Problemas NP-Completo

Tópicos II

Thiago Silva Vilela

Introdução

- Problemas intratáveis ou difíceis são comuns na natureza e nas áreas do conhecimento.
- Problemas “fáceis”: resolvidos por algoritmos polinomiais.
- Problemas “difíceis”: somente possuem algoritmos exponenciais para resolvê-los.
- A complexidade de tempo da maioria dos problemas é polinomial ou exponencial.
- Polinomial: função de complexidade é $O(p(n))$, onde $p(n)$ é um polinômio.
  - Ex.: algoritmos com pesquisa binária ($O(\log n)$), pesquisa sequencial ($O(n)$), ordenação por inserção ($O(n^2)$), e multiplicação de matrizes ($O(n^3)$).
- Exponencial: função de complexidade é $O(c^n)$, $c > 1$.
  - Ex.: problema do caixeiro viajante (PCV) ($O(n!)$).
  - Mesmo problemas de tamanho pequeno a moderado não podem ser resolvidos por algoritmos não-polinomiais.

Problemas NP

- É possível mostrar que os problemas para os quais não há algoritmo polinomial conhecido são computacionalmente relacionados.

- Formam a classe conhecida como NP.

- Propriedade: um problema da classe NP poderá ser resolvido em tempo polinomial se e somente se todos os outros problemas em NP também puderem.

- Este fato é um indício forte de que dificilmente alguém será capaz de encontrar um algoritmo eficiente para um problema da classe NP.

Classe NP

- Para o estudo teórico da complexidade de algoritmos considera-se problemas cujo resultado da computação seja “sim” ou “não”.

- Versão do Problema do Caixeiro Viajante (PCV) cujo resultado é do tipo “sim/não”:
  - Dados: uma constante *k*, um conjunto de cidades $C = \{c_1, c_2, \cdots, c_n\}$ e uma distância $d(c_i, c_j)$ para cada par de cidades $c_i, c_j \in C$.
  - Questão: Existe um “roteiro” para todas as cidades em C cujo comprimento total seja menor ou igual a k?

- Característica da classe NP: problemas “sim/não” para os quais uma dada solução pode ser verificada facilmente.

- A solução pode ser muito difícil ou impossível de ser obtida, mas uma vez conhecida ela pode ser verificada em tempo polinomial.

Algoritmos Não Deterministas

- Uma outra forma de caracterizar problemas NP é através de algoritmos não deterministas.

- Algoritmos não deterministas fazem uso de um "oráculo".
  - Utilizam uma função *escolhe(C)*, que escolhe um dos elementos de um conjunto *C* de forma arbitrária.
  - Se um conjunto de possibilidades levam a uma resposta, este conjunto é escolhido sempre e o algoritmo terminará com sucesso.

- Um algoritmo não determinista termina sem sucesso se e somente se não há um conjunto de escolhas que indica sucesso.

Algoritmos Não Deterministas

- Exemplo: pesquisa não determinista
- Pesquisar o elemento x em um conjunto de elementos A[1 : n], n ≥ 1.

```
void PesquisaND(A) {
     j ← escolhe(A)
     if (A[j] == x) sucesso;
     else insucesso;
}
```

- O algoritmo tem complexidade não-determinista O(1).

Classes P e NP

- P: conjunto de todos os problemas que podem ser resolvidos por **algoritmos deterministas** em tempo polinomial.
- NP: conjunto de todos os problemas que podem ser resolvidos por **algoritmos não-deterministas** em tempo polinomial.
- Para mostrar que um determinado problema está em NP, basta apresentar um algoritmo não-determinista que execute em tempo polinomial para resolver o problema.
- Outra maneira é encontrar um algoritmo determinista polinomial para verificar que uma dada solução é válida.

Classes P e NP

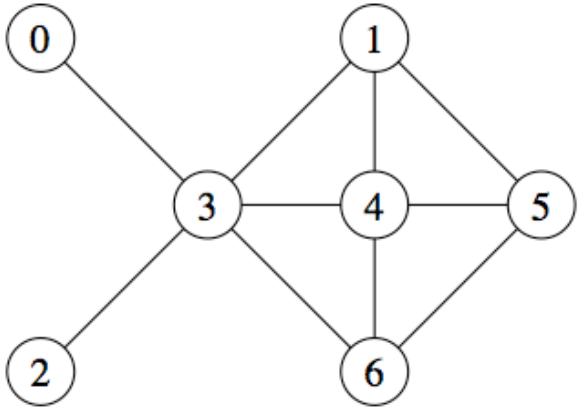
- $P \subseteq NP$, pois algoritmos deterministas são um caso especial dos não deterministas.
- A questão é se $P = NP$ ou $P \neq NP$.
- Esse é o problema não resolvido mais famoso que existe na área de ciência da computação.
- Se existem algoritmos polinomiais deterministas para todos os problemas em NP, então $P = NP$.
- Por outro lado, a prova de que $P \neq N P$ parece exigir técnicas ainda desconhecidas.
- Acredita-se que $NP \gg P$, pois para muitos problemas em NP, não existem algoritmos polinomiais conhecidos.

Transformação Polinomial

- Sejam X_1 e X_2 dois problemas de decisão.
- Suponha que o algoritmo A_2 resolva X_2.
- Se for possível transformar X_1 em X_2 e a solução de X_2 em solução de X_1, então podemos usar A_2 para resolver A_1.
- Se essas transformações puderem ser feitas em tempo polinomial, então X_1 é polinomialmente transformável em X_2.

Transformação Polinomial

- Exemplo: Conjunto independente de vértices e clique de um grafo.
  - Conjunto independente: conjunto de vértices não adjacentes. Ex.:{0, 2, 1, 6};
  - Clique: conjunto de vértices adjacentes. Ex.: {3, 1, 4}



Transformação Polinomial

- X_1: Clique
- X_2: Conjunto independente

- Podemos transformar X_1 em X_2:
  - Encontrar um clique de tamanho k em um grafo G equivale a encontrar um conjunto independente de tamanho k no grafo complementar de G.
  - Como encontrar um grafo complementar é um operação polinomial, a transformação é polinomial.

Problemas NP-Completo e NP-Difícil

- Dois problemas X_1 e X_2 são polinomialmente equivalentes se e somente se $X_1 \propto X_2$ e $X_2 \propto X_1$

- Um problema X é NP-difícil se e somente se todos os problemas NP-completo forem redutíveis a X em tempo polinomial.

- Um problema de decisão X é denominado NP-completo quando:
  - 1. $X \in$ NP;
  - 2. Todo problema de decisão $X' \in$ NP-completo satisfaz $X' \propto X$.

Problemas NP-Completo e NP-Difícil

- Um problema de decisão X que seja NP-difícil pode ser mostrado ser NP-completo exibindo um algoritmo não-determinista polinomial para X.

- Apenas problemas de decisão (“sim/não”) podem ser NP-completo.

- Problemas de otimização podem ser NP-difícil.

- A dificuldade de um problema NP-difícil não é menor do que a dificuldade de um problema NP-completo.

Problemas NP-Completo e NP-Difícil

- Exemplo: problema da parada.
  - É um exemplo de problema NP-difícil que não é NP-completo.

- Consiste em determinar, para um algoritmo determinista qualquer A com entrada de dados E, se o algoritmo A termina (ou entra em um loop infinito).

- É um problema **indecidível**. Não há algoritmo de qualquer complexidade para resolvê-lo.

- É possível reduzir problemas NP-Completo ao problema da parada.

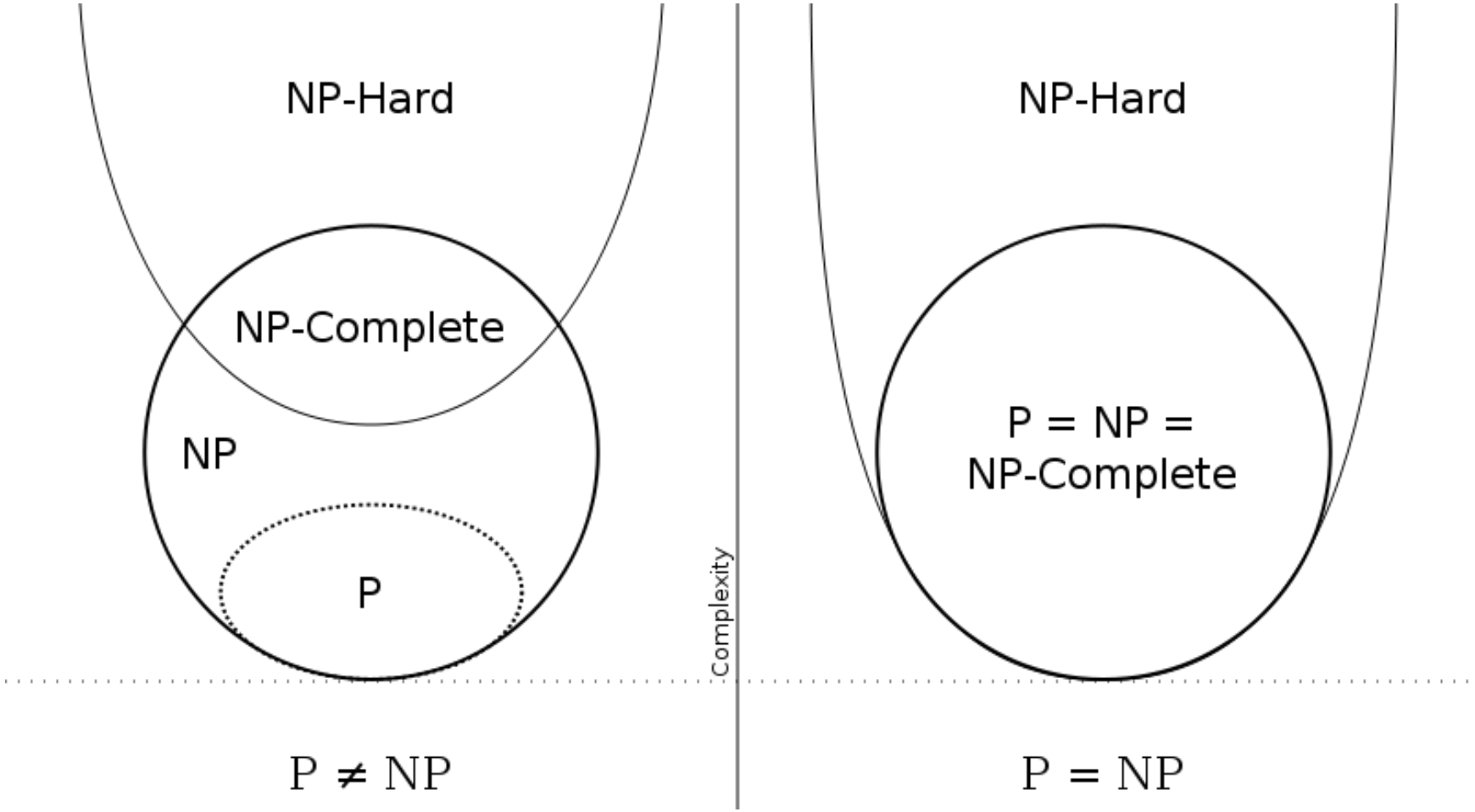
Classe NP-Intermediária

- Assumindo que P ≠ NP, existe uma outra classe de problemas: NP-Intermediária.

- Classe intermediária entre P e NP.

- NPI seria constituída por problemas que ninguém conseguiu uma redução polinomial de um problema NP-completo para eles, onde NPI = NP - (P ∪ NP-completo).

Resumo


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Exemplos de Questões

Considerando as classes de problema P, NP e NP-completo , analise as afirmações que se seguem.

I. Existem problemas na classe P que não estão na classe NP.

II. Se o problema A pode ser eficientemente transformado no problema B e B está na classe P, então A está na classe P.

III. Se P = NP, então um problema NP-completo pode ser solucionado eficientemente.

IV. Se P é diferente de NP, então existem problemas na classe P que são NP-completos.

É correto apenas o que se afirma em

A) I. B) IV. C) I e III. D) II e III. E) II e IV.

Exemplos de Questões

O problema da parada para máquinas de Turing, ou simplesmente problema da parada, pode ser assim descrito: determinar, para quaisquer máquina de Turing M e palavra w, se M irá eventualmente parar com entrada w.

Mais informalmente, o mesmo problema também pode ser assim descrito: dados um algoritmo e uma entrada finita, decidir se o algoritmo termina ou se executará indefinidamente.

Para o problema da parada,

A) existe algoritmo exato de tempo de execução polinomial para solucioná-lo.

B) existe algoritmo exato de tempo de execução exponencial para solucioná-lo.

C) não existe algoritmo que o solucione, não importa quanto tempo seja disponibilizado.

D) não existe algoritmo exato, mas existe algoritmo de aproximação de tempo de execução polinomial que o soluciona, fornecendo respostas aproximadas.

E) não existe algoritmo exato, mas existe algoritmo de aproximação de tempo de execução exponencial que o soluciona, fornecendo respostas aproximadas.