UDESC

Teoria da Computação

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

NP-Completude

Introdução

O status de muitos problemas NP é desconhecido:

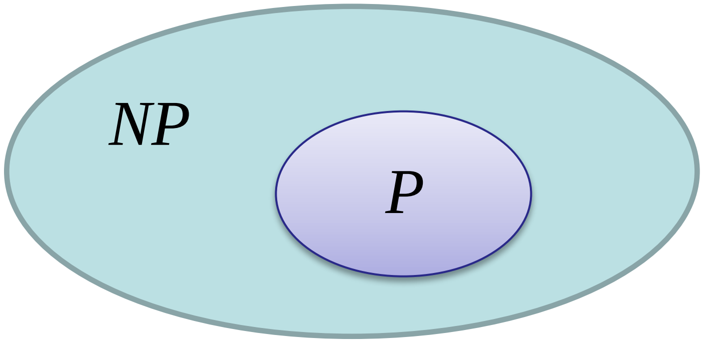
- existe um algoritmo determinista polinomial para o problema?

Investigar a complexidade relativa dos problemas da classe NP:

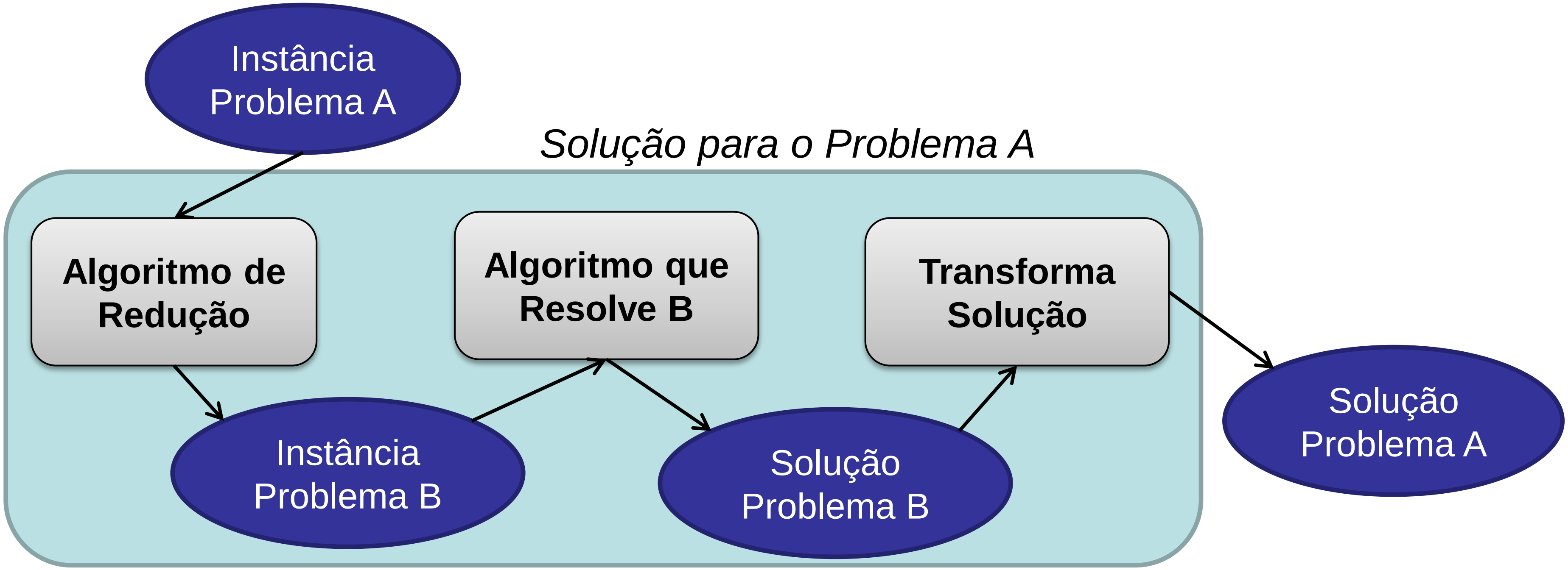
- Problema A é mais fácil ou mais difícil do que B?

Recorremos à ideia de *redução polinomial*:

- Mostra que A <u>não é mais difícil que</u> B ou que A é <u>polinomialmente redutível</u> ao problema B.



Redução de Problemas



$A \leq_p B$

Se o "**Algoritmo de Redução**", o "**Algoritmo que Resolve B**" e a "**Transformação de solução**" forem polinomiais, então podemos concluir algo sobre a solução do Problema A?

Redução de Problemas

Conclusões provenientes da redução:

Se A é polinomialmente redutível a B então A não é mais difícil do que B.

$$\mathbf{A} \leq_p \mathbf{B}$$

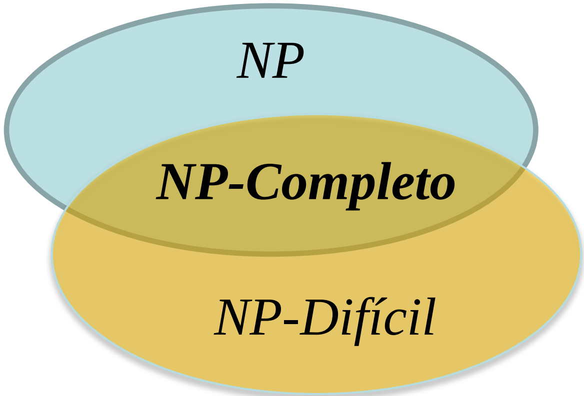
Cenário 1: sabe-se que B está na classe P.
Logo, A também deve estar na classe P.

Cenário 2: não se sabe se B está ou não em P,
mas sabe-se que A não está em P.
Como A não é mais difícil que B, então B deve estar fora de P.

NP-Completude

Em 1970 Cook e Levin publicaram um importante trabalho sobre a questão P *versus* NP – Teorema de Cook-Levin

- Se podemos determinar que um problema não é mais difícil do que outro, podemos separar os problemas mais difíceis em NP!
- Dizemos que um problema A é NP-Difícil se todos os problemas em NP não são mais difíceis do que A
- Logo, um problema NP-Difícil é tão difícil quanto, ou mais difícil, do que qualquer problema em NP.
- Entre a classe NP-Difícil, alguns problemas podem ser verificados em tempo polinomial (i.e. pertence a NP), ao passo que outros não.



NP-Completude

Um problema X é ***NP-Completo*** se:

1. O problema deve ser NP: $X \in NP$

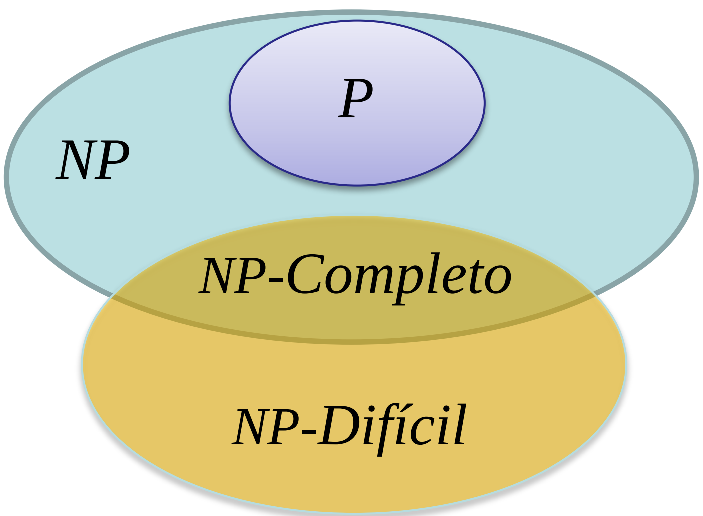
   a) *Conseguir um algoritmo não determinista que resolva o problema em tempo polinomial*

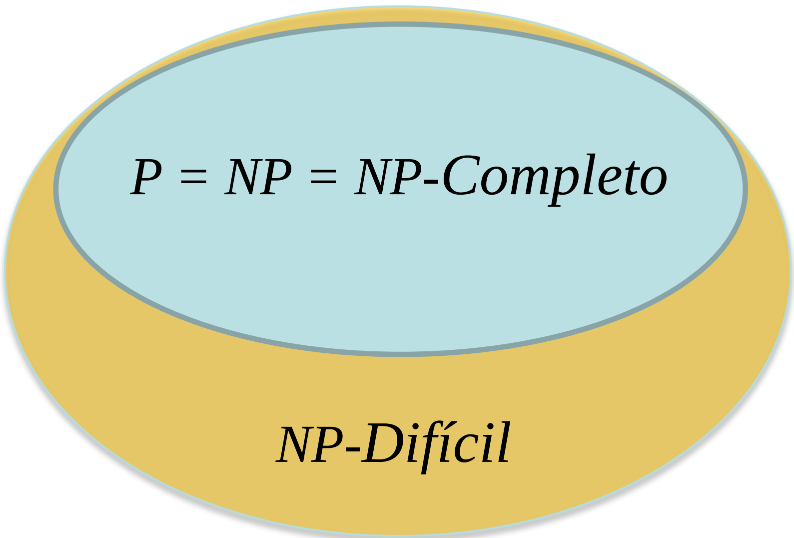
   b) *Conseguir um algoritmo determinista que **verifica** em tempo polinomial se uma resposta é verdadeira ou não (**certificado**)*

2. Fazer a redução de um problema NP-Completo (Y) conhecido para o problema X: $Y \leq_p X$ para todo $Y \in NP$

NP-Completude

- Essa distinção entre problemas mais difíceis em NP é extremamente útil para a questão P *versus* NP porque:
  - Para provar que P = NP, basta mostrar que um problema NP-Completo é resolvível em tempo polinomial para mostrar que P = NP
  - Para provar que P ≠ NP pode-se focar apenas nos problemas NP-Completos visto que eles são os mais difíceis em NP





NP-Completude

- Atualmente, provar que um problema é NP-Completo é uma forte evidência de sua não-polinomialidade!

- O ponto inicial com relação a NP-Completude veio através da primeira prova de que um problema é NP-Completo

Teorema de Cook-Levin: $SAT \in P \text{ sse } P = NP$

SAT – Problema da Satisfazibilidade

$$SAT = \{\langle \phi \rangle \mid \phi \text{ é } uma\ fórmula\ booleana\ satisfazível\}$$

(SAT) Satisfazibilidade de Fórmulas Booleanas

O problema da ***Satisfazibilidade de fórmulas booleanas*** consiste em determinar se existe uma atribuição de valores booleanos, para as variáveis que ocorrem na fórmula, de tal forma que o resultado seja ***verdadeiro***.

Um ***literal*** é uma variável proposicional ou sua negação.

Exemplo:

$$x_1 \wedge (x_2 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

Problema de Decisão: existe uma combinação de valores para x_1 e x_2 que satisfazem esta equação?

Complexidade algoritmo trivial: (2^n)

SAT ∈ NP-Completo

Classificando SAT como NP-Completo:

Passo 1: Algoritmo verificador (determinista e polinomial)

```
1. Para cada símbolo y da entrada w:
    i. Simule a operação booleana sempre que
       possível, armazenando os resultados parciais
       ou os símbolos que ainda não puderam ser
       simplificados
2. Considerando uma fórmula booleana válida,
   retorne o último valor booleano restante como
   resposta (V = aceite / F = rejeite)
```

$$x_1 \wedge (x_2 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$

Qual seria a complexidade do algoritmo?

SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Qualquer linguagem A em NP pode ser reduzida em tempo polinomial para o problema SAT.

Ideia: com base em A e uma cadeia w produzimos uma fórmula booleana ϕ que simula uma máquina NP para A sobre w.

Se a máquina aceita então ϕ possui uma atribuição que a satisfaz (computação de aceitação). Se a máquina não aceita então nenhuma atribuição satisfaz ϕ.

Lembre-se que uma fórmula booleana possui os operadores que formam a base de circuitos usados em computadores eletrônicos, então, o fato de projetar uma fórmula que simula uma MT não é surpreendente.

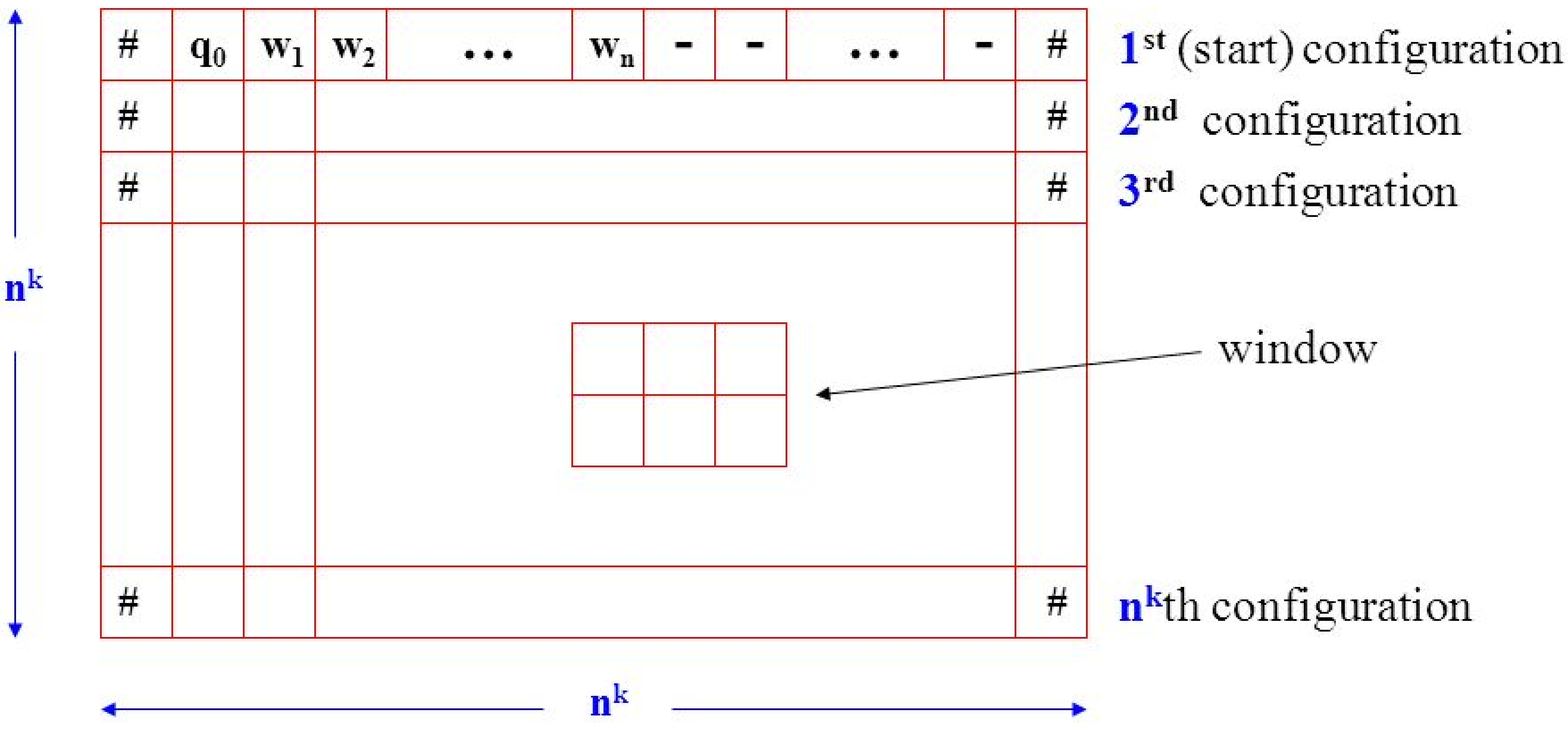
SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Ideia: construir um **tableau** para uma MTN N sobre uma entrada *w* ($|w| = n$) de tamanho $n^k \times n^k$ cujas linhas são configurações de um ramo da computação.

SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT



SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Ideia: construir um **tableau** para uma MTN N sobre uma entrada *w* ($|w| = n$) de tamanho $n^k \times n^k$ cujas linhas são configurações de um ramo da computação.

Dizemos que um **tableau** é de **aceitação** se qualquer linha dele for uma configuração de aceitação (i.e. algum caminho da MTN levou a um estado de aceitação para a entrada *w*).

A parte mais trabalhosa é projetar uma fórmula ϕ de modo que uma atribuição às variáveis que satisfaça ϕ corresponda a um **tableau** de aceitação para N com entrada *w*:

$$\phi_{célula} \wedge \phi_{início} \wedge \phi_{movimento} \wedge \phi_{aceita}$$

SAT ∈ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Ideia: $\phi_{célula} \wedge \phi_{início} \wedge \phi_{movimento} \wedge \phi_{aceita}$

Examinar o tamanho de ϕ para garantir seu tamanho polinomial

+ Tamanho *tableau*: n^{2k}

+ $\phi_{célula}$: cada célula terá *lit* variáveis (# símbolos em $\{Q \cup \Gamma \cup \{\#\}\}$)

Como *lit* depende somente da MTN N e não do comprimento da entrada *w*, então $|\phi_{célula}| = O(n^{2k})$

+ $\phi_{início}$: garante que a primeira linha seja uma configuração inicial

$|\phi_{início}|$ = tamanho de uma linha do *tableau* = $O(n^{k})$

SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Ideia: $\phi_{célula} \wedge \phi_{início} \wedge \phi_{movimento} \wedge \phi_{aceita}$

Examinar o tamanho de ϕ para garantir seu tamanho polinomial

+ $\phi_{movimento}$: representa os movimentos que podem ser realizados.
  Contém um fragmento de tamanho fixo da fórmula para cada célula do *tableau*, então $|\phi_{movimento}| = O(n^{2k})$

+ ϕ_{aceita}: representa linhas do tableau que são de aceitação.
  Contém um fragmento de tamanho fixo da fórmula para cada célula do *tableau*, então $|\phi_{aceita}| = O(n^{2k})$

SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT

Ideia: $\phi_{célula} \land \phi_{início} \land \phi_{movimento} \land \phi_{aceita}$

$$=$$

$$O(n^{2k}) + O(n^{k}) + O(n^{2k}) + O(n^{2k})$$

$$=$$

$$O(n^{2k})$$

[um polinômio!]

SAT $\in$ NP-Completo

Passo 2: MTND $\leq_p$ SAT (*passed*)

Existem materiais escritos ou em forma de vídeo que mostram com mais ou menos detalhas os passos desta redução.

Dois links de exemplo são divulgados abaixo:

http://www.inf.ufrgs.br/~prestes/Courses/Complexity/aula27.pdf

https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem

SAT $\in$ NP-Completo

Classificando SAT como NP-Completo:

Passo 1: Algoritmo de certificado (*passed*)

Passo 2: MTND $\leq_p$ SAT (*passed*)

Logo, provamos que SAT pertence ao conjunto de problemas NP-Completo!

Redução de Problemas

Relação entre Redução e Problemas NP-Completos:

Uma vez conhecido um problema NP-Completo, podemos usar *reduções polinomiais* para provar que algum problema X também é NP-Completo.

"se *Y* é um problema NP-Completo e *Y* não é mais difícil que um problema *X* (redução) então *X* também é NP-Completo"

$$\mathbf{Y} \leq_p \mathbf{X}$$

Ex: SAT $\leq_p$ X

A definição do passo 2, na verdade fala que todo problema pertencente a NP-Completo deve ser redutível em tempo polinomial ao problema X. Então por quê podemos reduzimos apenas um?

Exercícios

Verifique se as afirmações abaixo são **verdadeiras** ou **falsas**, justificando as **falsas**:

a) Se um problema NP for resolvido em tempo polinomial então P = NP.
b) Se um problema NP-Difícil (ou NP-Hard) for resolvido em tempo polinomial então P = NP.
c) Se P = NP então todos os problemas considerados NP-Difícil (ou NP-Hard) podem ser resolvidos em tempo polinomial.
d) Podemos afirmar que os problemas NP-Completo possuem apenas soluções em tempo exponencial ou maior.
e) Considerando um problema P1 que tem uma solução em tempo polinomial conhecida, e um problema P2 que é NP-Completo, apresentando uma redução que pode ser executada em tempo polinomial de P1 a P2 ($P1 \leq_p P2$) estamos provando que P = NP.
f) Se $P \cap$ NP-Completo $\neq \emptyset$ então P = NP.