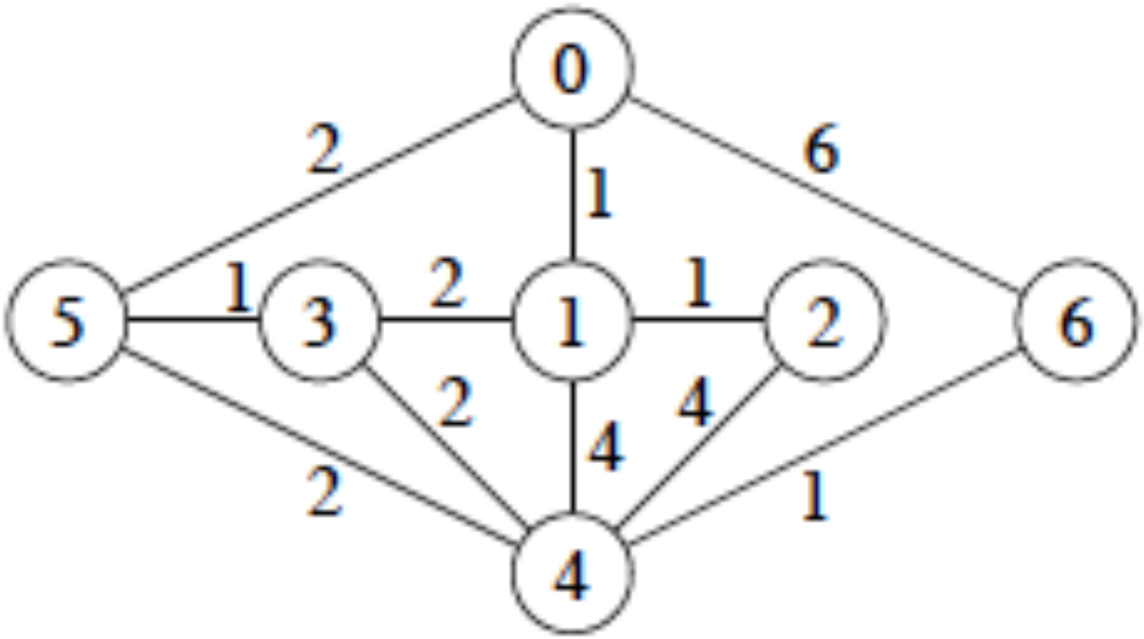
Algoritmos Aproximados

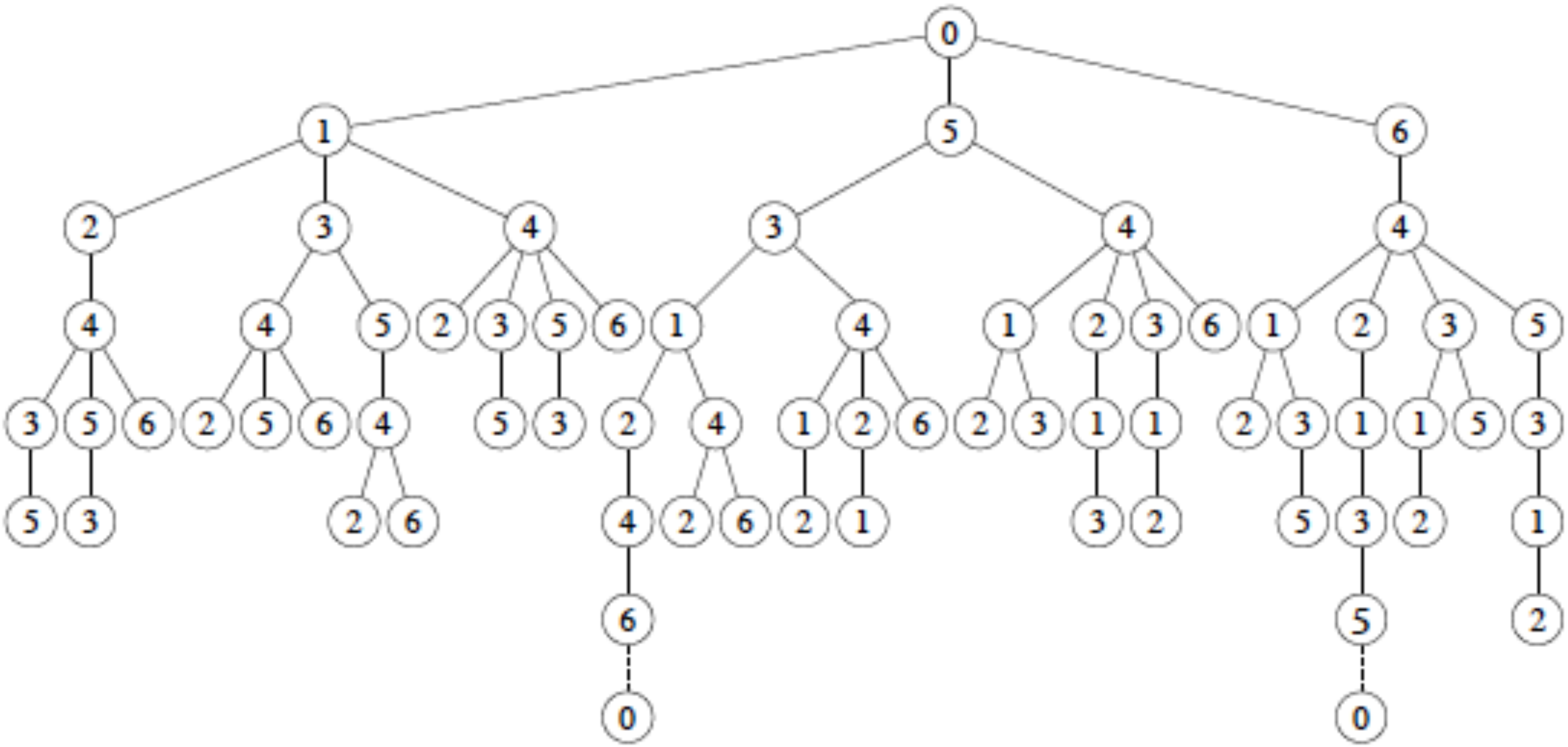
Problemas Exponenciais – O que fazer?

- Tamanho da entrada pequeno?
  - Algoritmo exponencial pode ser razoável
- Casos especiais polinomiais?
- Concentrar no caso médio: existem algoritmos que funcionam bem na prática
  - Simplex de programação linear
- Tentativa e erro com podas

Exemplo de Podas: Ciclo de Hamilton



Exemplo de Podas: Ciclo de Hamilton

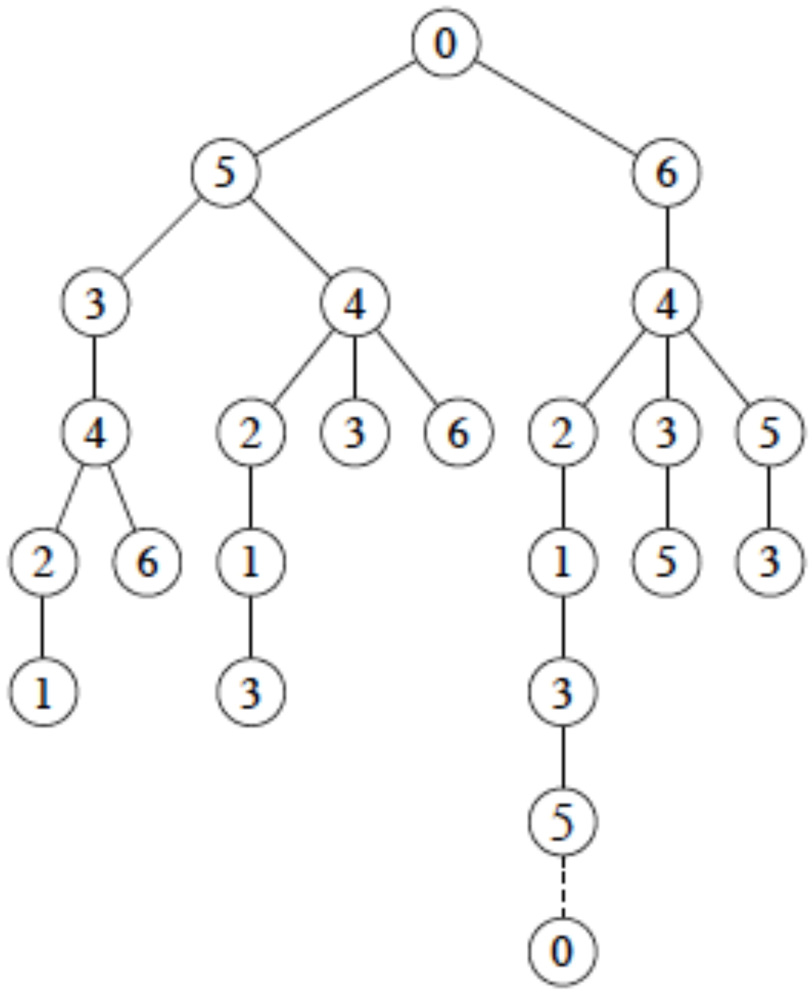


Exemplo de Podas: Ciclo de Hamilton

- Diminuir o número de passos do algoritmo (*branches* da árvore) impondo ordem relativa entre dois nós
- Explora simetria inerente ao problema
  - Ciclo aparece duas vezes na árvore

- Podemos impor, e.g., que 2 apareça antes do 1

Exemplo de Podas: Ciclo de Hamilton



Tentativa e erro com podas

- Outra estratégia é **branch-and-bound**
- A idéia é interromper a procura por um caminho tão logo se saiba que ele não levará à solução ótima
- EX: seja o problema do PCV
- Suponha que já se tenha um ciclo de tamanho k
- Podemos interromper qualquer caminho que já tenha um custo pelo menos k e não tenha um ciclo inteiro.

Heurísticas e
Algoritmos Aproximados

- Abre mão da otimalidade da solução pela eficiência no tempo de execução

- Preocupação: qualidade da solução
  - Heurística: nenhum controle
  - Algoritmo aproximado : fornece limite para quão ruim a solução pode ser, em relação ao ótimo

Algoritmos Aproximados

- Um algoritmo aproximado tem uma razão de aproximação R(n) se, para qualquer entrada de tamanho n, tem-se que:

$$\frac{S}{S^*} \leq R(n)$$ Problemas de minimização

onde S é a solução produzida pelo algoritmo aproximado e S* a solução ótima

Para problemas de maximização, a razão é invertida

Algoritmos Aproximados

- Para vários problemas, existem algoritmos aproximados de tempo polinomial com razão de aproximação constante (pequena)
  - PCV, Cobertura de vértices
- Para outros: os algoritmos têm razão de aproximação que cresce como uma função do tamanho da entrada

Problema da Cobertura de Vértices

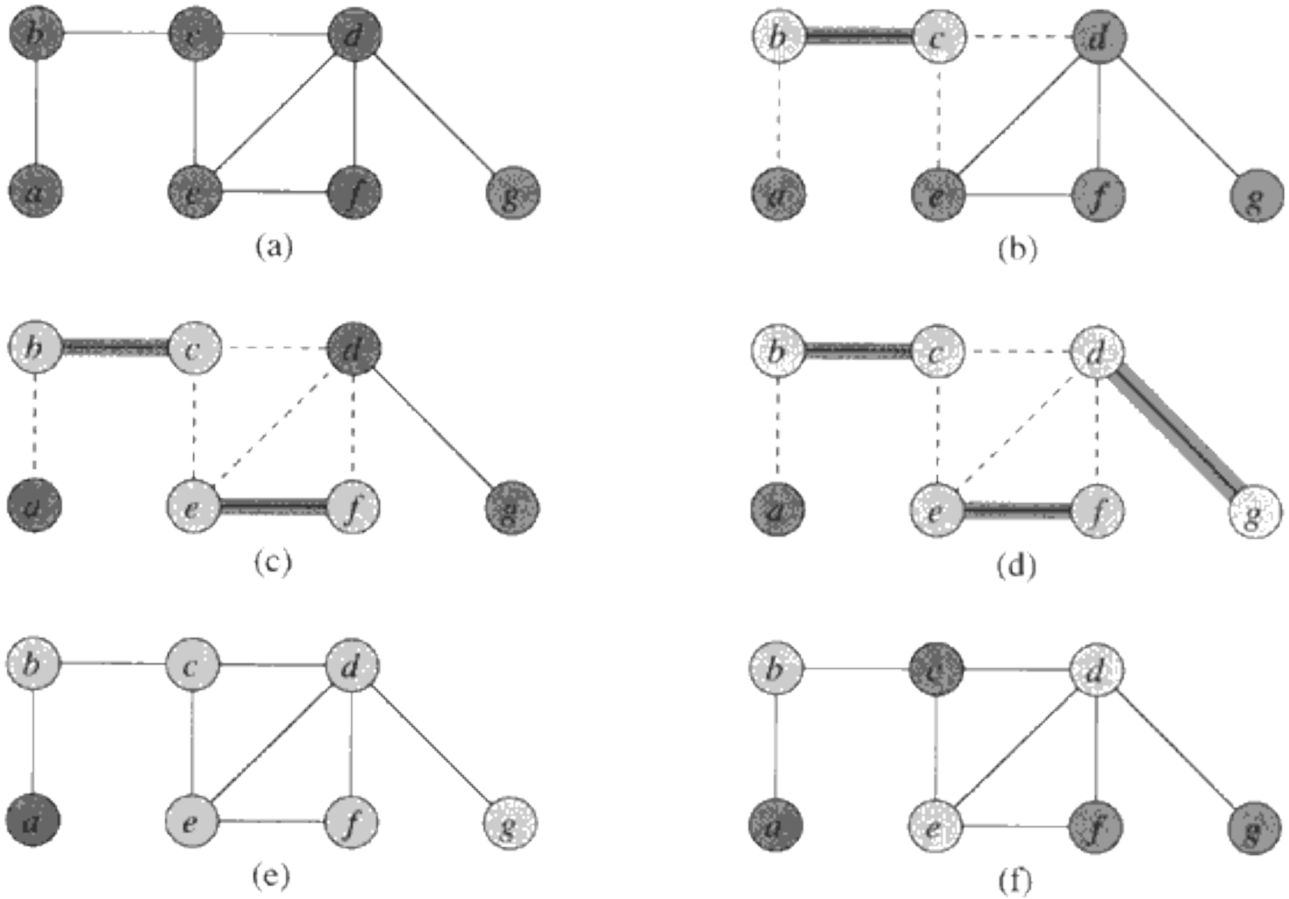
- Problema consiste em encontrar uma cobertura de vértices de tamanho mínimo em um dado grafo não orientado.

- Uma cobertura de vértices de um grafo não direcionado G(V,E) é um subconjunto V’ de V tal que, se (u,v) é aresta em G, ou u ou v ou ambos pertencem a V’

Algoritmo Aproximado para a Cobertura de Vértices

INPUT G   // grafo não direcionado
$S \leftarrow \{\ \}$
$E' \leftarrow E[G]$
while E' is not empty **do**
    Let (u,v) be an arbitrary edge of E'
    $S \leftarrow S \cup \{u, v\}$
    Remove from E' every edge incident on either u or v
return S

Algoritmo Aproximado para a Cobertura de Vértices: Execução



Solução produzida pelo algoritmo

Solução ótima

Algoritmo Aproximado para a Cobertura de Vértices

- O algoritmo apresentado é um algoritmo aproximado de tempo polinomial com razão de aproximação R(n) = 2 para o problema da cobertura de vértices

- O algoritmo produz uma cobertura de vértices (checar!)

- Complexidade: polinomial : O(|V| + |E|) usando lista de adjacências

- Qualidade da solução?

Algoritmo Aproximado para a Cobertura de Vértices

- Seja A o conjunto de arestas escolhidas pelo algoritmo (linha 5)
- Duas arestas de A não compartilham extremidade, logo não há duas arestas em A que podem ser cobertas pelo mesmo vértice em S*.
- Logo:

$$|S^*| \geq |A|$$

A é limite inferior para o tamanho da cobertura de vértices ótima

Algoritmo Aproximado para a Cobertura de Vértices

- Mas temos que:

$$|S| = 2|A|$$

- O que nos leva a:

$$|S| = 2|A| \leq 2|S^*|$$

$$\frac{S}{S^*} \leq 2$$

Problema do Caixeiro Viajante

- Dado um grafo G(V,A) não direcionado, completo ponderado, cujos pesos das arestas d(i,j) satisfaçam as seguintes restrições:
  - $d(i,j) = d(j,i) \quad \forall\ i,j \in V$
  - $d(i,j) >= 0 \quad \forall\ i,j \in V$
  - $d(i,j) <= d(i,k) + d(k,j)\ \forall\ i,j,k \in V$

    (desigualdade triangular)

Limite Inferior para o PCV

- AGM é um limite inferior para a solução ótima do PCV em um grafo G(V,A)

$$S^* \geq AGM$$

- Algoritmo de Prim para obter AGM: O(V^2)

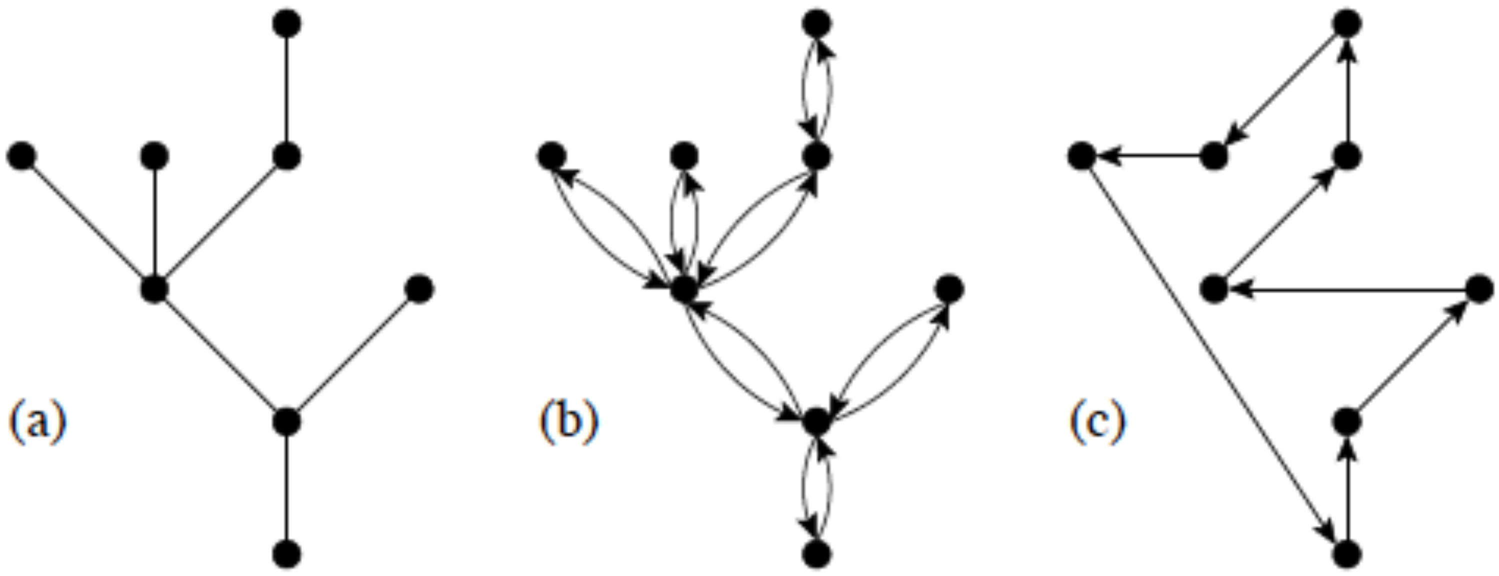
Algoritmo Aproximado para o PCV

Approx-TSP-Tour(G, c)

1. select root vertex r in V[G]
2. T = MST-Prim(G, c, r) // computes a MST
3. L = list of vertices in preorder traversal of T

   (Busca em profundidade em T)
4. **return** hamiltonian cycle with vertices ordered as in L

Algoritmo Aproximado para o PCV



Algoritmo Aproximado para PCV

- O algoritmo apresentado é algoritmo de aproximação 2 com tempo polinomial para o PCV com desigualdade triangular
- O algoritmo produz solução para o PCV (checar!)
- O algoritmo tem tempo polinomial (O(V^2))
- Razão de aproximação:

$$S \leq 2AGM \leq 2S^*$$

$$\frac{S}{S^*} \leq 2$$

Algoritmo Aproximado para PCV: Algoritmo de Christophides

- Como melhorar a razão de aproximação?
- Algoritmo de Christophides utiliza conceito de **grafo euleriano**
- **Grafo Euleriano:** todo vértice tem grau par
- **Circuito Euleriano:** circuito que passa por todas as arestas exatamente uma vez

Algoritmo Aproximado Revisitado

- O algoritmo com razão de aproximação 2 pode ser reescrito como:
  1. Construa uma AGM que tenha cidades do PCV como vértices.
  2. Dobre suas arestas para obter um grafo Euleriano.
  3. Encontre um caminho Euleriano para esse grafo, usando busca em profundidade
  4. Converta-o em um caminho do caixeiro viajante (pre-ordem).
- Algoritmo de Christophides altera passo 2

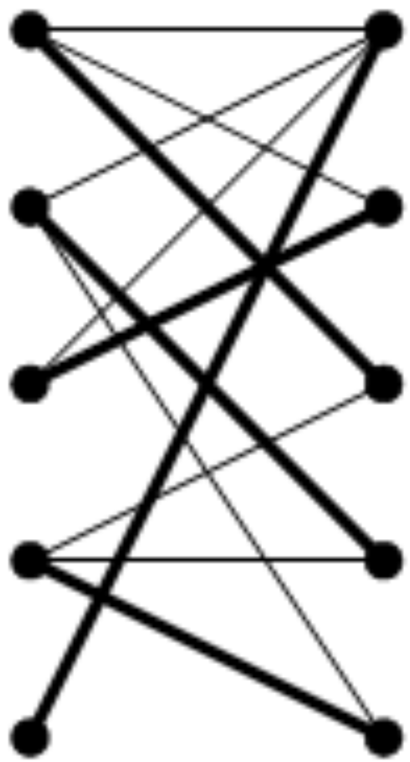
Algoritmo de Christophides para o PCV

- Premissa: não precisa dobrar as arestas da AGM para obter um grafo euleriano

- Todo grafo (e AGM em particular) tem um número par de vértices de grau impar
  - Apenas precisa aumentar o grau destes vértices (grau+1)

Casamento Mínimo com Pesos

- Dado um conjunto contendo um número par de cidades, um casamento é uma coleção de arestas M tal que cada cidade é a extremidade de exatamente um arco em M.



Um casamento mínimo é aquele para o qual o comprimento total das arestas é mínimo.

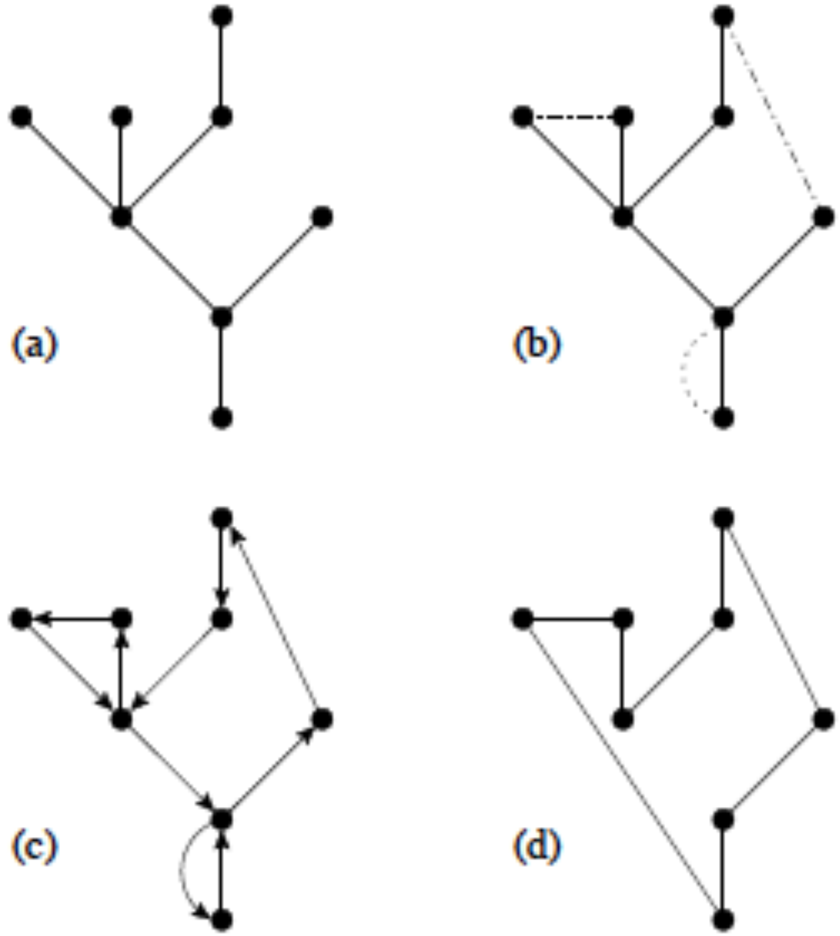
Todo vértice é parte de exatamente uma aresta do conjunto M.

Pode ser encontrado com custo O(n^3).

Algoritmo Aproximado de Christophides

1. Construa uma AGM que tenha cidades do PCV como vértices.
2. Adicione à AGM o casamento mínimo com pesos dos vértices de grau impar
3. Encontre um caminho Euleriano para o grafo resultante.
4. Converta-o em um caminho do caixeiro viajante usando curto-circuitos.

Algoritmo Aproximado de Christophides



Algoritmo Aproximado de Christophides

- O algoritmo apresentado é algoritmo de aproximação 1.5 com tempo polinomial para o PCV com desigualdade triangular

- O algoritmo produz solução para o PCV (checar!)

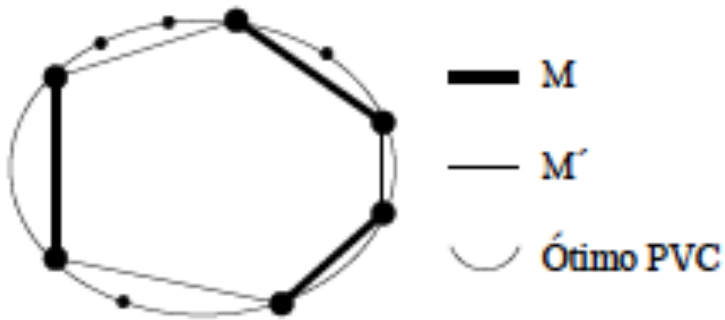
- O algoritmo tem tempo polinomial (O(V^3))

- Razão de aproximação?

Razão de Aproximação do Algoritmo de Christophides

- Seja M_{min} o custo total das arestas adicionadas pelo casamento mínimo
- Logo: $$S = AGM + M_{\min}$$
- Sejam M e M' dois casamentos tais que:

$$M + M' \leq S^*$$



Razão de Aproximação do Algoritmo de Christophides

$$Se: M + M' \leq S^*$$

$$Então \text{ ou M } \leq \frac{S^*}{2} \text{ ou M' } \leq \frac{S^*}{2}$$

$$Logo: \min(M, M') \leq \frac{S^*}{2} \qquad M_{\min} \leq \min(M, M') \leq \frac{S^*}{2}$$

$$S = AGM + M_{\min} \leq S^* + \frac{S^*}{2}$$

$$\frac{S}{S^*} \leq \frac{3}{2}$$

Problema da Cobertura de Conjuntos

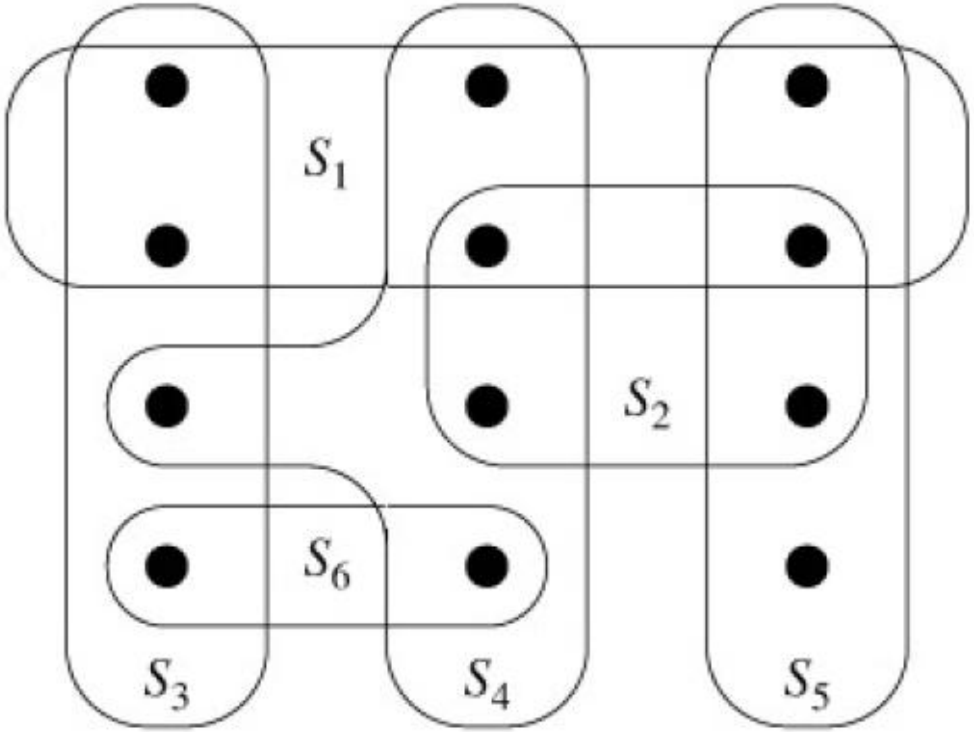
- Dados um conjunto finito X e uma família F de subconjuntos de X, tal que todo elemento de X pertence a pelo menos um subconjunto de F:

$$X = \bigcup_{S \in F} S$$

- O problema da cobertura de conjuntos consiste em encontrar um subconjunto de tamanho mínimo C de F cujos membros “cubram” todos os elementos de X:

$$X = \bigcup_{S \in C} S$$

Problema da Cobertura de Conjuntos



$$F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$$

$$C^* = \{S_3, S_4, S_5\}$$

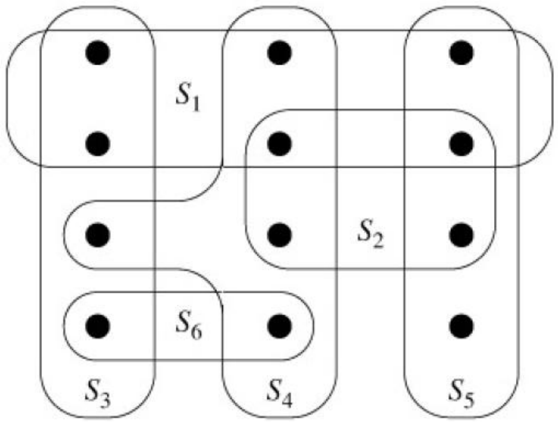
Exemplo de Aplicação

- X = conjunto de habilidades necessárias para desenvolver um certo projeto
- F = conjunto de pessoas disponíveis, sendo que cada pessoa tem um conjunto de habilidades
  – Mais de uma pessoa pode ter uma mesma habilidade
- Busca-se uma comissão com número mínimo de pessoas tal que exista pelo menos uma pessoa com cada habilidade necessária para o projeto nesta comissão (todas as habilidades são *cobertas*)
  – Versão de decisão: é possível criar uma comissão com até k pessoas tal que todas as habilidades estejam cobertas?
  – Versão de decisão: NP-Completo

Algoritmo de Aproximação Guloso para a Cobertura de Conjuntos

GREEDY-SET-COVER(X,F)

1. $U = X$
2. $C = \varnothing$
3. **while** $U \neq \varnothing$ **do**
    4. select $S \in F$ that maximizes $| S \cap U |$
    5. $U = U - S$
    6. $C = C \cup \{S\}$
7. **return** C



$$C = \{S_1, S_4, S_5, S_3\}$$

Complexidade: O(|S|*|F|)min(|X|,|F|)

Escolhe em cada fase o subconjunto restante que cobre o maior número de elementos ainda descobertos

Algoritmo de Aproximação Guloso para a Cobertura de Conjuntos

- A razão de aproximação do algoritmo GREEDY_SET_COVER é R(n) igual a ln|X|+1 (cresce com a entrada, mas devagar)

- Lembrar que: $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ d - esimo número harmônico

$$\mathrm{H}(0) = 0$$

$$\sum_{i=1}^{n} \frac{1}{i} \leq \ln(\mathrm{n}) + 1$$

Razão de Aproximação do Algoritmo Greedy_Set_Cover

- Idéia geral:
  - Atribua um custo 1 a cada conjunto selecionado no passo 4
  - Distribua este custo uniformemente entre todos os elementos deste conjunto **sendo cobertos pela primeira vez**.
  - Isto é apenas um **artifício** para derivar razão de aproximação
- Sejam:
  - S_i o i-esimo conjunto selecionado no passo 4
  - c_x o custo alocado ao elemento x para cada $x \in X$ (depende de quando ele foi coberto pela primeira vez)

Razão de Aproximação do Algoritmo Greedy_Set_Cover

- Se x é coberto pela primeira vez por S_i:

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

- Cada conjunto selecionado recebe um custo 1, e cada elemento recebe a atribuição de um custo apenas, logo:

$$|C| = \sum_{x \in X} c_x$$

C = conjunto solução produzido pelo algoritmo

Razão de Aproximação do Algoritmo Greedy_Set_Cover

- O custo atribuído à solução ótima C* é dado por:

$$\sum_{S \in C^*} \sum_{x \in S} c_x$$

- Como um elemento pode aparecer em mais de um conjunto de C*:

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x = |C|$$

Razão de Aproximação do Algoritmo Greedy_Set_Cover

- Então: $|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x$
- Assumindo que: $\sum_{x \in S} c_x \leq H(|S|)$
- Temos que:

$$|C| \leq \sum_{S \in C^*} H(|S|)$$

$$|C| \leq |C^*| \times H(\max\{|S| : S \in F\})$$

$$\frac{|C|}{|C^*|} \leq H(\max |S|) \leq H(|X|) \leq \ln |X| + 1$$

Razão de Aproximação do Algoritmo Greedy_Set_Cover

- Resta provar que:

$$\sum_{x \in S} c_x \leq H(|S|)$$

Derivação

- Considere qualquer conjunto S $\in$ F e prove que:

$$\sum_{x \in S} c_x \leq H(|S|)$$

Seja o número de elementos em S que permanecem **descobertos** depois que S_1, S_2, ... S_i foram selecionados pelo algoritmo:

$$u_i = |S - (S_1 \cup S_2 \cup \ldots \cup S_i)|$$

$$u_0 = |S| \qquad u_k = 0 \quad \mathrm{S_k} \text{ último conjunto selecionado}$$

$$\mathrm{u_{i\text{-}1}} \geq u_i$$

$$u_{i-1} - u_i = \# \text{ elementos de S cobertos pela 1a vez por } \mathrm{S_i}$$

Derivação

- Logo:

Custo associado a cada elemento coberto por S_i

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

elementos cobertos por S_i

Derivação

- Logo:

Custo associado a cada elemento coberto por S_i

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

elementos cobertos por S_i

- Note que:

$$|S_i - (S_1 \cup S_2 \cup \ldots S_{i-1})| \geq |S - (S_1 \cup S_2 \cup \ldots S_{i-1})| = u_{i-1}$$

porque a escolha gulosa de S_i garante que S não pode cobrir mais elementos novos que S_i

Derivação

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

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

$$\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \quad (\text{porque } \mathrm{j} \leq \mathrm{u}_{\mathrm{i-1}})$$

$$= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right) = \sum_{i=1}^{k} \left(H(u_{i-1}) - H(u_i) \right)$$

$$= \mathrm{H}(\mathrm{u}_0) - H(u_k) = H(u_0) - H(0) = H(u_0) = H(|S|)$$

Esquema de aproximação

- É um algoritmo de aproximação que:
  - toma como entrada uma instância do problema
  - e um valor $\varepsilon > 0$
- tal que para um ε fixo
  - o esquema é um algoritmo de aproximação com razão $(1+ \varepsilon)$
- Um **esquema de aproximação de tempo polinomial** (PTAS) é um esquema que
  - para qualquer $\varepsilon>0$ fixo o esquema executa em tempo polinomial em função do tamanho n da entrada.
  - Tempo pode variar para diferentes valores de ε

Esquema de aproximação

- O tempo de execução de um *esquema de aproximação em tempo polinomial* pode crescer tão rápido quanto *ε* decresce.
  - Ex: $O(n^{2/\varepsilon})$

- O ideal é que o tempo seja polinomial tanto em 1/*ε* quanto em n.
  – Compromisso entre qualidade da solução (*ε)* e tempo de execução

- Em um **esquema de aproximação de tempo completamente polinomial** (FPTAS) seu tempo de execução são polinomiais em 1/*ε* e n.
  - Ex: $O((1/\varepsilon)^2 n^3)$

Problema da Soma de SubConjuntos

- Dados S um conjunto de inteiros positivos $\{x_1,x_2,\ldots,x_n\}$ e um inteiro positivo t, existe um subconjunto de S cujos elementos somam exatamente t?
  - NP – Completo
- Problema de otimização associado: encontre o subconjunto de S tal que a soma de seus elementos seja tão grande quanto possível, mas não maior que t

Ex: temos um caminhão que não pode transportar mais que t kg, e n caixas diferentes para transportar, das quais a i-esima caixa pesa x_i.

Desejamos encher o caminhão com a carga mais pesada possível, sem exceder o limite de peso dado

Algoritmo Exponencial

L_i = lista das somas de todos os subconjuntos de $\{x_1 \ldots x_i\}$ que não excedem t

EXACT-SUBSET-SUM(S, t)

Soma x_i a todos os elementos da lista L_{i-1}

1. $n \leftarrow |S|$
2. $L_0 \leftarrow \{0\}$
3. **for** $i \leftarrow 1$ **to** n **do**
4. $L_i \leftarrow$ MERGE-LISTS(L_{i-1}, $L_{i-1} + x_i$)
5. remove from L_i every element that is greater than t
6. **return** the largest element in L_n

Complexidade: dominada pelo MERGE-LISTS, que é linear no # elementos

Merge-Lists (L, L') é O(|L| + |L'|)

Como # elementos possíveis é exponencial: complexidade exponencial

Outro Algoritmo

- Pode-se derivar um esquema de aproximação em tempo completamente polinomial
  - Realiza-ze um “trimming” de cada lista após sua criação

- Trimming:
  - Se dois valores em uma lista são próximos o suficiente, para o propósito de encontrar uma solução aproximada, não há necessidade de manter os dois explicitamente

Trimming

- Parâmetro δ definido tal que $0 < \delta < 1$
- Remova tantos elementos de L quanto for possível garantindo que:
  - Seja L' a lista após trimming
  - Para cada elemento y que foi removido de L, ainda existe em L' um elemento z que se aproxime de y tal que:

$$\frac{y}{1+\delta} \leq z \leq y$$

Trimming: Exemplo

- Dados L = {10,11,12,15,20,21,22,23,24,29} e $\delta = 0{,}1$

- Após realizarmos *Trim* em L teremos:

  L' = {10,12,15,20,23,29}

  onde:

  11 está representado por 10
  21 e 22 estão representados por 20
  24 está representado por 23

Trimming

TRIM(L, δ)

$m \leftarrow |L|$

$L' \leftarrow \{y_1\}$

$last \leftarrow y_1$

for $i \leftarrow 2$ **to** m **do**

  if $y_i > last * (1 + \delta)$ // $yi \geq last$ because L is sorted

  then append y_i onto the end of L'

    $last \leftarrow yi$

return L'

Ө(m) onde m e # elementos em L
e assumindo L ordenada

Esquema de Aproximação

APPROX-SUBSET-SUM(S, t,)

$n \leftarrow |S|$
$L_0 \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $L_i \leftarrow$ MERGE-LISTS(L_i-1, L_i-1 + x_i)
    $L_i \leftarrow$ TRIM(L_i, ε /2n)
    remove from L_i every element that is greater than t
let z be the largest value in Ln
return z

- Com 0<ε<1
- Valor retornado está dentro de um fator de 1+ ε da solução ótima

Exemplo

S={104,102,201,101}, t=308, ε=0,40 e $\delta = \varepsilon/8$=0,05
linha 2: $L0$ = {0},

linha 4: $L1$ = {0, 104},
linha 5: $L1$ = {0, 104},
linha 6: $L1$ = {0, 104},

linha 4: $L2$ = {0, 102, 104, 206},
linha 5: $L2$ = {0, 102, 206},
linha 6: $L2$ = {0, 102, 206},

linha 4: $L3$ = {0, 102, 201, 206, 303, 407},
linha 5: $L3$ = {0, 102, 201, 303, 407},
linha 6: $L3$ = {0, 102, 201, 303},

linha 4: $L4$ = {0, 101, 102, 201, 203, 302, 303, 404},
linha 5: $L4$ = {0, 101, 201, 302, 404},
linha 6: $L4$ = {0, 101, 201, 302}

z = 302 que está dentro dos 40% de aproximação da solução ótima (307)

Trimming: Exemplo

•Approx-Subset-Sum é um esquema de aproximação de tempo completamente polinomial

- Razão de aproximação : $1+ \varepsilon$

- Complexidade de tempo: polinomial em n e $1/\varepsilon$