Introdução à classe de problemas NP-Completos: Redução de problemas

J. Pascoal Faria, R. Rossetti, L. Ferreira

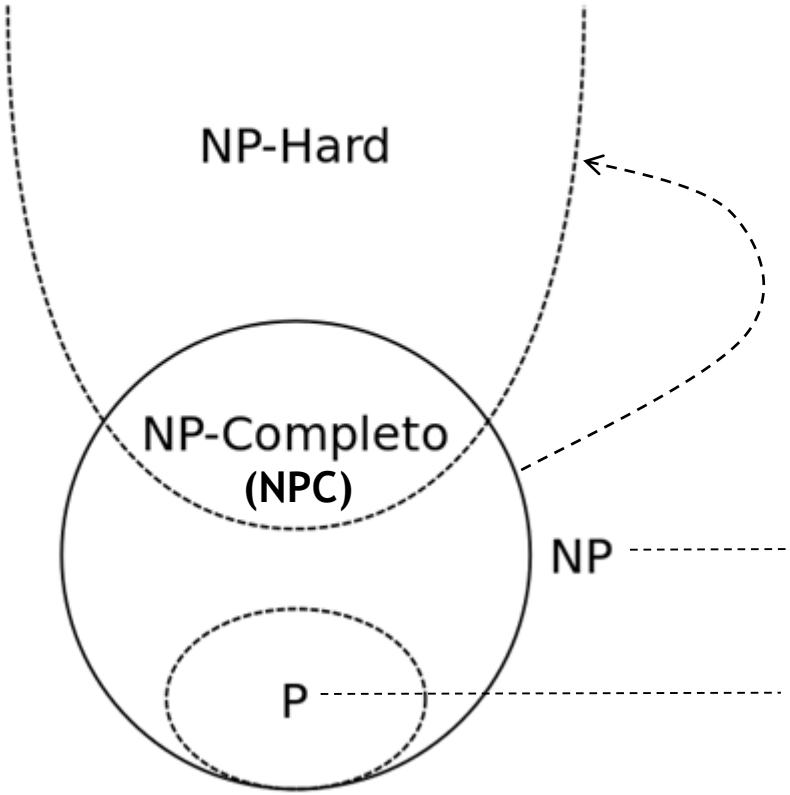
FEUP, MIEIC, CAL, 2017/2018

Índice

- Revisão da aula anterior
- Exemplos de reduções
- Problema do caixeiro viajante

Revisão da aula anterior

Classes P, NP, NP-*complete* e NP-*hard*



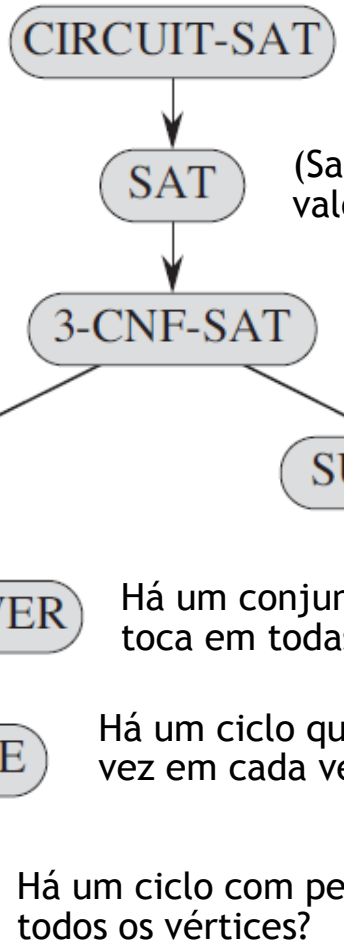
Redução em tempo polinomial

Verificáveis em tempo polinomial

Resolúveis em tempo polinomial

Na hipótese P ≠ NP!

Exemplos de problemas NPC e reduções



Há um atribuição de valores lógicos às entradas de um circuito que torna as saídas verdadeiras?

(Satisfação booleana) Há um atribuição de valores lógicos que torna a expressão verdadeira?

Variante de SAT, com 3 literais apenas

Há um clique (subgrafo completo) de tamanho ≥k?

Há um subconjunto de soma zero (ou soma k)?

Há um conjunto com ≤ k vértices que toca em todas as arestas?

Há um ciclo que passa exatamente uma vez em cada vértice?

Há um ciclo com peso ≤ k que passa em todos os vértices?

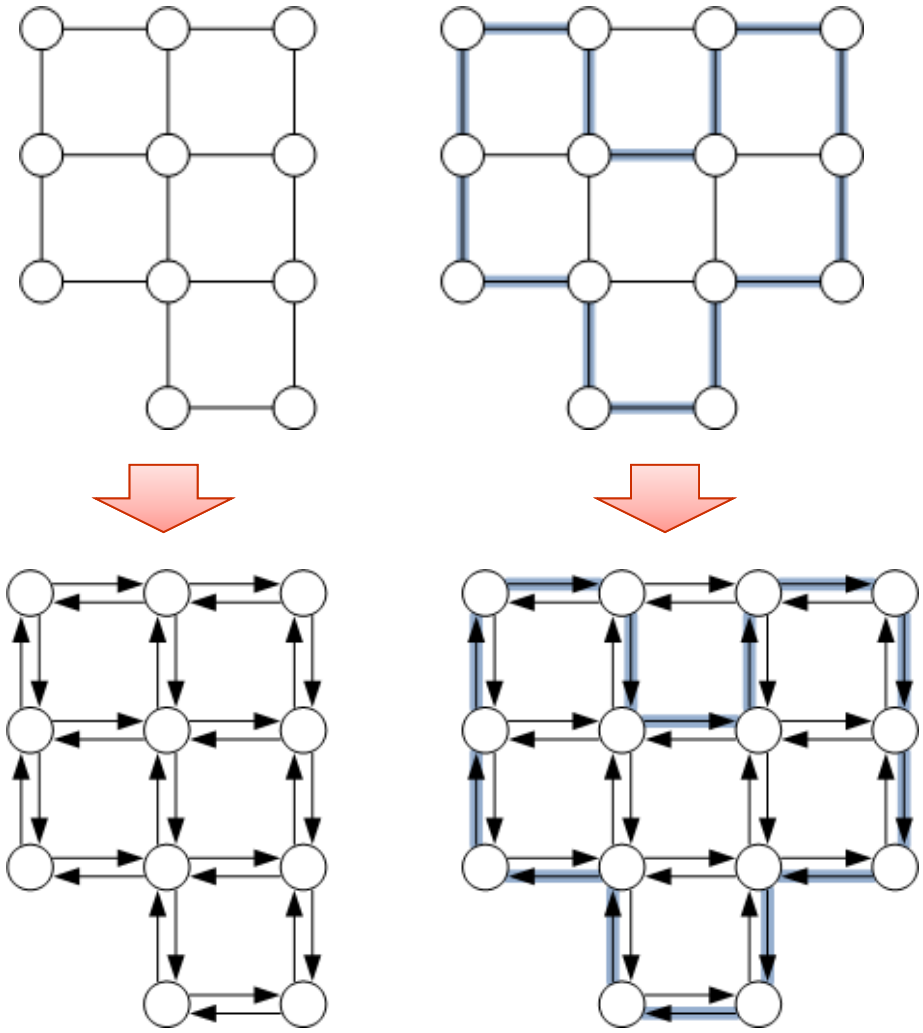
Exemplos de reduções

Undirected Hamiltonian Cycle (UHC) é NPC?

- Problema: Sabendo-se que o problema UHC (*Undirected Hamiltonian Cycle*) é NP-completo, provar que o problema DHC (*Directed Hamiltonian Cycle*) é também NP-completo
- Resolução:
  a) Um ciclo Hamiltonian candidato é facilmente verificável em tempo polinomial, logo DHC $\in$ NP
  b) O problema UHC é facilmente redutível ao problema DHC em tempo polinomial (ver slide seguinte), logo DHC $\in$ NP-hard

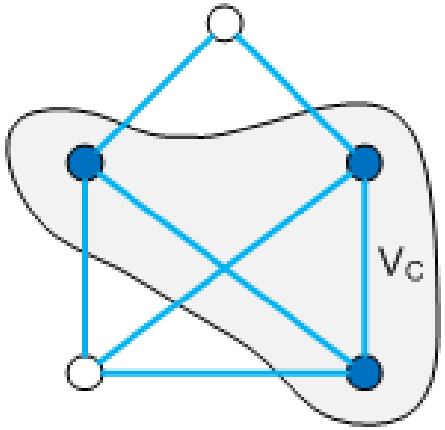
Redução de UHC a DHC

- Dado grafo não dirigido $G=(V,E)$, cria-se grafo dirigido $G'=(V,E')$ pela substituição de cada aresta $\{u, v\} \in E$ por duas arestas dirigidas (u, v) e $(v, u) \in E'$
- É fácil de constatar que G terá um ciclo Hamiltoniano sse G' também o tiver!
  - Excetua-se caso de grafo não-dirigido completo com 2 vértices, que, por convenção, não se considera Hamiltoniano



Vertex Cover (VC)

- Uma cobertura de vértices de um grafo $G = (V, E)$ é um subconjunto $V_C \subseteq V$, tal que toda aresta $(a, b) \in E$ é incidente em pelo menos um vértice $u \in V_C$.

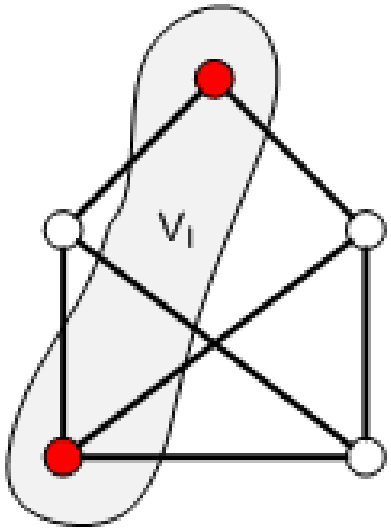


  - Vértices em V_C "cobrem" todas as arestas em G.

- Problema de decisão (*VC*):
  - O grafo G tem uma cobertura de vértices de tamanho $\leq k$?

Independent Set (IS)

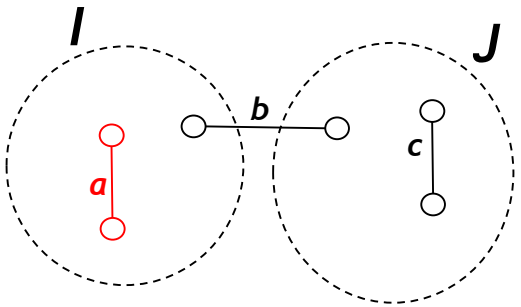
- Um conjunto independente de um grafo $G = (V, E)$ é um subconjunto $V_I \subseteq V$, tal que não há dois vértices em V_I que partilham uma aresta de E
  - $u, v \in V_I$ não podem ser vizinhos em G.



- Problema de decisão (*IS*):
  - O grafo G tem um conjunto independente de tamanho $\geq k$?

Dualidade VC ↔ IS

- Dado grafo não dirigido *G=(V,E)*, seja *I*, *J* uma partição de V em dois subconjuntos disjuntos (i.e., $I \cup J = V$ e $I \cap J = \varnothing$)
- Se *I* é um conjunto independente de vértices, então não podem existir arestas do tipo *a*, logo os vértices em *J* tocam todos as arestas de G, logo *J* é uma cobertura de vértices
- Se *J* é uma cobertura de vértices, então não podem existir arestas do tipo *a*, logo *I* é um conjunto independente de vértices.
- ***I*** é um conj. indep. de vértices ⇔ ***V*** \ ***I*** é uma cobertura de vértices



Vertex Cover é NPC?

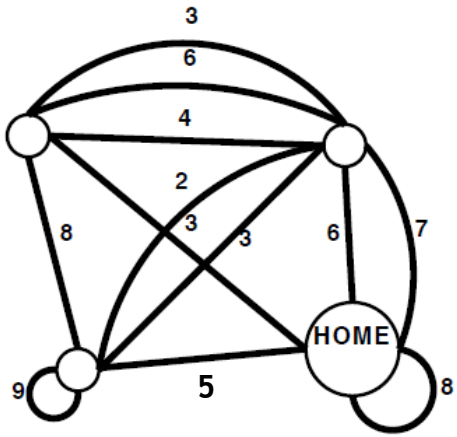
- Problema: Sabendo-se que **IS** $\in$ **NPC**, provar que **VC** $\in$ **NPC**
- Resolução:
  a) Dada um conjunto candidato de vértices V_C, é fácil verificar em tempo polinomial *se* $|V_C| \leq k$ e se toca em todas as arestas, logo **VC**$\in$**NP**
  b) Para provar que **VC** $\in$ **NP-hard**, indicamos de seguida uma redução de tempo polinomial de **IS** em **VC**

Redução de IS a VC

- Seja uma instância qualquer de IS: $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}), k$
- Pela propriedade da dualidade, G tem um conjunto independente de vértices (V_I) de tamanho $\geq k$ sse tiver uma cobertura de vértices (V_c) de tamanho $\leq k'$, com $k' = |V| - k$
- Assim, a conversão de entradas é trivial:
  - Dada uma instância qualquer de IS: $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}), k$
  - Constrói-se uma instância de VC: $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}), k' = |V| - k$
- A conversão de saídas é também trivial:
  - Conversão de 'certificados': $V_c \rightarrow V_I = V \setminus V_c$
  - Conversão de decisão: mantém-se a mesma decisão

Problema do Caminhada (*Jogging (J)*)

- Considere um grafo não dirigido *G*, admitindo arestas paralelas e anéis, com pesos inteiros positivos nas arestas, no qual se distingue um vértice *home*.
- O problema da caminhada (*Jogging (J)*) consiste em verificar se existe um caminho de peso total *k*, começando e terminando em *home*, sem repetir arestas.
- Prove que é um problema NPC, sabendo-se que o problema da soma de subconjuntos é NPC.

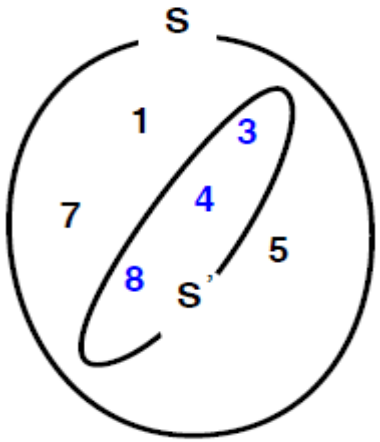


Problema da Soma de Subconjuntos (SS)

- Dado um conjunto de inteiros positivos, ***S***, há um subconjunto, ***S'*** em ***S***, tal que a soma dos elementos de ***S'*** seja *k*?

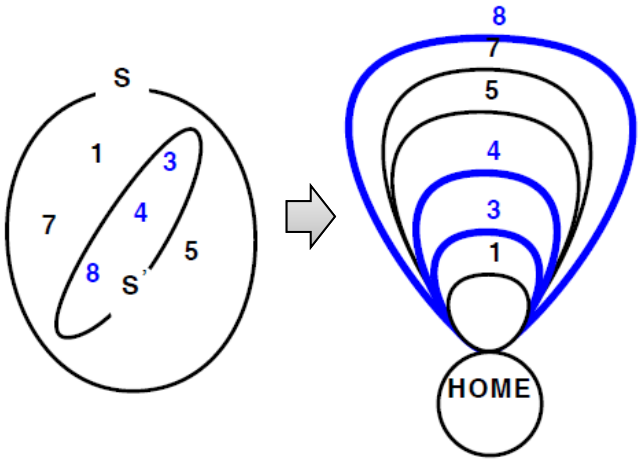
Ex:
S = {1, 3, 4, 5, 7, 8}

Find S' with sum = 15!



Problema da Caminhada é NPC

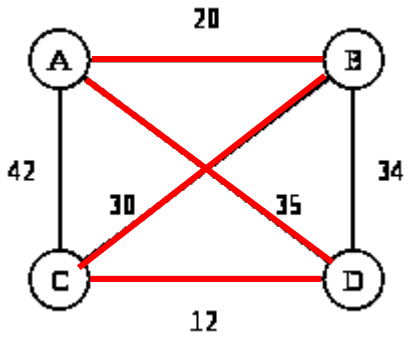
- Um caminho candidato é facilmente verificável em tempo polinomial, logo J ∈ NP
- Para provar que J ∈ NP-hard, reduz-se SS a J em tempo polinomial da seguinte forma:
  - Dado um conjunto *S*, cria-se um grafo *G* com um único vértice *home* e um anel de peso *x* para cada elemento $x \in S$
  - *S* tem um subconjunto de soma *k* sse *G* tem um caminho de peso total *k* sem repetir arestas



Problema do caixeiro viajante

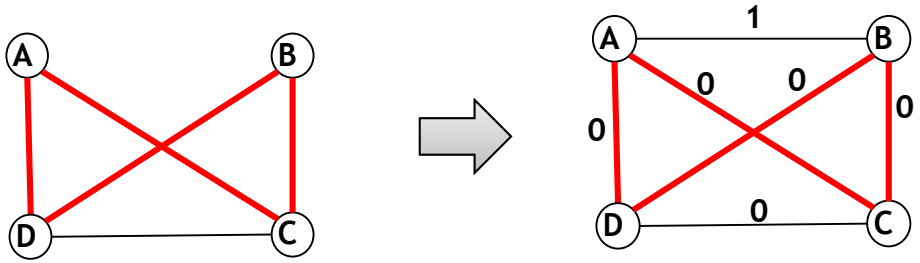
Travelling Salesman Problem (TSP)

- Problema de optimização: dado um grafo não dirigido G=(V, E) com pesos inteiros não negativos, encontrar um ciclo de peso total mínimo que passa em todos os vértices (cidades)
- Problema de decisão: dado um grafo não dirigido G=(V, E) com pesos inteiros não negativos, existe um ciclo de peso $\leq$ k (k inteiro não negativo) que passa em todos os vértices?
- Pré-processamento habitual: constrói-se um grafo completo com as distâncias mínimas entre todos os pares e vértices; trata-se então de encontrar um ciclo Hamiltoniano de peso mínimo (ou peso $\leq$ k)



TSP é NP-Completo?

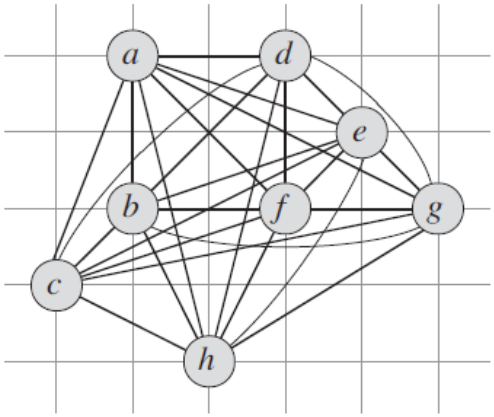
- Provar que TSP $\in$ NPC, sabendo-se que UHC $\in$ NPC:
  a) Uma solução candidata é facilmente verificável em tempo polinomial, logo TSP $\in$ NP
  b) Para provar que TSP $\in$ NP-hard, reduz-se UHC a TSP em tempo polinomial da seguinte forma:
     - Dada uma instância de UHC, num grafo não dirigido G=(V,E), constrói-se um grafo completo pesado G' = (V, E'), com peso 0 nas arestas {u,v} $\in$ E, e peso 1 nas arestas {u,v} $\notin$ E
     - G tem um ciclo Hamiltoniano sse G' tem um ciclo Hamiltoniano de peso $\leq$ 0.



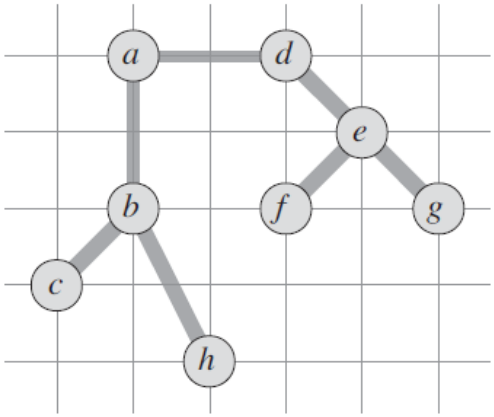
Algoritmos para resolver TSP

- Algoritmos exatos (n = nº de vértices, m = nº de arestas):
  - Testar todas as permutações $\rightarrow$ T(n) = O(n!) (tratável até n ~ 10)
  - Resolver o problema de forma recursiva, com programação dinâmica, para todos os subconjuntos de vértices (algoritmo Held-Karp), usando um mapa de bits para representar cada subconjunto $\rightarrow$ $T(n) = O(n^2 2^n)$ (tratável até n ~ 20)
- Heurísticas e algoritmos aproximados de tempo polinomial
  - Heurística (gananciosa) do vizinho mais próximo é fácil de implementar, mas não dá garantias de aproximação no caso geral
  - Quando se verifica a desigualdade triangular (caso de mapas de estradas), visita em pré-ordem de árvore de expansão mínima dá solução de peso que não excede **2x** o ótimo
  - No mesmo pressuposto, algoritmo de Christofides, garante solução que não excede em **1.5x** o ótimo

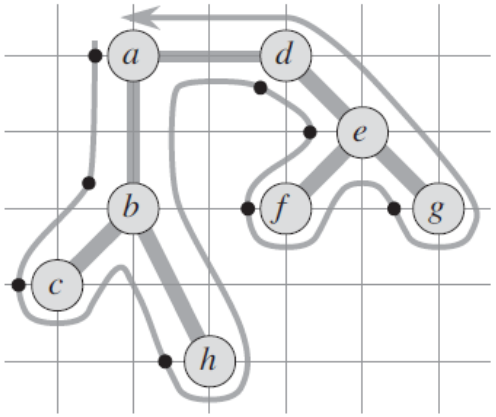
TSP e MST



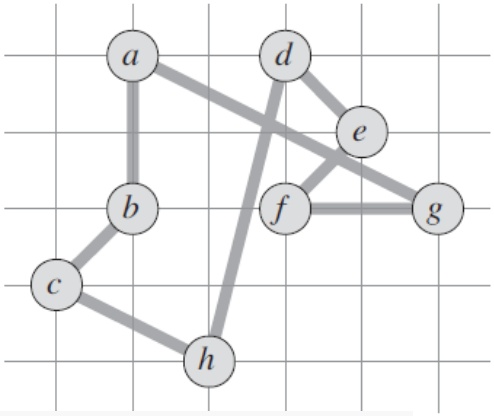
Grafo original (pesos omitidos)



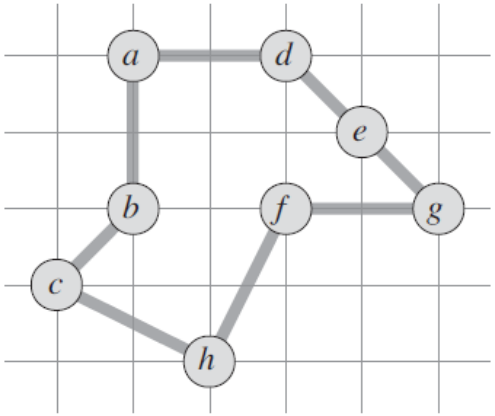
Árvore de expansão mínima



Visita em pré-ordem começando em *a*



Percurso correspondente



Percurso ótimo

* Solução exata de TSP com prog.dinâmica

- Definição recursiva para encontrar percurso dirigido de custo mínimo entre vértices *s* e *t* (iguais ou distintos), passando por um conjunto *U* de outros vértices (*s*,*t* $\notin$ *U*)

$$minCost(s,t,U) = \begin{cases} dist(s,t), & se\ U = \{\} \\ \min\{ dist(s,u) + minCost(u,t,U\backslash\{u\}) \mid u \in U\}, & se\ U \neq \{\} \end{cases}$$

  - Inicialmente, procura-se *minCost(s, s, V\{s})*
  - Memoriza-se a solução para cada combinação de argumentos
    - *U* pode ser representado por um mapa de bits
    - *t* pode ser omitido pois não muda
    - Em vez de guardar a solução completa de cada sub-problema (sequência de vértices), basta guardar o vértice *u* escolhido

Referências e mais informação

- T. Cormen *et al*. (2009) "**Introduction to Algorithms.**" Cambridge, MA: MIT press.
  - Capítulo 34 NP-Completeness
  - Capítulo 35 Approximation Algorithms
- R. Johnsonbaugh & M. Schaefer (2004) "**Algorithms.**" Upper Saddle River, NJ: Prentice Hall.
- C.A. Shaffer (2001) "**A Practical Introduction to Data Structures and Algorithm Analysis.**" Upper Saddle River, NJ: Prentice Hall.