Vértices.

Sea $G=(V,E)$ una gráfica no dirigida. Una **Cubierta de vértices** de G es un conjunto de vértices tal que cada arista en G incide en al menos uno de estos vértices.

Ejemplo. Considere la gráfica $G=(V,E)$ de la Figura 8(a). Una cubierta de vértices trivial para la gráfica es su propio conjunto V de vértices. El sub-conjunto de vértices $Cv=\{4,7,6,1,3\}$ representa una cubierta de vértices para la gráfica G . La Figura 8(b) ilustra como el conjunto Cv cubre todas las aristas de la gráfica. Otro sub-conjunto de vértices $Cw = \{4,7,6,1,5\}$ representa también una cubierta de vértices para la G .

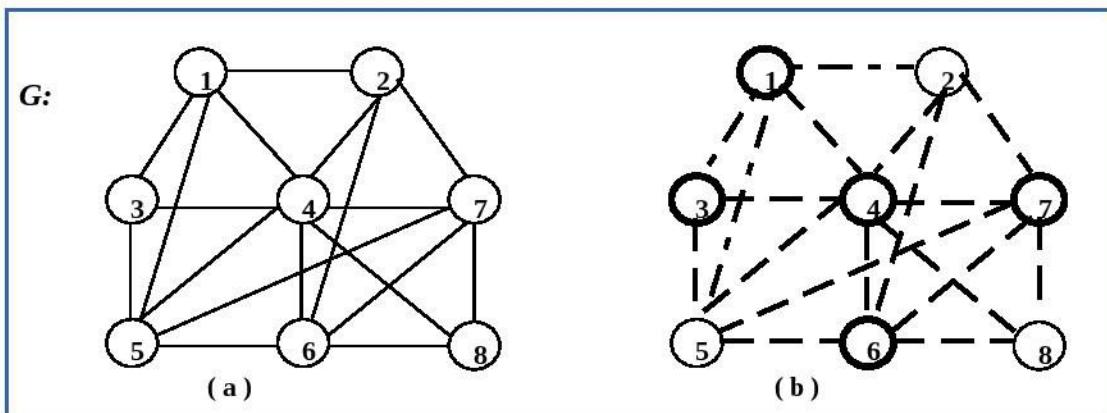


Figura 8: Ejemplo de una Cubierta de vértices

Problema CV: Cubierta de Vértices. Dada una gráfica $G=(V,E)$ no dirigida y un entero positivo k , determinar si G tiene una cubierta de vértices que contenga menos de k vértices.

Ejemplo. Considere la gráfica $G=(V,E)$ de la Figura 8(a). Si $k=5$, si podremos encontrar una cubierta de vértices de tamaño 5, pero si $k=4$ no será posible hacerlo.

Teorema 4: Cubierta de Vértices. El problema de la cubierta de vértices es NP-Completo.

Demostración.

El problema **CV** pertenece a la clase NP, ya que se puede adivinar o sugerir una cubierta de tamaño menor que k y verificarlo fácilmente en tiempo polinomial.

Para probar que el problema de la cubierta de vértices es NP-Completo tenemos que reducirlo a un problema NP-Completo ya conocido. Elegimos el Problema del **Clan**.

Construiremos la reducción: **Clan** \leq_{C} **CV**.

Dada una gráfica $G=(V, E)$ no dirigida, un clan C en G es una subgráfica de G tal que todos los vértices en C están conectados entre sí; es decir, un clan es una subgráfica completa de G . El problema del clan consiste en determinar si una gráfica G contiene un clan de tamaño mayor que k .

Debemos transformar un ejemplar arbitrario del problema del Clan a un ejemplar arbitrario del problema **CV** de tal forma que la respuesta del problema del Clan es positiva *si y sólo si* la respuesta correspondiente al problema de la cubierta de vértices es positiva.

Sean la gráfica $G=(V, E)$ y k un ejemplar para el problema del Clan. Sea la gráfica $G^c=(V^c, E^c)$ el complemento de G . La gráfica G^c tiene el mismo conjunto de vértices que G y dos vértices en G^c están conectados si y sólo si no están conectados en G .

Afirmamos que: el problema del clan se reduce al problema de la cubierta de vértices representado por la gráfica G^c y el entero $n-k$, donde n es el número de vértices en G .

\Rightarrow Suponga que $C = (U, F)$ es un clan en G . El conjunto de vértices $V \setminus U$ cubre todas las aristas de G^c , ya que en G^c no hay aristas que conecten vértices en U , todas están en G . Entonces, $V \setminus U$ es una cubierta de vértices para G^c . Por lo tanto, G tiene un clan de tamaño k y G^c tiene una cubierta de tamaño $n-k$.

\Leftarrow Sea D una cubierta en G^c , entonces D cubre todas las aristas en G^c , así en G^c no podría contener vértices en $V \setminus D$. Entonces genera un clan en G . Por lo tanto, existe una cubierta de vértices de tamaño k en G^c , entonces existe un clan de tamaño $n-k$. $\diamond \diamond \diamond$

Esta reducción puede obviamente ser efectuada en tiempo polinomial, ya que sólo requiere de la construcción de G^c a partir de G y el cálculo de $n-k$.

5.2 El Problema del Conjunto Dominante.

Sea $G=(V,E)$ una gráfica no dirigida. Un Conjunto Dominante de Vértices D es un conjunto de vértices en G tal que cada vértice de G está en D o es adyacente a al menos un vértice en D .

Ejemplo. Considere la gráfica $G=(V,E)$ de la Figura 9(a), un conjunto dominante de vértices para G resulta ser $D_v = \{x, r, z\}$. Otro conjunto dominante es $D_w = \{z, t\}$. La Figura 9(b) ilustra cómo los elementos del conjunto D_w son adyacentes al resto de los vértices en G .

Problema CDV: Conjunto Dominante de Vértices. Dada una gráfica $G=(V,E)$ no dirigida y un entero positivo k , determinar si G tiene conjunto dominante con a lo más k vértices.

Ejemplo. Para la gráfica de la Figura 9, tenemos que si $k=2$ podemos encontrar un conjunto dominante, de hecho resulta ser $D_w = \{z, t\}$.

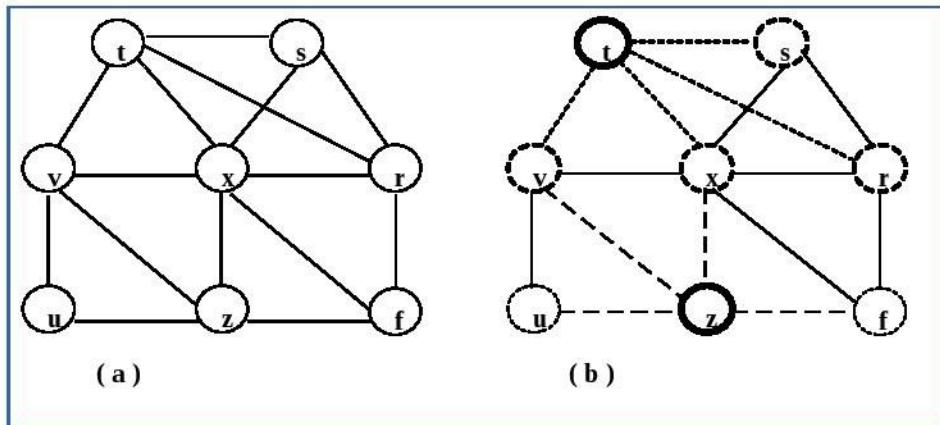


Figura 9: Ejemplo de un Conjunto Dominante de Vértices

Teorema 5: Conjunto Dominate de Vértices.

El problema del conjunto dominante **CDV** de vértices es NP-Completo.

Demostración.

El problema CDV pertenece a la clase NP, ya que podemos proponer un conjunto de tamaño menor o igual a k y verificar fácilmente en tiempo polinomial si es un conjunto dominante.

Reduciremos el problema de la cubierta de vértices al problema CDV: **CV** \propto **CDV**. Dado un ejemplar arbitrario del problema CV, digamos $\langle G=(V,E), k \rangle$ con $|V|=n$ y $|E|=m$, el objetivo es construir una nueva gráfica G' que tenga un conjunto dominante de cierto tamaño si y sólo si la gráfica G tiene una cubierta de vértices de tamaño a lo más k .

Empezamos con la gráfica G y añadimos m nuevos vértices y $2m$ nuevas aristas a G de la siguiente manera: Por cada arista (v,w) de G agregamos un nuevo vértice vw y dos aristas, (v,vw) y (w,vw) . Lo que estamos haciendo es transformar cada arco en un triángulo. La Figura 10 ilustra esta construcción. Denominamos a la nueva gráfica G' , es fácil ver que puede construirse en tiempo polinomial.

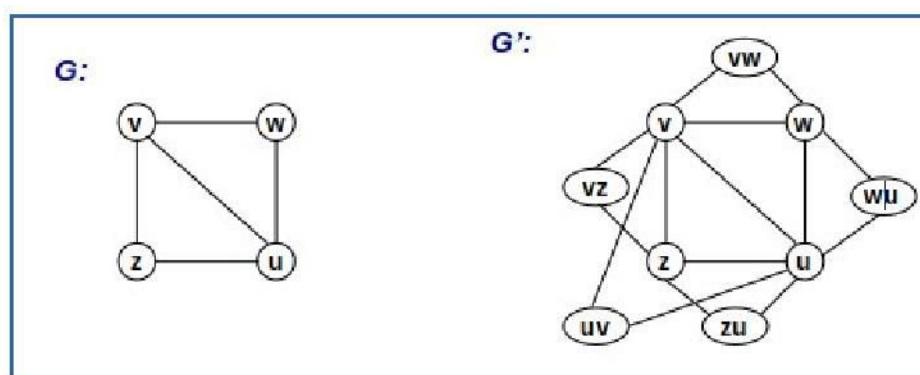


Figura 10: Construcción de la Gráfica \mathbf{G}' .

Afirmamos que: La gráfica G' tienen un conjunto dominante de vértices de tamaño m si y sólo si la gráfica G tiene una cubierta de vértices de tamaño m .

⇒ Sea \mathbf{D} el conjunto dominante de vértices para G' . Si \mathbf{D} contiene cualquiera de los nuevos vértices vw , entonces este vértice puede ser reemplazado por v o w y el

conjunto continuará siendo dominante, ya que v y w cubren los vértices que el vértice vw cubría. Así, sin pérdida de generalidad, podemos suponer que D contiene sólo vértices de G . Pero como D domina a todos los nuevos vértices, entonces contiene al menos un vértice de cada arista original. Por lo tanto, D cubre a la gráfica G .

\Leftarrow) Si C es una cubierta para G , entonces cada arco es cubierto por C y todos los nuevos vértices están dominados. Los vértices originales también están dominados ya que todos los arcos están cubiertos. $\diamond\diamond\diamond$

5.3 El Problema 3-SAT.

El Problema 3-SAT es una simplificación del problema SAT. Un ejemplar del 3-SAT es una expresión lógica en CNF tal que cada cláusula C contiene exactamente tres variables.

Problema 3-SAT: Dada una expresión lógica en CNF tal que cada cláusula C contiene exactamente tres variables, determinar si C se satisface.

Teorema 6: 3-SAT. El problema 3-SAT es NP-Completo.

Demuestra. Este problema es más fácil que el SAT original, pues sólo contiene tres variables por cláusula. Se mostrará que una solución del 3-SAT puede ser usada para resolver el SAT original: **SAT \leq 3-SAT**.

Primero, 3-SAT es claramente NP. Podemos sugerir una asignación de verdad y verificar en tiempo polinomial que la expresión se satisface.

Sea E un ejemplar arbitrario del SAT. Reemplazamos cada cláusula de E con diversas cláusulas, cada una con tres variables. Sea $C = (x_1 + x_2 + \dots + x_k)$ una cláusula arbitraria de E tal que $k \geq 4$. Escribimos cada variable en su forma positiva (no usamos x_i^-) únicamente por notación conveniente. Mostramos ahora como reemplazar C con diversas cláusulas, cada una con tres variables. La idea es introducir nuevas variables y_1, y_2, \dots, y_{k-3} que conformen la cláusula en una fórmula 3-SAT, sin afectar su capacidad de ser satisfecha (*satisfiability*). Usamos nuevas y diferentes variables para cada cláusula. C es transformada en C' de la siguiente manera:

$$C' = (x_1 + x_2 + y_1) \cdot (x_3 + y_1^- + y_2) \cdot (x_4 + y_2^- + y_3) \cdot \dots \cdot (x_{k-1} + x_4 + y_{k-3}^-)$$

Afirmamos que: C' se satisface si y sólo si C se satisface.

\Leftarrow) Si la expresión C se satisface, una de las x_i debe ser 1. En tal caso, podemos asignar valores a las y_i en C' tal que todas las cláusulas en C' también se satisfacen. Por ejemplo, si $x_3 = 1$, entonces asignamos $y_1 = 1$ y $y_2 = 0$ y el resto de las serán cero, lo cual satisface C' .

\Rightarrow) Si C' se satisface, podemos afirmar que al menos una de las x_i debe ser 1. Si todas las x_i son cero, la expresión se convierte en:

$$(y_1) \cdot (y_1^- + y_2) \cdot (y_2^- + y_3) \cdot \dots \cdot (y_{k-3}^-)$$

y claramente tal expresión no se satisface.

Usando esta reducción, se puede reemplazar cualquier cláusula que tenga más de tres variables con varias cláusulas con exactamente tres variables. Esto deja a las cláusulas transformadas con una o dos variables. Si C tiene sólo dos variables, $C = (x_1 + x_2)$, entonces $C' = (x_1 + x_2 + z) \cdot (x_1 + x_2 + z^-)$ donde z es una nueva variable.

Finalmente, si tiene una variable, $C = (x_1)$, entonces

$C' = (x_1 + y + z) \cdot (x_1 + y + z^-) \cdot (x_1 + y^- + z) \cdot (x_1 + y^- + z^-)$, donde z y y son las nuevas variables.

De esta manera, reducimos un ejemplar del SAT a un ejemplar del 3-SAT de tal forma que un ejemplar se satisface si y sólo si el otro también se satisface. Es claro que esta reducción puede hacerse en tiempo polinomial. $\diamond \diamond \diamond$

5.4 El Problema del Clan.

Sea $G=(V,E)$ una gráfica no dirigida. Un clan C en una gráfica G , es una subgráfica completa de G .

Problema Clan, PC: Dada una gráfica no dirigida $G=(V,E)$ y un entero positivo k , determinar si G tiene un clan de tamaño mayor o igual que k .

Teorema 7: Clan. El problema del Clan es NP-Completo.

Demostración.

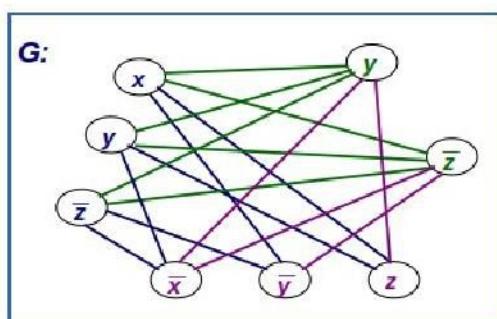
Este problema del clan pertenece a la clase NP ya que podemos sugerir un subconjunto de al menos k vértices y verificar si es un clan en tiempo polinomial.

Reduciremos SAT al problema del clan: **SAT \leq PC**. Sea E una expresión lógica arbitraria en CNF: $E = E_1 \cdot E_2 \cdot \dots \cdot E_m$. Sea $E_i = (x+y+z+w)$, usaremos cuatro variables para ilustrar. Asociamos una columna de cuatro vértices con las variables en E_i , aún si ellas aparecen en otras cláusulas. Esto es, la gráfica G tendrá un vértice por cada aparición de cada variable. La cuestión es cómo conectar estos vértices tal que G contenga un clan de tamaño mayor o igual a k si y sólo si E se satisface.

Nótese que tenemos libertad de elegir el valor de k , ya que queremos reducir el SAT al problema del Clan, lo cual significa resolver el SAT usando una solución del problema del clan. Una solución para el clan deberá funcionar para toda k . Esta es una importante flexibilidad que es usada frecuentemente en las pruebas de NP-Completos. Elegimos $k = m$, el número de cláusulas.

Las aristas de G se definen de la siguiente forma: Vértices de la misma columna (vértices asociados con variables de la misma cláusula) no están conectados. Vértices de diferentes columnas están casi siempre conectados, cuando ellos corresponden a la misma variable que aparece en forma complementaria. Esto es, la única forma en que no conectamos dos vértices de diferentes cláusulas es cuando uno corresponde a la variable x y el otro a \bar{x} . G se construye en tiempo polinomial.

Ejemplo. Para el ejemplar $E = (x + y + z^-) \cdot (x^- + y^- + z) \cdot (y + z^-)$ se tiene la siguiente gráfica:



Afirmamos que: G tiene un clan de tamaño mayor o igual a m si y sólo si E se satisface. En efecto, la construcción garantiza que el clan de tamaño máximo no excede m .

\Leftarrow) Supongamos que E se satisface. Entonces existe una asignación de verdad tal que cada cláusula contiene al menos una variable con valor 1. Elegiremos el vértice correspondiente a esa variable para el clan. Si más de una variable en una cláusula tiene valor 1, elegimos arbitrariamente. El resultado es un clan, ya que la única vez que dos vértices de dos columnas no están conectados es cuando ellos son el complemento uno del otro, lo cual, por supuesto, no puede suceder en una asignación de verdad consistente.

\Rightarrow) Supongamos que G contiene un clan de tamaño mayor o igual a m . El clan debe contener exactamente un vértice de cada columna, ya que dos vértices de una misma columna nunca están conectados. Asignamos a las correspondientes variables el valor de 1. Si algunas variables no están asignadas de esta manera, pueden ser asignados arbitrariamente. Como todos los vértices en el clan están conectados entre sí, podemos asegurar que x y x^- nunca están conectados, y esta asignación de verdad es consistente. $\diamond \diamond \diamond$

5.5 El Problema 3-Coloración.

Sea $G=(V,E)$ una gráfica no dirigida. Una **coloración válida** de G es una asignación de valores para los vértices tal que a cada vértice se le asigna un color y dos vértices adyacentes no tienen el mismo color.

Problema 3-Coloración, 3C: Dada una gráfica no dirigida $G=(V,E)$ determinar si G puede ser coloreada con tres colores.

Teorema 7: 3-Coloración. El problema de 3-Coloración es NP-Completo.

Demostración.

Este problema pertenece a la clase NP ya que podemos sugerir una 3 coloración para los vértices de G y verificar si es válida en tiempo polinomial.

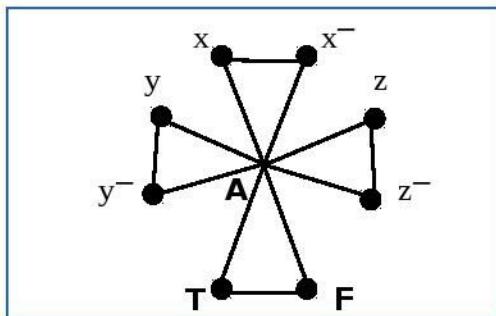
Reduciremos 3-SAT al problema de 3-Coloración: **3-SAT \leq 3C**. Esta resulta ser una de las pruebas más complicadas, por las siguientes dos razones:

- 1) los dos problemas trabajan con diferentes objetos, uno con expresiones lógicas y el otro con gráficas;
- 2) No podemos reemplazar un objeto, digamos vértice arista, con otro, digamos cláusula. Tenemos que repartir o distribuir la estructura completa.

La idea es usar bloques y ligarlos. Sea E un ejemplar arbitrario de 3-SAT. Tenemos que construir una gráfica G tal que E se satisface si y sólo si G es tres coloreable.

Primero construimos un triángulo principal M , esto significa que M requiere al menos tres colores. Etiquetamos M con los colores: **T**, *True*, **F**, *False* y **A** (otro). Estos colores son usados únicamente para la prueba, no son parte de la gráfica. Despues los asociamos con una asignación de valores de verdad para E . Para cada variable x construimos un triángulo Mx cuyos vértices son etiquetados con x , x^- y A , donde A es el mismo vértice en M . Gráficamente queda ilustrado en la Figura 11.

De esta forma, si existen k variables tendremos $k+1$ triángulos, todos compartiendo el vértice A . La idea es que, si x es coloreado con T , entonces x^- debe ser coloreada con F , ya que ambos están conectados a A , y viceversa. Lo cual es consistente con el significado de x .

Figura 11. Primera parte de la construcción de G para la 3-coloración.

Tenemos que imponer la condición de que al menos una variable en cada cláusula tiene valor 1, *true*. Realizamos la siguiente restricción: Supongamos que la cláusula es $(x+y+z)$, introducimos seis nuevos vértices y los conectamos a los vértices existentes. la Figura 12 muestra la sub-gráfica correspondiente las cláusulas de la reducción.

Las etiquetas son consistentes, así que hay un sólo vértice en la gráfica total etiquetado con T y un vértice para cada x , y o z . Llamamos a los tres nuevos vértices conectados a T , vértices exteriores O y los del triángulo, vértices interiores I .

Afirmamos que: esta construcción garantiza que si no son usados más de tres colores entonces al menos uno de los vértices x , y o z debe ser colorado con T . Ninguno puede estar colorado con A , ya que todos están unidos a A . Si todos están coloreados con F , entonces los tres nuevos conectados a ellos deben ser colorados con A . Entonces, ¡el triángulo interno no puede ser coloreado con tres colores!

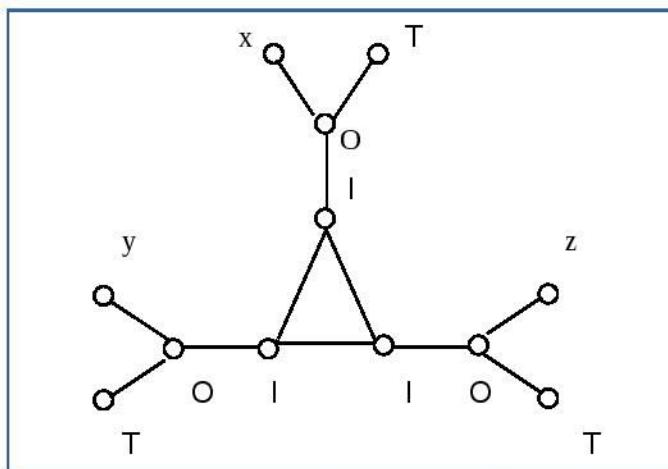
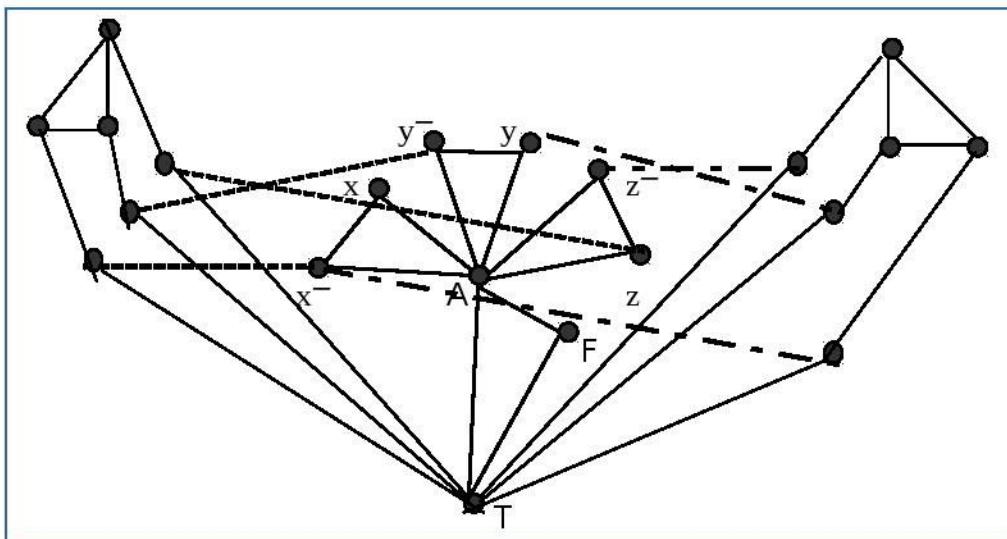


Figura 12. Sub-gráfica correspondiente a la reducción

La gráfica completa correspondiente a $E = (x^- + v + z^-) \cdot (x^- + v^- + z)$ está dada en la Figura 13.

Ahora podemos completar la prueba. Tenemos dos casos:

1. Si E se satisface entonces G puede ser coloreada con tres colores.
2. Si G es 3-coloreable entonces E se satisface.

Figura 13. Gráfica correspondiente a $E = (x + v + z^-) \cdot (x^- + v + z)$.

Ahora podemos completar la prueba. Tenemos dos casos:

1. Si E se satisface entonces G puede ser coloreada con tres colores.
2. Si G es 3-coloreable entonces E se satisface.

Si E se satisface entonces existe una asignación de verdad que lo satisface. Coloreamos los vértices asociados con las variables de acuerdo con su asignación de verdad, T si $x=1$ y F en otro caso.

El triángulo M se colorea con T , F y A como se indicó. Cada cláusula debe tener al menos una variable cuyo valor sea 1. Entonces podemos colorear los correspondientes vértices exteriores con F , el resto con A ; y los vértices interiores se colorean de acuerdo al valor del O correspondiente. Por lo tanto, podemos colorear con tres colores.

Inversamente, si G es 3-coloreable, nombramos los colores de acuerdo a la coloración de M , el cual debe ser coloreado con 3 colores. Por los triángulos de la Figura 11, los colores de las variables correspondientes corresponden a una asignación de verdad consistente. La construcción de la Figura 13 garantiza que al menos una variable en cada cláusula es coloreada con T . Finalmente, G puede ser construida en tiempo polinomial, lo cual completa la prueba. $\diamond \diamond \diamond$

5.6 Observaciones Generales.

Hemos discutido muy brevemente algunos métodos generales para probar que un problema Q es NP-Completo. La primera condición, mostrar que Q pertenece a NP, es usualmente fácil, pero no siempre. Entonces tenemos que seleccionar un problema NP-Completo conocido que sea similar a Q . Resulta difícil definir esta similaridad ya que algunos problemas parecen muy diferentes, como el Clan y el SAT. Encontrar el problema correcto a partir del cual reducir es algunas veces una difícil tarea, la cual es aprendida únicamente con la experiencia. Resulta ser buena idea tratar diferentes reducciones con diversos problemas hasta encontrar la reducción exitosa.

Hacemos hincapié en que la reducción es hecha de un ejemplar arbitrario de un problema NP-Completo al problema Q , el cual queremos probar que es NP-completo. El error

más común en tales pruebas es efectuar al revés la reducción. Una manera de recordar el orden correcto es asegurar que el problema NP-Completo puede ser resuelto por un algoritmo tipo caja negra para **Q**. Esto es un pequeño "contra-tiempo" La cosa más natural que hacemos cuando nos dan un problema es tratar de resolverlo. Aquí, sin embargo, tratamos de mostrar que podemos resolver otro problema, el que sabemos es NP-completo, usando la solución de **Q**. No tratamos nunca de resolver **Q**.

Existen diferentes grados de libertad que pueden ser usados en la reducción. Por ejemplo, si **Q** incluye un parámetro, entonces éste valor puede ser usado o transformado de manera conveniente. En contraste con el parámetro en el problema que es reducido a **Q**, el cual no puede ser fijado. Otra vez, **Q** es sólo una herramienta para resolver el problema NP-completo; por lo tanto, podemos usarlo de la manera que queramos. El problema **Q** puede ser restringido a casos especiales en otras formas, además de fijar su parámetro. Por ejemplo, podríamos querer usar sólo cierto tipo de entradas para **Q**.

Otro aspecto sobre la flexibilidad es el hecho de que la eficiencia de la reducción no importa, siempre y cuando la reducción se realiza en tiempo polinomial. No sólo podemos ignorar las constantes y duplicar el tamaño del problema, sino que además podríamos tener que el tamaño del problema sea el cuadrado del tamaño original! Podemos introducir un número polinomial de nuevas variables, podemos reemplazar cada vértice en la gráfica por una nueva gráfica y cosas así.

No existe la necesidad de ser eficiente, mientras las cotas sean polinomiales, ya que la reducción no será convertida en un algoritmo, al menos no hasta que se pruebe que P es igual a NP, si es que esto llega a suceder.

Existen algunas técnicas comunes usadas en la construcción de las reducciones. La más simple es mostrar que un problema NP-Completo es un caso especial de **Q**. Si esto resulta ser así, la prueba será inmediata, ya que solucionar **Q** implica resolver el problema NP-Completo. Por ejemplo, consideremos el problema de la cubierta de vértices. La entrada para el problema es una colección de subconjuntos S_1, S_2, \dots, S_n de un conjunto **U** y un entero **k**. El problema es determinar si existe un subconjunto $W \subseteq U$, con a lo más **k** elementos, que contenga al menos un elemento de cada conjunto S_i . Podemos ver que el problema de la cubierta de vértices es un caso especial del problema cubierta de conjuntos, set-cover, en el cual **U** corresponde a un conjunto de vértices **V**, y cada conjunto S_i corresponde a una arista y los dos vértices incidentes a tal arista. Así, si podemos resolver el problema de la cubierta de conjuntos para conjuntos arbitrarios, entonces podemos resolver el problema de la cubierta de vértices.

Sin embargo, debemos ser muy cuidadosos cuando usemos esta técnica. No resulta ser verdad, en general, que agregar más requerimientos a un problema lo hace más difícil. Retomemos el problema de la cubierta de vértices. Suponga que añadimos la restricción de que la cubierta de vértices no incluya dos vértices adyacentes. En otras palabras, estamos buscando un conjunto de vértices más pequeño que forme una cubierta de vértices y un conjunto independiente¹ a la vez. Este problema parece más difícil que cualquiera de los dos problemas originales, ya que tenemos que preocuparnos por más requerimientos. Sin embargo, este problema es más fácil y puede ser resuelto en tiempo polinomial.

Otra técnica relativamente fácil involucra reducciones locales. En este caso, un objeto en un problema es proyectado en un objeto de otro problema. La proyección se realiza localmente, independientemente de otros objetos del problema. La prueba de que el problema del conjunto dominante es NP-completo, sigue este patrón. Reemplazamos cada arista en la gráfica por un triángulo en otra gráfica. Estos cambios locales son suficientes para reducir el problema. La dificultad en esta técnica es definir los objetos de la mejor manera.

¹ Un conjunto independiente es un conjunto de vértices que no son adyacentes entre sí.

La técnica más complicada es usar bloques, como lo hicimos para probar que el problema de 3-coloración es NP-completo. Usualmente, los bloques dependen de ellos entre sí y diseñar cada uno de ellos por separado es imposible. Tenemos que considerar todos los objetivos de los problemas para coordinar el diseño de los distintos bloques.

5.7 Más Problemas NP-Completos

La siguiente lista contiene algunos de los más útiles y conocidos problemas NP-Completos, que sirven como base para otras reducciones.

i) Ciclo Hamiltoniano. Un ciclo hamiltoniano en una gráfica es un ciclo simple que contiene cada vértice de la gráfica exactamente una vez. El problema consiste en determinar si una gráfica existe un ciclo hamiltoniano. El problema es NP-Completo tanto para gráficas no dirigidas y dirigidas. Reducción con Cubierta de Vértices.

ii) Agente Viajero. Sea $G=(V,E)$ una gráfica completa con pesos sobre los arcos. Un tour para el agente viajero es un ciclo hamiltoniano. El problema es determinar si, dada una gráfica G y un número w , existe un tour para el agente viajero tal que el costo total de las aristas que lo forman sea menor o igual que w . Reducción con Ciclo Hamiltoniano.

iii) Trayectoria Hamiltoniana. Una trayectoria hamiltoniana en una gráfica es una trayectoria simple que contiene cada vértice de la gráfica exactamente una vez. El problema es determinar si una gráfica dada contiene una trayectoria hamiltoniana. El problema es NP-Completo tanto para gráficas no dirigidas como para gráficas dirigidas. Reducción con Cubierta de Vértices.

iv) Conjunto Independiente. Un conjunto independiente en una gráfica no dirigida $G=(V,E)$ es un conjunto de vértices para el cual los vértices que lo conforman no son adyacentes entre sí. Dada una gráfica G y un entero k , el problema es determinar si G contiene un conjunto independiente con al menos k vértices. Reducción con Clan.

v) Apareamiento 3-dimensional (3DM). Sean X , Y y Z conjuntos disjuntos de tamaño k . Sea M un conjunto de tripletas (x,y,z) tal que $x \in X$, $y \in Y$, $z \in Z$. El problema consiste en determinar si existe un subconjunto M que contenga cada elemento exactamente una vez. R. con 3-SAT.

vi) Partición. La entrada es un conjunto X tal que $\forall x \in X$ se tiene asociado un tamaño $s(x)$. El problema es determinar si es posible dividir el conjunto en dos subconjuntos con exactamente el mismo tamaño en total. Reducción con A3-D.

vii) Problema de la Mochila. La entrada es un conjunto X tal que $\forall x \in X$ se tiene asociado un tamaño $s(x)$ y un valor $v(x)$. El problema es determinar si $\exists B \subseteq X$ tal que el tamaño total sea menor que s y cuyo valor total sea mayor que v . Reducción con Partición.

viii) BinPacking. La entrada es una secuencia de números a_1, \dots, a_n y otros dos números b y k . El problema es determinar si el conjunto puede ser dividido en k subconjuntos tal que la suma de los números en cada subconjunto sea menor o igual que b . Reducción con Partición.

6. Algoritmos de Aproximación.

La noción de problemas NP-Complejos es una base para una elegante teoría que nos permite identificar problemas para los cuales probablemente no existen algoritmos polinomiales que lo resuelva. Pero probar que un problema dado es NP-Complejo no hace que el problema desaparezca o que nos olvidemos de él. Aún necesitamos resolverlo. Las técnicas para resolver problemas NP-completos son generalmente diferentes de las que hemos revisado en los cursos básicos de algoritmos y estructuras de datos. Ciertamente, no podemos resolver un problema NP-Complejo con un algoritmo de tiempo polinomial, así que debemos suponer un compromiso concerniente con la funcionalidad del algoritmo a diseñar. Tal algoritmo debe ser óptimo,

robusto, debe tener un desempeño garantizado y debe proporcionar una buena solución. Un mismo algoritmo puede ser usado en diferentes situaciones y no siempre es posible alcanzar las características anteriores. Existen varias alternativas para atacar a los problemas NP-Complejos, todas sacrifican algo.

Un algoritmo que podría no llevarnos al resultado óptimo o preciso es llamado **Algoritmo de Aproximación**. éstos son de particular interés ya que garantizan una cota sobre el grado de imprecisión. En esta sección veremos algunos ejemplos.

6.1 Técnicas Backtracking and Branch-and-Bound

Describiremos estas técnicas a través de un ejemplo. Considere el **Problema de 3-coloración**, el cual involucra asignación de colores, bajo ciertas restricciones, a n vértices de una gráfica. Este es un ejemplo de un problema que requiere encontrar valores óptimos (colores en este caso) para n parámetros. En el ejemplo de 3-coloración, hay tres posibles valores, que corresponden a los colores, para cada parámetro. Por supuesto, sin considerar las aristas de la gráfica, el número posible de coloraciones válidas será un poco menor que 3^n , ya que las aristas imponen restricciones sobre la posible coloración.

Para explorar todas las posibles maneras de colorear los vértices podemos empezar asignando arbitrariamente un color a un vértice y continuar coloreando los demás vértices mientras se mantenga la restricción impuesta por las aristas - digamos que vértices adyacentes deben ser coloreados con diferente color. Cuando coloreamos un vértice, intentamos todos los posibles colores que sean consistentes con la coloración que se lleve hasta ese momento. Este proceso puede ser ejecutado por un algoritmo para recorrido en árboles el cual es la esencia de las técnicas *Backtracking and Branch-and-Bound*. Para evitar confusión entre los vértices de la gráfica y los del árbol, llamaremos a los vértices del árbol nodos.

La raíz del árbol corresponde al estado inicial del problema, y cada rama corresponde a una decisión concerniente al parámetro. Denotemos los tres colores por **R** para Rojo, **A** para Azul y **V** para Verde. Inicialmente, podemos tomar cualesquiera dos vértices adyacentes, digamos **v** y **w**, y colorearlos, por ejemplo con **A** y **V**. Ya que ellos tendrán colores diferentes en cualquier coloración valida, esta primera asignación no tiene importancia. La coloración de estos dos vértices corresponde al estado inicial del problema, el cual queda asociado con la raíz. El árbol se va construyendo como va siendo visitado, o recorrido. En cada nodo **t** del árbol, seleccionamos el siguiente vértice **u**, de la gráfica, a colorear. Agregamos uno dos o tres hijos a **t** según el número de colores que pueden ser usados para colorear al vértice **u**. Por ejemplo, si nuestra primera elección (después de **v** y **w**) es **u**, y si **u** es adyacente a **w** (que tiene color **V**) entonces hay dos posibles colores para **u**, **A** o **R**, y añadimos dos hijos a la raíz. Continuamos el proceso eligiendo uno de estos hijos. Después de que un vértice es coloreado, hay menos flexibilidad para colorear el resto de los vértices, por lo tanto el número de hijos es cada vez más pequeño a medida de que vamos recorriendo el árbol en profundidad.

Si logramos asignar colores a todos los vértices de la gráfica, entonces habremos terminado. Pero es posible que no alcancemos a colorear un vértice, ya que éste puede ser adyacente a vértices que fueron coloreados con los tres colores. En este punto, nos regresamos (*backtrack*) en el árbol, al menos un nivel, y exploramos otro hijo. Un ejemplo de una gráfica y de su correspondiente árbol *backtrack* para la 3-coloración es dado en la Figura 14. Note que en este caso, una vez que los colores para los vértices 1 y 2 quedan fijos, existe sólo una manera de colorear el resto de los vértices de la gráfica. Tal coloración puede encontrarse sobre la trayectoria de la extrema derecha en el árbol.

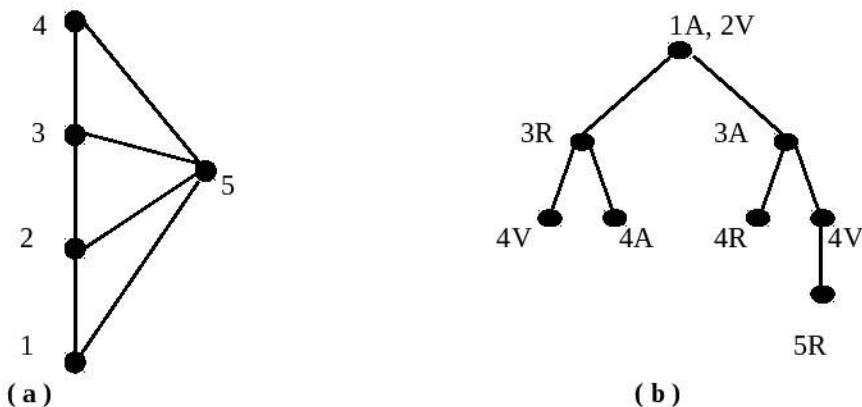


Figura 14. Ejemplo de 3-Coloración

Podemos pensar este algoritmo para recorrido de árboles como un algoritmo basado sobre inducción. Damos una hipótesis directa para incluir la coloración en gráficas para algunos de aquellos vértices que ya han sido coloreados. En otras palabras, la hipótesis de inducción, no manipulará la coloración en gráficas, pero si completará una 3-coloración parcial.

Hipótesis de Inducción: Sabemos como completar una 3-coloración de una gráfica que tiene menos de k vértices que aun no han sido coloreados, o determinar que tal 3-coloración no puede ser completada.

Dada una gráfica con de k vértices que aun no han sido coloreados, elegimos uno de ellos y encontramos todos los posibles colores que puedan asignársele. Si todos los colores ya han sido usados con sus vecinos, entonces la 3-coloración no puede ser completada. De otra manera, coloreamos el vértice con los posibles colores, uno a la vez, y resolvemos el problema para los vértices restantes, que son $k-1$, por inducción. Este proceso es ilustrado en el algoritmo de la Figura 15.

La Técnica *Branch-and-Bound* es una variación de *Backtracking* para problemas que involucran encontrar el mínimo o máximo de alguna función objetivo. Considere el problema general de coloración – ahora estamos interesados en encontrar el mínimo número de colores requeridos para colorear una gráfica.

Algoritmo 3-Coloración

Entrada: $G=(V,A)$ // una gráfica no dirigida.
U conjunto de vértices // ... que han sido coloreados y sus colores
Salida: Asignación de uno de los tres colores a cada vértice de G .

Inicio

Si $U=V$ **Entonces**
 Imprime "Coloración completada"; **halt**.

EnOtroCaso

Elige un vértice v que no esté en U ;

Para C desde 1 hasta 3 **Hacer**

Si ninguno vecino de v está coloreado con el color C

Entonces

Agregar v a U con color C ;

3-Coloración (G,U)

Fin

Figura 15. Algoritmo para una 3-Coloración.

Podríamos construir un árbol similar al que hicimos para la 3-coloración, pero el número de ramas sería muy grande. Cada nuevo vértice podría ser coloreado con uno de los colores ya usado o por un nuevo color. El algoritmo para la 3-coloración se modifica de dos maneras: (1) la constante 3 se reemplaza por el número máximo de colores a usar, y (2) el algoritmo no termina cuando $V=U$, ya que puede haber muchas mejores formas de colorear la gráfica.

El problema es que este algoritmo regresa (hace *backtrack*) únicamente cuando una hoja es alcanzada, es decir cuando $U=V$, ya que un color nuevo siempre puede ser asignado a un vértice. Así que este algoritmo tiene un desempeño muy pobre. Es posible mejorar el desempeño de este algoritmo haciendo la siguiente observación, la cual es la base del método *Branch-and-Bound*.

Supongamos que recorremos el árbol siempre hacia la izquierda y encontramos una coloración válida con k colores. Suponga además que, después de regresar varios pasos hacia arriba en el árbol, recorremos otra trayectoria y alcanzamos un vértice que requiera el color $k+1$. En este punto, podríamos regresar, esto es hacer *backtrack*, ya que hemos encontrado una mejor solución. Así k sirve como una cota (*bound*) para el regreso, *backtracking*. En cada nodo calculamos una cota mínima sobre la mejor solución que podemos encontrar recorriendo el árbol hacia abajo. Si tal cota mínima resulta ser mayor que otra solución conocida, regresamos. La clave para hacer eficientes algoritmos con el método *Branch-and-Bound* es calcular buenas cotas mínimas, o cotas máximas si se desea maximizar la función objetivo. Otra clave es encontrar un buen orden de recorrido, así las buenas soluciones serán alcanzadas más rápido, en tal caso los regresos se realizan pronto, sin profundizar mucho en el árbol.

6.2 Algoritmos de Aproximación con Desempeño Garantizado.

En esta sección describiremos algoritmos de aproximación para 3 problemas NP-completos: Cubierta de vértices, Bin Packing y Agente viajero, versión euclídea. Tales algoritmos tienen un desempeño garantizado, es decir, podemos probar que la solución que generan no está muy alejada de la solución óptima.

Problema de la Cubierta de Vértices

Empezamos con un algoritmo de aproximación simple para encontrar la menor cubierta de vértices para una gráfica dada. Este algoritmo garantiza encontrar una cubierta que contenga no más del doble de vértices de la cubierta óptima.

Sea $G = (V, A)$ una gráfica y sea M un apareamiento maximal para G . Ya que M es un apareamiento, sus aristas no tienen vértices en común y como M es maximal, todas las otras aristas tienen un vértice en común con al menos una de las aristas en M .

Teorema 8. El Conjunto de vértices incidentes a las aristas de un apareamiento maximal M es una cubierta de vértices con no más de dos veces el número de vértices de una cubierta mínima de vértices.

Demostración.

El conjunto de vértices que pertenecen a M forma una cubierta de vértices, ya que M es maximal. Cada vértice en la cubierta debe cubrir todas las aristas –en particular, las aristas de M . Pero como M es un apareamiento, un vértice de M no puede cubrir más que una arista de M . Por lo tanto, al menos la mitad de los vértices de M deben pertenecer a la cubierta de vértices. $\diamond \diamond \diamond$



Es posible encontrar una cubierta de vértices maximal, simplemente coleccionando aristas hasta que todas las aristas queden cubiertas. Ya que la cubierta de vértices incluye todos los vértices en el apareamiento, nos gustaría encontrar el apareamiento maximal más pequeño. Desafortunadamente, este problema (encontrar el apareamiento maximal más pequeño) también es NP-Completo.

Problema Bin Packing

El problema del BinPacking se refiere a empacar objetos de diferentes tamaños en cajas o recipientes (*bins*) de tamaño fijo, usando el menor número posible de recipientes. Por ejemplo, si queremos hacer la mudanza de una casa usando pocos carros, o el mismo carro pocas veces, tratando de llenar los carros tanto como sea posible. Una mudanza es un problema de tres dimensiones, nos concentraremos en la versión de una dimensión. Supongamos, por simplicidad, que todas cajas tienen tamaño 1.

El Problema. Sea $\{x_1, x_2, \dots, x_n\}$ un conjunto de n números reales cada uno entre 0 y 1. Partitionar los números en el menor número de subconjuntos como sea posible, de tal forma que la suma de los números en cada subconjunto sea a lo más 1.

Este problema tiene aplicaciones en el problema de manejo de memoria, en donde se tienen requerimientos para varios bloques de diferentes tamaños de memoria y los bloques necesitan ser asignados de diversas partes de memoria disponible.

El problema *Bin Packing* es un problema NP-completo. Una heurística para este problema es poner x_i en la primera caja, y entonces, para cada i , poner x_i en la primera caja que tenga lugar para él, o empezar a llenar una nueva caja si no hay lugar disponible para él en alguna de las cajas usadas. Este algoritmo es llamado Algoritmo Fija en el Primero, *first fit*. No resulta ser tan malo para el peor de los casos, como lo muestra el siguiente resultado.

Teorema 9. El Algoritmo Fija en el Primero requiere a lo más $2*opt$ cajas, donde opt es el mínimo número de cajas en la solución óptima.

Demostración.

El algoritmo no deja dos cajas con menos de la mitad del contenido, ya que el elemento en la segunda caja podría ser puesto en la primera. Por lo tanto, el número de cajas usadas no es mayor al doble de la suma de los tamaños de todos los elementos. El teorema se cumple por el hecho de que el número de cajas en la mejor solución no puede ser menor que la suma de todos los tamaños, en tal caso todos los elementos están perfectamente almacenados. $\diamond\diamond\diamond$

La cota dada por el Teorema 9 es un tanto conservadora. La constante 2 se puede reducir a 1.7 en un análisis más complicado. La constante 1.7 es justa, ya que existen casos \square en los cuales el algoritmo requiere 1.7 veces el óptimo.

Para mejorar al Algoritmo se requiere la siguiente modificación, la cual es muy simple. El peor caso ocurre cuando muchos números pequeños aparecen al principio. En vez de ponerlos en las cajas en el orden en que éstos aparecen, primero los ordenamos en orden decreciente y después usamos el algoritmo *first-fit*. Esta modificación es llamada fija en el primero decrecientemente, *decreasing first fit*. En el peor de los casos, su solución alcanza una constante de 1.22 del óptimo, además tal constante resulta justa. A continuación se enuncia el teorema que garantiza esta cota.

Teorema 10. El Algoritmo Fija en el Primero Decrecientemente requiere a lo más

$(11/9)*opt + 4$ cajas, donde opt es el mínimo número de cajas en la solución óptima.

◊ ◊ ◊

Estos algoritmos con la estrategia *first fit* son heurísticas simples. Hay otros métodos que llevan a mejores constantes. En la mayoría de los casos el análisis resulta ser complicado.

Problema Agente viajero, versión euclídea

El problema del agente viajero es muy importante y tiene muchas aplicaciones. Discutiremos una variante de él, la cual tiene una restricción adicional donde los pesos corresponden a distancias euclídeanas.

El Problema. Sea $\{C_1, C_2, \dots, C_n\}$ un conjunto de n puntos en el plano correspondientes a la localización de n ciudades. Encontrar el ciclo hamiltoniano de distancia mínima, el tour del agente viajero, entre ellos.

Este problema continua siendo NP-difícil, pero veremos que considerar las distancias euclídeanas nos ayuda en el diseño de un algoritmo de aproximación. Podemos relajar el supuesto, suponiendo que las distancias satisfacen la **desigualdad del triángulo**, la cual establece que la distancia directa entre cualesquiera dos puntos es más corta que cualquier ruta a lo largo de otros puntos. En particular, si $d(u,v)$ corresponde a la distancia entre las ciudades u y v , entonces para cualquier otro vértice (ciudad) y se tiene que: $d(u, v) \leq d(u, y) + d(y, v)$.

El algoritmo inicia calculando un árbol generador de peso mínimo, donde el costo es igual a la distancia. Para el problema del árbol generador de peso mínimo tenemos algoritmos polinomiales.

Afirmamos que el costo del árbol de peso mínimo no es mayor que el mejor tour para el problema del agente viajero. Esto es porque el mejor tour contiene un ciclo con todos los vértices, por lo tanto, quitar una arista del tour produce un árbol generador, cuyo costo debe ser mayor o igual al costo del árbol generador de peso mínimo.

Un árbol generador, sin embargo, no corresponde directamente a un tour del agente viajero. Necesitamos modificarlo. Primero consideremos el circuito que consiste de un recorrido DFS, iniciando en cualquier ciudad, además incluimos una arista en la dirección opuesta cuando la búsqueda haga un regreso. Cada arista será recorrida exactamente dos veces, así que el costo de este circuito es dos veces el costo del árbol generador de peso mínimo, el cual no es dos veces mayor que el mínimo tour para el agente viajero. Podemos ahora convertir este circuito en un tour para el agente viajero tomando rutas directas en vez de usar los regresos. La Figura 16 ilustra un ejemplo. Es decir, en vez de hacer un regreso (backtracking) usamos la misma arista y vamos directamente al primer nuevo vértice. El supuesto de que las distancias son euclídeanas es importante, ya que garantiza que la ruta directa entre dos ciudades es, siempre, al menos tan buena como una ruta no directa. La longitud del tour resultante es por lo tanto no mayor a dos veces la longitud del tour óptimo, aunque es frecuentemente menor.

A continuación damos una versión un poco más simple del mismo algoritmo. Primero obtenemos el árbol generador de peso mínimo para la gráfica dada, podemos usar Prim o Kruskal, después recorremos dos veces el árbol convirtiéndolo así en una trayectoria que visita cada ciudad.

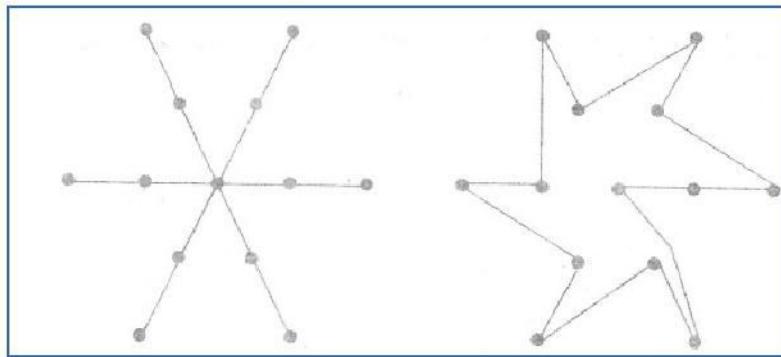


Figura 16. Ejemplo para el problema del Agente Viajero.

Esto es ilustrado en la Figura 17. Como se puede observar estamos visitando los vértices más de una vez. Podemos convertir tal trayectoria en una que no haga eso, tomando "atajos." Esto es, recorremos la gráfica empezando en cualquier vértice y visitando cada vértice no visitado en la secuencia. Cuando haya más de un vértice no visitado adyacente al vértice actual (el que estamos revisando) en el árbol, simplemente visitamos al más cercano. Si sólo los vértices adyacentes al vértice actual han sido visitados, los desviamos tomando un atajo al siguiente vértice no visitado. La desigualdad del triángulo garantiza que los atajos no incrementaran la longitud de la trayectoria. La Figura 17(d) muestra el tour obtenido usando esta estrategia. Este algoritmo queda esquematizado en la Figura 18.

El tour obtenido no es necesariamente el óptimo, sin embargo, el siguiente resultado nos da una garantía de que tan cercano está del óptimo.

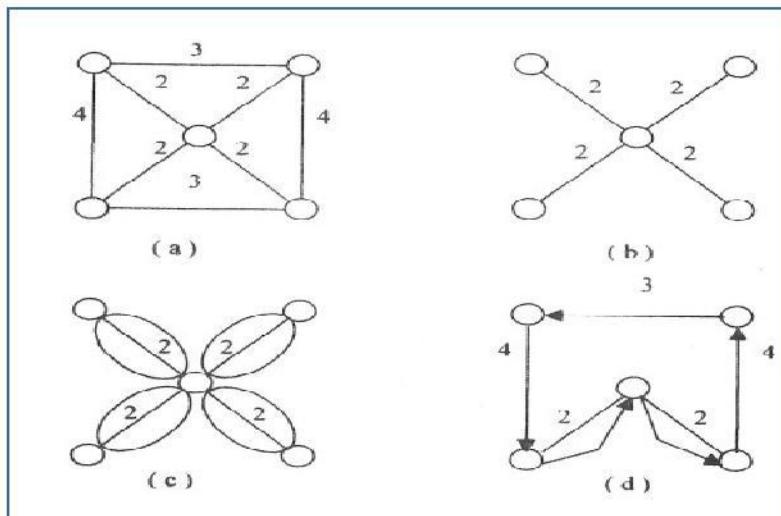


Figura 17. Tour para el Agente viajero usando atajos.

Teorema 11. Sea $mdist$ el peso total de un tour óptimo y $maprox$ el peso total del tour obtenido usando el método descrito. Entonces, $maprox < 2 * mdist$. $\diamond \diamond \diamond$

Algoritmo del Agente Viajero**Inicio**

1. Obtener el árbol generador de peso mínimo, T .
2. Crear una trayectoria que visite cada vértice, recorriendo cada arista del árbol exactamente dos veces.
3. Crear una trayectoria tomando atajos que no visite dos veces ningún vértice, esto es un tour.

Fin

Figura 18. Algoritmo de Aproximación para el Agente Viajero, versión 1.

Una Mejor Aproximación.

Es posible construir un mejor algoritmo de aproximación para el problema del agente viajero. Primero obtenemos un árbol generador de peso mínimo, con un algoritmo clásico. Entonces consideramos el conjunto V' de todos los vértices con grado impar. No es difícil mostrar que debe haber un número par de estos vértices. Creamos parejas entre estos vértices, es decir construimos un apareamiento para V' . Agregamos la arista que conecta cada pareja de vértice al árbol generador.

Ya que cada vértice tiene un número par de aristas que inciden en él, la trayectoria resultante visita cada ciudad. Además, el número total de aristas en la trayectoria no es mayor que el número total de aristas que obtenemos si visitamos dos veces el árbol de expansión. La Figura 19(a) un árbol generador de peso mínimo, la Figura 19(b) muestra el apareamiento de los vértices impares y la Figura 19(c) muestra el tour obtenido usando atajos.

Esta mejora del algoritmo de aproximación queda especificado en la Figura 20. El Teorema 12 establece que este algoritmo da una mejor cota que el algoritmo anterior.

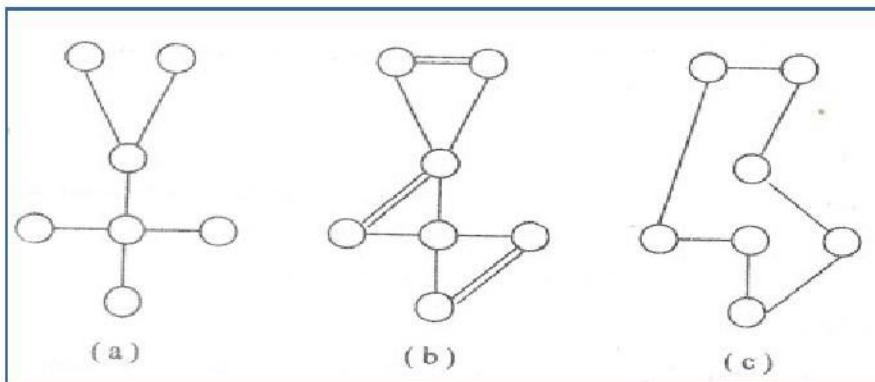


Figura 19. Tour obtenido usando apareamientos.

Teorema 12. Sea $mdist$ el peso total de un tour óptimo y $minapprox$ el peso total del tour obtenido usando el método descrito con apareamiento. Entonces,

$$minapprox < (1.5) * mdist.$$

Demostración.

Sea T el árbol generador de peso mínimo. Sea V' el conjunto de vértices de grado impar. Convierte en tour una trayectoria que conecte sólo vértices de V' por una trayectoria que no pase por vértices de V' . Por la desigualdad del triángulo tal

trayectoria no puede ser mayor que ***mdist***. Además esta trayectoria, nos proporciona dos apareamientos para ***V'***, los cuales se construyen de la siguiente manera: Primero se elige un vértice cualquiera de ***V'*** y se aparea con el vértice de algún lado de esta trayectoria; después se continua apareando vértices adyacentes en la misma dirección. Este es un apareamiento. El segundo apareamiento se forma tomando un vértice cualquiera en ***V'*** y se aparea con un vértice del otro lado de la trayectoria y se continua seleccionando vértices en la otra dirección.

Como las aristas de los dos apareamientos abarcan todas las aristas de la trayectoria, la suma total de los pasos de los dos apareamientos es igual al peso de la trayectoria. Por lo tanto, al menos uno de los apareamientos tiene peso total no mayor a la mitad del peso de la ruta. Como el peso de la trayectoria no es mayor que ***mdist*** y como el peso de cualquier apareamiento es al menos tan grande como el peso de un apareamiento de peso mínimo, se tiene que: Si ***M*** es el apareamiento de peso mínimo entonces ***w(M) ≤ (0.5) *mdist***, donde ***w(M)*** representa el peso del apareamiento ***M***.

Ahora bien, el peso del árbol ***T***, ***w(T)***, es menor que ***mdist***, es decir, ***w(T) < mdist***. Ya que las aristas obtenidas del Paso 3, del algoritmo de la Figura 20, fueron aristas de un árbol generador de peso mínimo y de las aristas del apareamiento de peso mínimo, el peso total de estas aristas es menor que: ***(1.5)* mdist***.

El teorema se cumple, ya que el peso total de las aristas en el tour final, obtenido en el Paso 4 no es mayor que el peso que se había obtenido en el Paso 3. ◊ ◊ ◊

Algoritmo del Agente Viajero Mejorado

Inicio

1. Obtener el árbol generador de peso mínimo, ***T***.
2. Sea ***V'*** el conjunto de vértices de grado impar de ***T***
3. Aparear los vértices de ***V'*** de la manera más barata posible.
4. Añadir las aristas del apareamiento al árbol ***T***.
5. Ahora cada vértice tiene grado par, es posible encontrar un circuito euleriano. Basta con una trayectoria euleriana que pase por dada arista del apareamiento.
6. De la ruta euleriana tomamos el camino corto, es decir, un atajo.

Fin

Figura 20. Mejoramiento del algoritmo de aproximación

Bibliografía.

- [1] M.R. Garey & D.S. Johnson
Computer and Intractability, A Guide to the Theory of NP-Completeness
WH. Freeman, USA, 1979
- [2] J. Kingston. ***Algorithms and Data Structures: Design, Correctness, and Analysis***,
Addison Wesley, Pu. Co., USA 1990.
- [3] U. Manber. ***Introduction to Algorithms. A Creative Approach***.
Addison Wesley, Pu. Co., USA 1989.
- [4] R. Neapolitan & K. Naimipour. ***Foundations of Algorithms***,
DC Heath and Co., USA, 1996