DCC207 – Algoritmos 2

Aula 13 – Soluções aproximadas para problemas difíceis

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- As últimas aulas foram dedicadas ao estudo de técnicas para desenho de soluções exaustivas mais eficientes para problemas difíceis
- Vimos que a grande vantagem dos algoritmos de backtracking e/ou branch-and-bound segue fundamentalmente de podas bem executadas da árvore de busca
- Essas soluções, contudo, não garantem tempo polinomial, e, em alguns casos, continuam sendo inviáveis
- Alternativamente, podemos propor algoritmos que não garantam otimalidade, mas que encontrem boas soluções em tempo polinomial
- Essa abordagem se mostra muito relevante na prática, já que, frequentemente, é melhor ter uma boa solução em tempo hábil, do que a melhor em tempo consideravelmente maior
  - Até porquê problemas reais lidam com imprecisões já na coleta dos dados

Algoritmos aproximados

- Embora fundamentalmente abrimos mão da solução ótima para encontrar uma solução boa em tempo razoável, muitas vezes queremos saber quão distante do ótimo estamos
- Na verdade, estamos interessados em ter uma garantia de que a solução apresentada pelo algoritmo estará, no pior caso, a uma certa distância da ótima
- Algoritmos para os quais é possível estimar esse fator de qualidade (distância da otimalidade) são conhecidos como **algoritmos aproximados** (ou aproximativos, ou de aproximação)

Algoritmos aproximados

- Podemos medir através do erro relativo entre a solução apresentada pelo algoritmo aproximado e a solução ótima
  - Erro relativo máximo = max $\alpha(I) = (r(I) - OPT(I))/OPT(I)$
  - I é uma instância do problema; r(I) a solução do algoritmo; OPT(I) a solução ótima
- Simplificando, o erro de aproximação pode ser calculado por r(I)/OPT(I) -1
  - Consequentemente, podemos aproximá-lo por r(I)/OPT(I)
- Um algoritmo é dito ser **c-aproximado** se, para qualquer instância do problema, $\alpha(I) \leq c$
- O menor valor de c tal que a inequação continua válida é chamado de **garantia de desempenho do algoritmo (ou fator de aproximação)**
  - $r(I) \leq c * OPT(I)$, para toda instância I

Algoritmos aproximados

- Dessa forma, quanto mais próximo de 1 for o fator de aproximação de um algoritmo, melhor ele será
- Uma questão importante na definição da qualidade de um algoritmo aproximado é o valor da solução ótima
- Evidentemente não podemos computar tal valor. Portanto, devemos estimar limites realistas para tal, assim como fizemos com os algoritmos branch-and-bound
- Além disso, ao contrário do que acontecia com as soluções exatas em que um problema seria resolvido tão rapidamente quanto outro salvo um fator polinomial, as soluções aproximadas não podem ser ‘reaproveitadas’
  - Elas são desenvolvidas especificamente para um problema
  - Na verdade, alguns problemas não possuem soluções aproximadas com garantia de desempenho (TSP, por exemplo, não pode ser aproximado por um fator constante, salvo se P=NP; veremos a demonstração mais à frente)

Vertex Cover

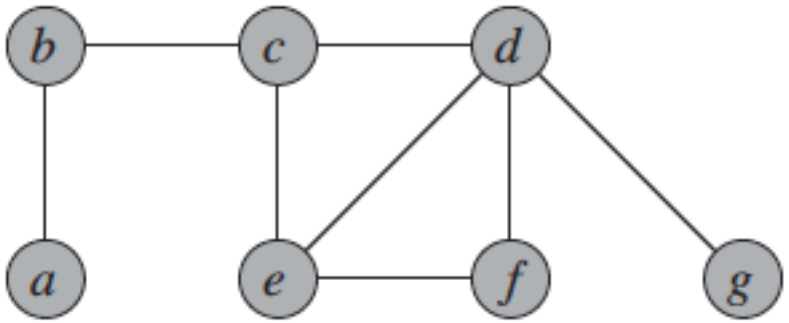
- Vamos começar nossos estudos pelo problema de Vertex Cover, o qual possui um algoritmo 2-aproximado
- Relembrando: o problema de otimização do vertex cover consiste em encontrar um subconjunto mínimo de vértice que cobre todas as arestas do grafo (todas as arestas são incidentes em algum vértice da cobertura)
- Nosso algoritmo aproximado, então, receberá um grafo G=(V,E) como parâmetro, e retornará uma cobertura de vértices cujo tamanho será, no máximo, duas vezes o tamanho da menor cobertura

Vertex Cover

```
APPROX-VERTEX-COVER(G)
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4       let (u, v) be an arbitrary edge of E'
5       C = C ∪ {u, v}
6       remove from E' every edge incident on either u or v
7  return C
```

Vertex Cover

- Exemplo:



Vertex Cover

- Se representarmos as arestas do conjunto E’ por listas de adjacências, o algoritmo possui custo da ordem de O(|V| + |E|)
  - Ou seja, ele é executado em tempo polinomial
- Resta mostrar, portanto, que a solução retornada pelo algoritmo é no máximo 2x maior que a solução ótima
  - Mas qual é a solução ótima?
  - Como estimar um limite superior realista para o tamanho da menor cobertura?
- Vamos responder a essas perguntas na demonstração de que o algoritmo é de fato 2-aproximado.

Vertex Cover

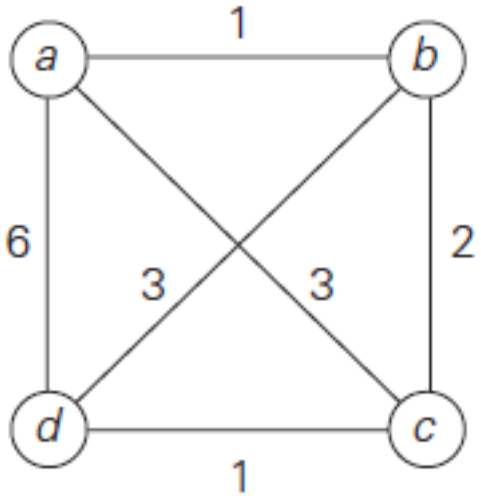
- Teorema: O Approx-Vertex-Cover é um algoritmo polinomial 2-aproximado para o problema de otimização de vertex cover
- Prova (ideia):
  - Como analisado anteriormente, o algoritmo executa em tempo polinomial.
  - Podemos perceber que o algoritmo encontra um matching maximal no grafo.
  - Qualquer cobertura deve cobrir todas as arestas do grafo. Logo, uma cobertura mínima deve ter, pelo menos, a mesma quantidade de vértices quanto arestas no matching maximal ($|C^*| \geq |M|$)
  - Como o matching é maximal, o conjunto C, construído na linha 5, cobre todas as arestas do grafo.
  - Esse conjunto contém $2|M|$ vértices, portanto, $2|M|=|C|$.
  - Portanto, pela desigualdade acima, temos $2|C^*| \geq 2|M|=|C|$

Problema do Caixeiro Viajante

- Vamos revisitar agora o problema do caixeiro viajante
  - Vamos considerar o caso de grafos não-direcionados como simplificação; os resultados podem ser facilmente extrapolados para grafos direcionados
- Vamos explorar 3 algoritmos aproximados, antes de provar que, no caso geral, o problema não admite aproximação por um fator constante, exceto se P=NP
- Primeiro, vamos considerar a seguinte abordagem gulosa:
  - Escolha um vértice arbitrário como início do percurso
  - Enquanto houver vértices não visitados:
    - Visite o vértice mais próximo do último visitado
  - Retorne ao vértice inicial

Problema do Caixeiro Viajante

- Considerando o grafo ao lado e o vértice a como o inicial, o algoritmo retorna:
  - a-b-c-d-a com custo 10
- A solução ótima é: a-b-d-c-a com custo 8
- Nesse caso, nosso algoritmo é 25% pior que o ótimo
- Contudo, esse não é o fator de aproximação
- O algoritmo, na verdade, pode levar a uma solução muito pior
- Se o custo de a-d for w≥6, então o fator de aproximação será (4+w)/8; ou seja, ele pode ser tão grande quanto quisermos, escolhendo um valor apropriado para w



Problema do Caixeiro Viajante

- A abordagem gulosa, apesar de extremamente simples, mostrou-se muito ruim, já que não temos garantia alguma da qualidade da resposta
- Vamos considerar, agora, um caso particular do problema. Vamos supor que a função de custo do grafo seja uma métrica:
  - $c(u,v) \geq 0$
  - $c(u,v) = 0$ sse $u = v$
  - $c(u,v) = c(v,u)$
  - $c(u,v) \leq c(u,w) + c(w,v)$
- Na prática, muitas instâncias do problema satisfazem essas restrições
- Um caso particular seria o conjunto de instâncias geométricas:
  - Vértices são pontos no plano
  - Custo é a distância euclidiana entre os vértices
- Essa versão do problema é conhecida como Problema do Caixeiro Viajante Euclidiano

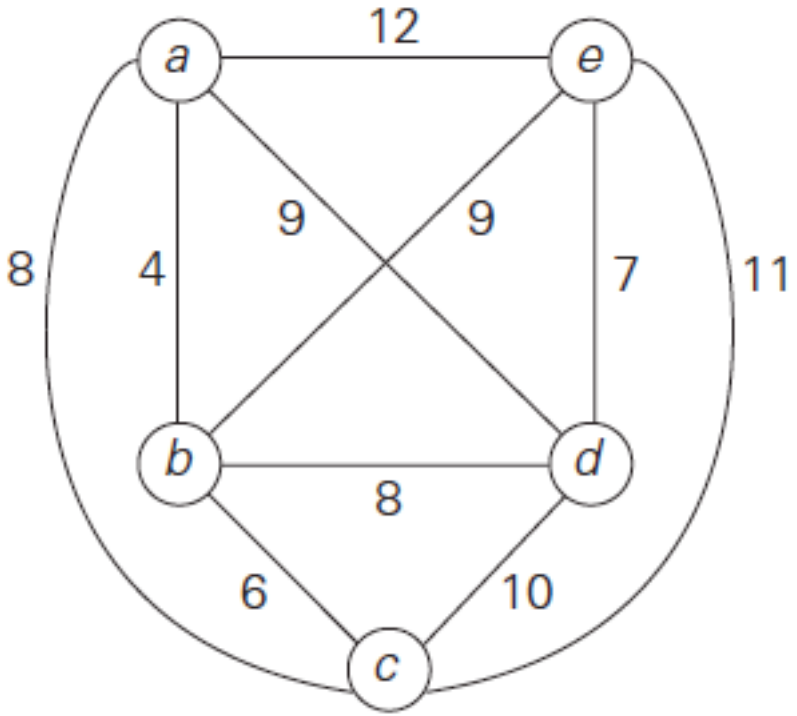
Problema do Caixeiro Viajante

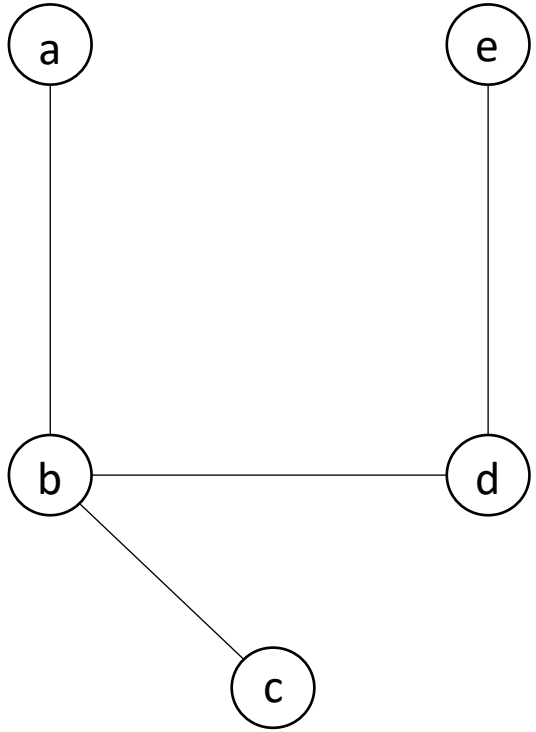
- Uma boa aproximação para o custo de uma solução é o custo da árvore geradora mínima do grafo
- Assim, podemos usar o seguinte algoritmo (twice-around-the-tree):

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r
using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited
in a preorder tree walk of T
4 **return** the hamiltonian cycle H

Problema do Caixeiro Viajante





Problema do Caixeiro Viajante

- Teorema: O Approx-TSP-Tour é um algoritmo polinomial 2-aproximado para o problema do caixeiro viajante euclidiano
- Prova (ideia):
  - O custo do algoritmo é dominado pela computação da árvore geradora mínima (Prim ou Kruskal)
  - Considere o circuito hamiltoniano de menor custo H* (solução ótima). A remoção de qualquer aresta resulta em uma árvore geradora T de G. Portanto, $c(H^*) > c(T) \geq c(T^*)$; em que T* é a árvore geradora mínima de G.
  - Considere um circuito que passe por todos os vértices da árvore geradora mínima de G. Esse circuito tem custo $2c(T^*)$.
  - Como a função de custo é uma métrica, qualquer repetição de vértices pode ser eliminada, substituindo o subcaminho u-w-v pela aresta u-v
    - Essa substituição é equivalente ao caminhamento em pré-ordem usada no algoritmo
  - Logo, $2c(T^*) \geq c(H)$
  - Combinando as duas inequações, temos $2c(H^*) > 2c(T^*) \geq c(H)$
  - Portanto, o fator de aproximação do algoritmo é, de fato, estritamente menor que 2

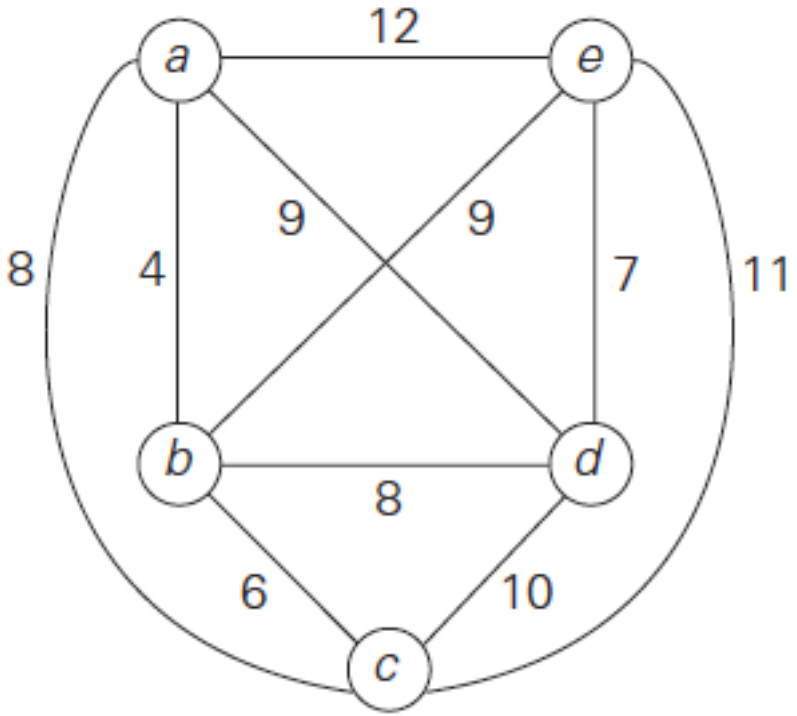
Problema do Caixeiro Viajante

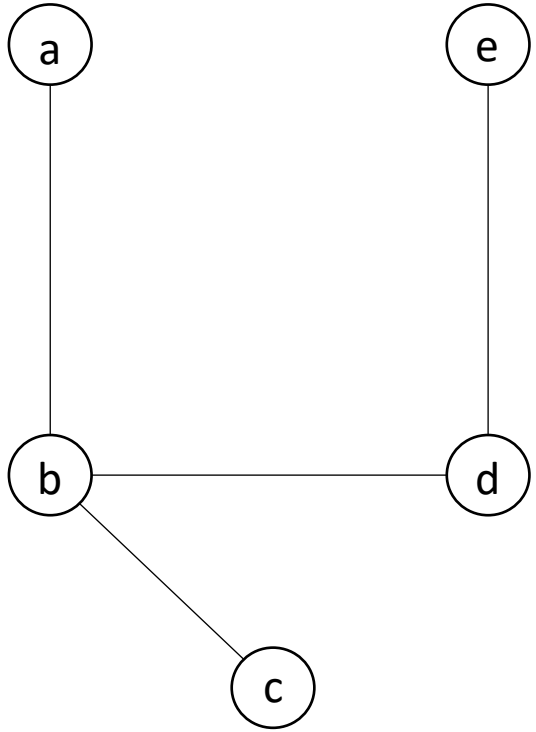
- O algoritmo twice-around-the-tree constrói o circuito hamiltoniano a partir do circuito euleriano no multigrafo com as arestas da árvore geradora mínima duplicadas
- Partindo dessa ideia de transformar um circuito euleriano em um hamiltoniano, podemos encontrar um matching perfeito de peso mínimo entre os vértices de grau ímpar na AGM, e construir o multigrafo com as arestas da AGM e do matching
- Depois, podemos transformar o circuito euleriano no hamiltoniano da mesma forma que fizemos no algoritmo anterior
- Esse algoritmo garante um fator de aproximação de 1.5. Ele é conhecido como o algoritmo de Christofides (publicado em 1976 por Nicos Christofides na CMU)

Problema do Caixeiro Viajante

- Algoritmo de Christofides:
  1. Compute T uma árvore geradora mínima de G
  2. Seja I o conjunto de vértices de grau ímpar de T. Compute M um matching perfeito de peso mínimo no subgrafo induzido por I
  3. Seja G’ o multigrafo formado com os vértices de V e aresta de M e T. Compute um circuito euleriano em G’ (aplicar DFS)
  4. Elimine vértices duplicados, substituindo subcaminhos u-w-v por arestas u-v

Problema do Caixeiro Viajante





Problema do Caixeiro Viajante

- Teorema: O algoritmo de Christofides é um algoritmo polinomial 1.5-aproximado para o TSP euclidiano
- Prova (ideia):
  - O custo do algoritmo é dominado pela computação do matching perfeito de peso mínimo. O algoritmo de Edmonds (Blossom) computa o matching em tempo $O(|V|^3)$
  - A primeira parte da prova do twice-around-the-tree continua válida. $c(H^*) > c(T) \geq c(T^*)$
  - Considere agora um circuito hamiltoniano de menor custo obtido no subgrafo induzido por I. O custo desse circuito $c(H'^*) \leq c(H^*)$, já que o subgrafo induzido contém um subconjunto de arestas de G.
  - Considere uma enumeração das arestas de H'^*, iniciando do vértice inicial do circuito, e ignorando a última aresta (a de retorno ao inicial). O conjunto de arestas de H'^* pode ser dividido entre arestas pares e ímpares conforme a enumeração. Como o custo do circuito é $c(H'^*)$, tem-se que o custo de cada um desses conjuntos é, no máximo, $c(H'^*)/2$. Como cada um deles é um matching no subgrafo induzido, tem-se que o custo do matching mínimo é $c(M) \leq c(H'^*)/2 \leq c(H^*)/2$
  - O custo do circuito euleriano é, portanto, $c(T) + c(M) \leq c(H^*) + c(H^*)/2 \leq 1.5c(H^*)$
  - Como a função de custo é uma métrica, a substituição dos vértices repetidos por ligações diretas somente reduz o custo do circuito euleriano. Portanto, $c(H) \leq 1.5c(H^*)$

Problema do Caixeiro Viajante

- Teorema: O problema do caixeiro viajante no caso geral não pode ser aproximado por um fator constante, salvo se P=NP.
- Prova (ideia): A prova será por contradição.
  - Suponha que exista um algoritmo com fator de aproximação constante c para o problema do caixeiro viajante.
  - Vamos mostrar que esse algoritmo resolve o problema do circuito hamiltoniano.
  - Seja G=(V,E) um grafo no qual queremos determinar a existência de um circuito hamiltoniano
  - Vamos construir, a partir de G, uma instância do TSP.

Problema do Caixeiro Viajante

- Seja G’ um grafo completo construído a partir de G
- Definimos a função de custo da seguinte forma:
  - $c(u,v) = 1$, se $(u,v) \in E$
  - $c(u,v) = c^*|V| + 1$, caso contrário
- Essa redução é feita em tempo polinomial
- Se G contém um circuito hamiltoniano, então a solução ótima do TSP será esse circuito com custo $|V|$
- Se G não contém um circuito hamiltoniano, então a solução ótima deve incluir ao menos uma aresta que não pertence a G
  - $c(H^*) \geq c^*|V|+1 + (|V|-1) = c^*|V| + |V| > c^*|V|$
- Logo, a solução que não utiliza o circuito hamiltoniano de G é, pelo menos, c+1 vezes maior que a que usa o circuito
- Como o algoritmo aproximado tem fator de aproximação c, ele pode ser usado em G’ para determinar a existência de um circuito hamiltoniano em G.
- Como o problema do circuito hamiltoniano é NP-completo, esse algoritmo só pode existir se P=NP.

Leitura

- Seções 35.1, 35.2 (CLRS)
- Seção 12.3 até Knapsack (Levitin)
- Complementar:
  - Seção 18.1 (Goodrich e Tamassia)