Uma Introdução a Algoritmos de Aproximação

Mário César San Felice
(com materiais da Profa. Carla Negri Lintzmayer do CMCC-UFABC
e do Prof. Flávio Keidi Miyazawa do IC-Unicamp)

Universidade Federal de São Carlos
Departamento de Computação

16 de Julho de 2021

Classes de Complexidade

Classe P: problemas de decisão que possuem algoritmos eficientes.

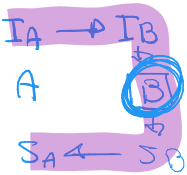
Classe NP: problemas de decisão cuja solução pode ser verificada em tempo polinomial.

Classe NP-Completo: problemas em NP para os quais todo problema em NP é redutível.

Questão P vs. NP: se houver um algoritmo que resolve qualquer problema NP-Completo em tempo polinomial, então todos os problemas em NP também são resolvidos.

Classe NP-Difícil: problemas para os quais todo problema em NP é redutível.

Estamos particularmente interessados em problemas NP-Difíceis.



Abordagens

Se $P \neq NP$, então não podemos ter algoritmos para problemas NP-Difíceis que, simultaneamente,

1. encontrem soluções ótimas
2. em tempo polinomial
3. para qualquer entrada.

Nos resta não exigir todas essas propriedades do mesmo algoritmo!

Algumas abordagens possíveis são

- métodos exatos
- heurísticas
- algoritmos de aproximação
- parametrização

Algoritmos de aproximação

Relaxando a restrição de exigir soluções ótimas, com um detalhe.

Geram soluções viáveis em tempo polinomial e dão garantia no custo da solução gerada com relação ao custo da solução ótima.

Seja A um algoritmo polinomial para um problema de minimização, $A(I)$ o custo da solução de A para a entrada I e $OPT(I)$ o custo da solução ótima.

A é uma α-aproximação se, para toda instância I,

$$A(I) \leq \alpha \, OPT(I) \qquad \alpha \geq 1$$

Se o problema for de maximização, temos a definição

$$A(I) \geq \alpha \, OPT(I) \qquad \alpha \leq 1$$

Problema da Mochila

Mochila Binária

Entrada: conjunto de n itens, cada item i tem peso w_i e valor v_i, e capacidade W da mochila.

Soluções viáveis: conjuntos de itens $S \subseteq \{1, \ldots, n\}$ com $\sum_{i \in S} w_i \leq W$.

Função objetivo: soma dos valores dos itens em S.

Objetivo: encontrar uma solução de valor máximo.

Exemplo para Algoritmo Guloso para a Mochila

O que acontece no seguinte cenário?

$W = B$

razão

- $v_1 = 2$ $w_1 = 1$ $\frac{v_1}{w_1} = 2$
  $v_2 = B$ $w_2 = B$ $\frac{v_2}{w_2} = 1$

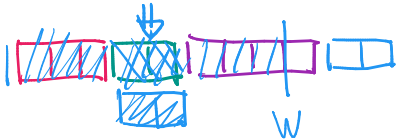
ordem	fração	espaço livre = B
1	100%	B - 1
2	0%	

$ALG = 2 << B = OPT$

Algoritmo de aproximação para a Mochila

1: **função** MochilaAprox(n, w, v, W)
2: ordene e renomeie os itens para que $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$
3: seja q um inteiro tal que $\sum_{i=1}^{q} w_i \leq W$ e $\sum_{i=1}^{q+1} w_i > W$
4: **devolve** $\max\{v_1 + v_2 + \ldots + v_q, v_{q+1}\}$

Análise:



De onde vemos que esse algoritmo é uma $\frac{1}{2}$-aproximação.

Problema do Escalonamento

Escalonamento de Tarefas

Entrada: conjunto de tarefas $\{1, \ldots, n\}$, cada tarefa i tem tempo de processamento t_i, e m máquinas idênticas.

Soluções viáveis: partição das tarefas em m conjuntos $M_1, M_2, \ldots, M_m$.

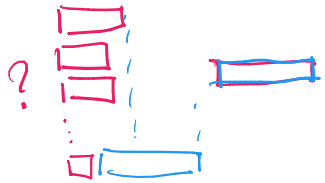
Função objetivo: $\max_{j=1\ldots m} \sum_{i \in M_j} t_i$ (makespan).

Objetivo: encontrar solução de custo mínimo.

Algoritmo de aproximação para o Escalonamento

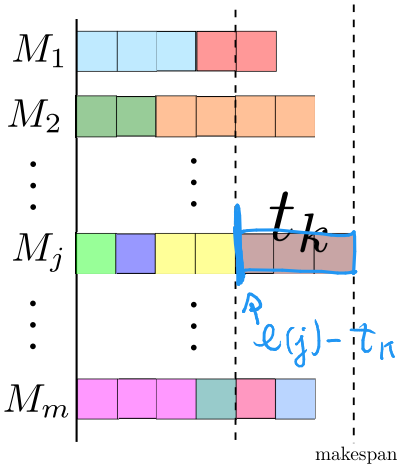
Ideia: alocar cada tarefa à máquina que, no momento, tem a menor soma de tempos de processamento.

1: **função** EscalonaAprox(n, t, m)
2: faça $M_j = \emptyset$ para $j = 1, \ldots, m$
3: **para** $i \leftarrow 1$ até n **faça**
4: seja j uma máquina em que $\sum_{i' \in M_j} t_{i'}$ é mínimo
5: $M_j \leftarrow M_j \cup \{i\}$
6: **devolve** $\max_{j=1,\ldots,m} \sum_{i \in M_j} t_i$



Análise

Seja j a máquina que define o makespan e seja k a última tarefa que foi alocada a essa máquina.

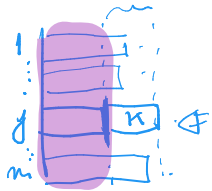


Análise

Seja $l(j) = \sum_{i \in M_j} t_i$ a carga da máquina j.

- Note que, qualquer máquina j' tem carga $l(j') \geq l(j) - t_k$ pela escolha gulosa do algoritmo. Então

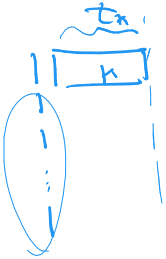
$$(l(j) - t_k) \cdot m \leq \sum_{j'=1}^{m} l(j') = \sum_{i=1}^{n} t_i$$



De onde

$$l(j) - t_k \leq \frac{1}{m} \sum_{j=1}^{m} t_i = OPT \leq OPT(I)$$

Análise



Temos

$$l(j) - t_k \leq OPT(I)$$

Assim

$$\text{Escalona Aprox}(I) = l(j) = (l(j) - t_k) + t_k$$
$$\leq 1 OPT(I) + t_k$$
$$t_k \leq ? OPT$$
$$\leq 2 OPT(I)$$

De onde vemos que esse algoritmo é uma 2-aproximação.

Quiz: se o tempo de processamento de qualquer tarefa for no máximo 10% de $\sum_{i=1}^{n} t_i/m$, você consegue melhorar a garantia obtida?

Problema do Caixeiro Viajante

Caixeiro Viajante (TSP)

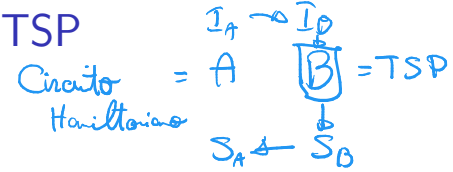
Entrada: $G = (V, E)$ e função w de peso nas arestas.

Soluções viáveis: circuitos hamiltonianos de G.

Função objetivo: soma dos pesos das arestas do circuito.

Objetivo: encontrar um circuito de menor custo.

Inaproximabilidade do TSP



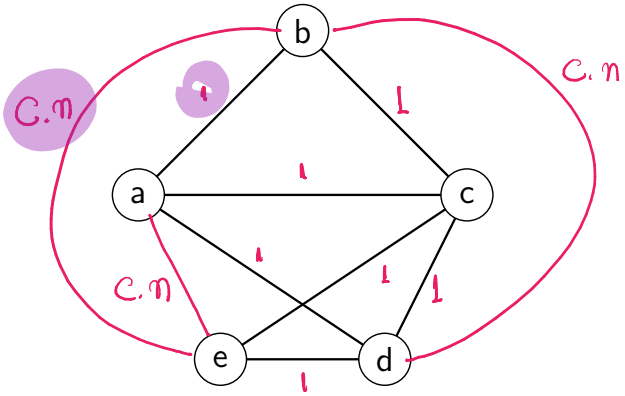
Considere o problema do Circuito Hamiltoniano, que é NP-Completo:

- dado um grafo $G = (V, E)$, existe um circuito que visita cada vértice de G exatamente uma vez?

Lembre que, se P$\neq$NP então não existe algoritmo polinomial para problemas NP-Completos.

Inaproximabilidade do TSP

Vamos construir um grafo completo $G' = (V, E')$ com peso w nas arestas tal que $w(e) = 1$ se $e \in E$ e $w(e) = c \cdot n$ se $e \notin E$, para uma constante c.



n

c - aprox

$A(I) \leq c\, OPT(I)$

$\| \quad n$

Inaproximabilidade do TSP

Relacionando com o TSP, temos duas possibilidades:

1. se G tem um circuito hamiltoniano, então G' tem um circuito hamiltoniano (o mesmo) de custo n;
   - i.e., $OPT(G', w) = n$;
2. se G não tem circuito hamiltoniano, então G' tem (pois é completo), mas ele usa pelo menos uma aresta de custo $c \cdot n$,
   - i.e., $OPT(G', w) > c \cdot n$.

Se existir um algoritmo A que é uma c-aproximação para o TSP, então:

① $A(G', w) \leq c\, OPT(G', w) = cn$

② $A(G', w) \geq OPT(G', w) > c \cdot n$

Inaproximabilidade do TSP

Se existir um algoritmo A que é uma c-aproximação para o TSP, então:

1. $A(G', w) \leq c \cdot OPT(G', w) = c \cdot n$;
2. $A(G', w) \geq OPT(G', w) > c \cdot n$.

Ou seja, G possui circuito hamiltoniano se e somente se o custo do algoritmo sobre G' for menor ou igual a $c \cdot n$.

Assim, algoritmo A permite resolver o problema do Circuito Hamiltoniano, que é NP-Completo.

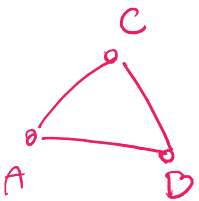
Portanto, não é possível obter uma aproximação constante para o TSP!

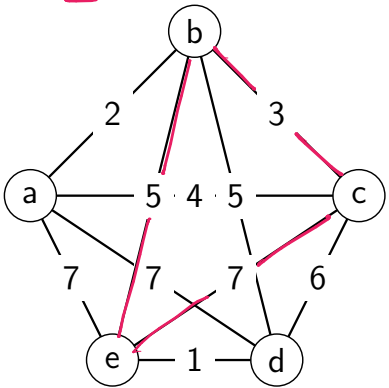
Desafio: generalize a prova para mostrar que não é possível obter sequer uma $f(n)$-aproximação para o TSP.

Algoritmo de aproximação para o TSP Métrico

Se um grafo $G = (V, E)$ com peso w nas arestas é métrico

- então ele é completo e para todo trio $u, v, x \in V$ temos que $w(uv) \leq w(ux) + w(xv)$

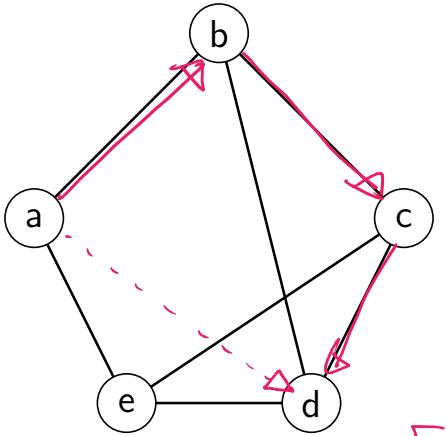




O TSP Métrico é o TSP onde o grafo de entrada é métrico.

Algoritmo de aproximação para o TSP Métrico

Um atalho (short-cut) é a substituição de um caminho entre vértices u e v, $(u, x_1, x_2, \ldots, x_k, v)$, pela aresta uv.

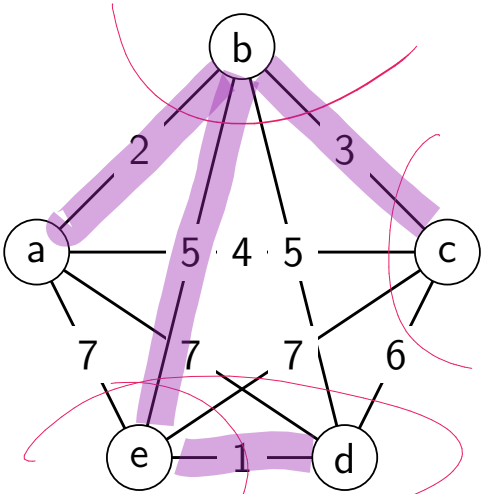


$(a, b, c, d) \rightarrow (a, d)$

Note que em um grafo métrico, $w(uv) \leq w(u, x_1, \ldots, x_k, v)$.

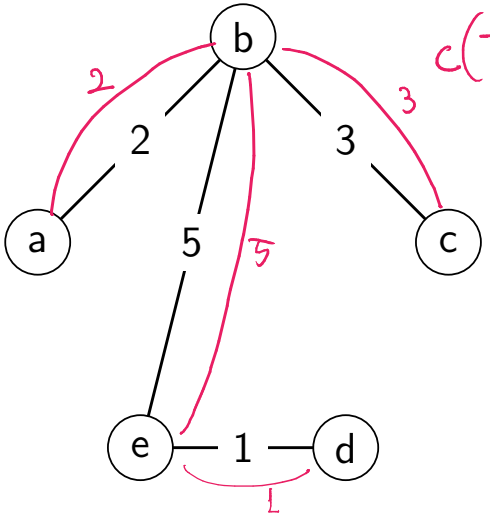
Algoritmo de aproximação para o TSP Métrico

Encontre uma árvore geradora mínima (MST) T de G.



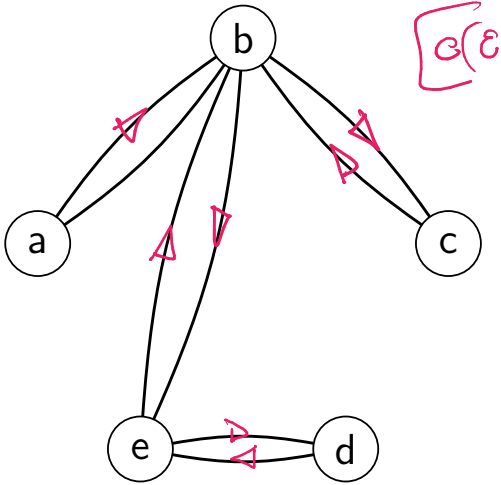
Algoritmo de aproximação para o TSP Métrico

Duplique as arestas de T.



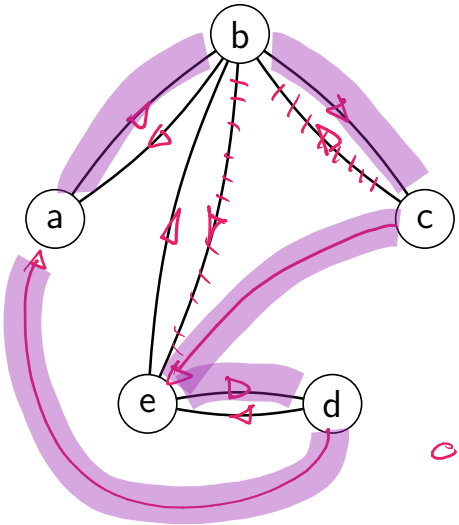
Algoritmo de aproximação para o TSP Métrico

Encontre um ciclo Euleriano $\mathcal{E}$.



$$c(\mathcal{E}) = c(T^2) = 2c(T)$$

Algoritmo de aproximação para o TSP Métrico



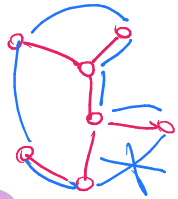
Percorra $\mathcal{E}$ fazendo short-cut se for repetir vértice.

$$\mathcal{E} = (a, b, c, b, e, d, e, b, a)$$

$$\mathcal{C} = (a, b, c, e, d, a)$$

$$c(\mathcal{C}) \leq c(\mathcal{E}) = 2c(T)$$

Algoritmo de aproximação para o TSP Métrico



1: **função** TSPMetricoAprox($G = (V, E)$, w)
2: $T \leftarrow MST(G, w)$
3: $T^2 \leftarrow T$ com arestas duplicadas
4: $\mathcal{E} \leftarrow$ ciclo euleriano em T^2
5: $\mathcal{C} \leftarrow$ circuito hamiltoniano obtido com short-cuts sobre $\mathcal{E}$
6: **devolve** $\mathcal{C}$

Análise:

$$TSP\ Metrico\ Aprox(I) \leq 2c(T)$$
$$\leq 2c(P)$$
$$\leq 2c(OPT)$$

De onde vemos que esse algoritmo é uma 2-aproximação.

Problema da Cobertura por Vértices

Cobertura por Vértices

Entrada: grafo $G = (V, E)$ com peso w nos vértices.

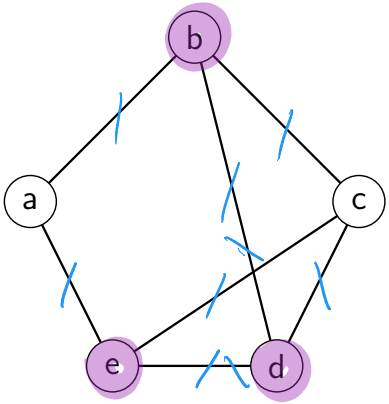
Soluções viáveis: subconjunto $S \subseteq V$ tal que para toda aresta $uv \in E$, $u \in S$ ou $v \in S$.

Função objetivo: soma dos pesos dos vértices em S.

Objetivo: encontrar solução de custo mínimo.

Esse problema é um caso particular da Cobertura por Conjuntos.

Problema da Cobertura por Vértices



Quiz: como podemos reduzir o problema da cobertura por vértices ao problema da cobertura por conjuntos?

Formulação em PLI para a Cobertura por Vértices

Sejam x_v variáveis binárias que indicam se o vértice v foi escolhido.

$$\min \sum_{v \in V} w_v x_v$$

$$\Rightarrow \text{s.a.} \left[x_u + x_v \geq 1 \right] \quad \forall e = \{u, v\} \in E$$

$$x_u \in \{0, 1\} \quad \forall u \in V$$

Programação linear é muito utilizada para encontrar soluções aproximadas.

Assim, considere o PL fruto da relaxação da integralidade do modelo anterior.

Algoritmo de aproximação para a Cobertura por Vértices

1: **função** VertexCoverAprox($G = (V, E)$, w)
2: monte e resolva o PL, obtendo x^*
3: **para** $v \in V$ **faça**
4: $x_v = 1$, se $x_v^* \geq 0.5$
5: $x_v = 0$, se $x_v^* < 0.5$
6: **devolve** $\sum_{v \in V} w_v x_v$

$O(n)$

Note que a solução construída é viável: como toda aresta satisfaz $x_u^* + x_v^* \geq 1$, algum dentre x_u^* e x_v^* deve ser pelo menos 0.5.

Análise

Se $x_v = 1$, então $x_v^* \geq 0.5 \rightarrow x_v = 1 \leq 2x_v^*$.

Se $x_v = 0$, então também vale que $x_v = 0 \leq 2x_v^*$.

$\Rightarrow x_v \leq 2x_v^*$

$$\text{VertexCoverApprox}(I) = \sum_{v \in V} w_v x_v$$

$$\leq 2 \sum_{v \in V} w_v x_v^* = 2\,OPT_{PL} \leq 2\,OPT(I)$$

De onde vemos que esse algoritmo é uma 2-aproximação.

Desafio: generalizar nosso algoritmo para tratar o problema da Cobertura por Conjuntos.

Problema do Corte Máximo

Corte Máximo em Grafos

Entrada: um grafo $G = (V, E)$.

Soluções viáveis: um corte de G, i.e., um conjunto S tal que $\emptyset \neq S \subset V$.

Função objetivo: número de arestas que sai de S, i.e., $|\delta(S)|$.

Objetivo: encontrar solução de custo máximo.

Algoritmo de aproximação para o Corte Máximo

Vizinhança de um corte S:

- cortes que tem apenas um vértice a mais ou a menos que S

Seja $c(v) = \#$ de arestas incidentes a v que atravessam o corte
e $d(v) = \#$ de arestas incidentes a v que não atravessam o corte

1: **função** CorteMaximoBuscaLocal($G = (V, E)$)
2: $S \leftarrow$ um corte inicial; $\overline{S} \leftarrow V \setminus S$
3: **enquanto** houver vértice v tal que $c(v) < d(v)$ **faça**
4: **se** $v \in S$ **então** $S \leftarrow S \setminus \{v\}$; $\overline{S} \leftarrow \overline{S} \cup \{v\}$
5: **senão** $\overline{S} \leftarrow \overline{S} \setminus \{v\}$; $S \leftarrow S \cup \{v\}$
6: **devolve** S

Exemplo de funcionamento do algoritmo

Análise

Vamos mostrar que esse algoritmo de busca local termina em tempo polinomial e tem razão de aproximação constante.

Quiz: como generalizar essa heurística para a versão ponderada do Corte Máximo?