Complexidade Computacional

Disclaimer

- Slides baseados nos slides dos professores
  - Carla N. Lintzmayer
  - Lehilton Pedrosa

Algoritmo de Tempo Polinomial

Um algoritmo é polinomial se o tempo de execução for limitado por $O(n^k)$ para alguma constante k

- Neste caso, dizemos que ele é um algoritmo eficiente
  - Não necessariamente é rápido na prática
    - mas exclui muitos dos algoritmos considerados lentos

Organizando os Problemas

Por que se preocupar com isso?

- desconhecemos algoritmos rápidos para vários problemas
- acreditamos que não há algoritmos eficientes para eles
- queremos saber quais deles têm algoritmos polinomiais

Antes, vamos discutir:

1. Como representar problemas?

Linguagens Formais

Cadeias e Linguagens

- Um <u>alfabeto</u> é um conjunto finito de elementos chamados <u>símbolos</u>

Ex

$\Sigma = \{0, 1\}$

$\Gamma = \{a, b, c, d, \ldots, z\}$

$\Omega = \{if, while, for, =\}$

Dado um alfabeto Σ, uma <u>cadeia</u> é uma sequência $w_1 w_2 \cdots w_n$, onde $w_i \in \Sigma$ para todo $1 \leq i \leq n$.

- Uma cadeia é uma sequência finita de símbolos de um alfabeto

Ex
- 01001 é uma cadeia sobre o alfabeto $\{0,1\}$
- abracadabra é uma cadeia sobre o alfabeto $\{a, b, \ldots, z\}$

- Cadeias também são chamadas de <u>strings</u> ou <u>palavras</u>

Comprimento de uma Cadeia

O comprimento de uma cadeia w, denotado por $|w|$, é o número de elementos na sequência.

Exemplo

- $w = maycon$ sobre o alfabeto $\{a, b, c, \cdots, z\}$
  - $|w| = 5$

- $\beta = 011011$ sobre o alfabeto $\{0, 1\}$
  - $|\beta| = 6$

A cadeia vazia, denotado por ε, é a cadeia de comprimento 0.

Dado um alfabeto Σ, denotamos por Σ^* o conjunto formado por todas as palavras possíveis de serem formadas com símbolos de Σ

Exemplo

$\Sigma = \{0, 1\}$

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

A cadeia vazia, denotado por ε, é a cadeia de comprimento 0.

Dado um alfabeto Σ, denotamos por Σ^* o conjunto formado por todas as palavras possíveis de serem formadas com símbolos de Σ

Exemplo

$\Sigma = \{0, 1\}$

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

contém a cadeia vazia

conj. infinito

Linguagem

Uma linguagem L sobre um alfabeto Σ é um subconj. de Σ^*, i.e.,

$$L \subseteq \Sigma^*.$$

Ex

$\Sigma = \{0, \dots, 9\}$

$L = \{0, 2, 4, \dots\} \subseteq \Sigma^*$

Linguagem dos números pares

$\Sigma = \{a, b, \dots, z\}$

$L = \{a, aba, abba, \dots\}$

Linguagem Palíndromos

Codificação

Uma **codificação** é o mapeamento utilizado para representar um objeto como uma palavra de Σ^*.

- Dado um objeto A, sua codificação é denotada por $\langle A \rangle$

Representando Problema de Decisão como Linguagem

A linguagem L correspondente a um problema de decisão Q é o conjunto das codificações de instâncias sim, isso é,

$$L = \{\langle I \rangle \in \{0,1\}^* : (I, \text{sim}) \in Q\}$$

Problema do Caminho mínimo

Entrada: um grafo G, vértices $u, v \in V(G)$, e $k \in \mathbb{R}_+$

Saída: sim, se existe um uv-caminho de comprimento no máximo k; não, caso contrário.

$$PATH = \{ \langle G, u, v, k \rangle : G \text{ é um grafo, } u, v \in V(G), k \in \mathbb{R}_+ \text{ e existe um } uv\text{-caminho de comprimento no máximo } k \}$$

- O problema de decidir se x é uma instância sim ou não pode ser visto como o problema de determinar se

$$\langle x \rangle \in L$$

- Vamos tratar o problema de decisão e a sua linguagem de forma intercambiável.

Tamanho Instância

O **tamanho de uma instância** $x \in \{0,1\}^*$, denotado por $|x|$, é o número de bits de x.

Linguagem Aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$

- pode ser que A termina ao receber x e devolve $A(x)=1$.
- pode ser que A termina ao receber x e devolve $A(x)=0$.
- pode ser que A não termina ao receber x.

Um algoritmo A **aceita** uma linguagem L se

$$L = \{ x \in \{0,1\}^* : A(x) = 1 \}.$$

Um algoritmo A **decide** L se para todo $x \in \{0,1\}^*$

- Se $x \in L$, então $A(x) = 1$ (A aceita x)
- Se $x \notin L$, então $A(x) = 0$ (A rejeita x)

Um algoritmo A **decide** L se para todo $x \in \{0,1\}^*$

- Se $x \in L$, então $A(x) = 1$ (A aceita x)
- Se $x \notin L$, então $A(x) = 0$ (A rejeita x)

Dizer que temos um algoritmo A que decide uma linguagem L é o mesmo que dizer que temos um algoritmo para o problema de decisão atrelado à L.

Classes

Classe P

AKA problemas

A classe P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.

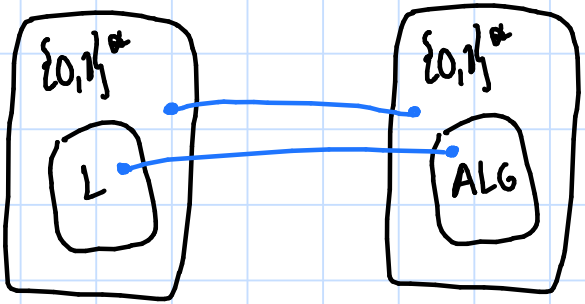
Em outras palavras, se $L \in P$, então

1. Existe algoritmo $A(x)$ que decide L
2. Esse algoritmo executa em tempo polinomial em $|x|$

A classe P é a que contém os problemas de decisão que podem ser resolvidos em tempo polinomial.

Como Mostramos que uma Linguagem Pertence à P?

- Exibimos um algoritmo que decide ela em tempo polinomial.
  - ↳ Resolve o problema em $O(n^k)$



- Seja $w \in \{0,1\}^*$

$w \in L \Rightarrow w \in ALG$ $\quad (L \subseteq ALG)$

$ALG(w) = 1 \Rightarrow w \in L$ $\quad (ALG \subseteq L)$

- No jargão do problema de decisão, fornecemos um algoritmo que resolve o problema em tempo polinomial.

Teo. PATH $\in$ P

Demonstração

Vamos mostrar que o seguinte algoritmo decide PATH

Vamos sempre assumir que dá para fazer em tempo polinomial

```
ALG-PATH(<G, u, v, k>) {
1   se <G, u, v, k> não é uma codificação válida
      de uma instância de PATH, devolva 0.
2   d, pred ← BFS-dist(G, u)
3   se d[v] ≤ k
4         devolva ⊥
5   senão
6         devolva 0
}
```

Teo. PATH $\in$ P

Vamos sempre assumir que dá para fazer em tempo polinomial

Demonstração (continuação)

```
ALG-PATH (<G, u, v, k>) {
 1  se <G, u, v, k> não é uma codificação válida
      de uma instância de PATH, devolva 0.
 2  d, pred ← BFS-dist(G, u)
 3  se d[v] ≤ k
 4        devolva 1
 5  senão
 6        devolva 0
}
```

Primeiro, note que esse algoritmo roda em tempo polinomial. Sabemos que a linha 2 roda em $O(E+V)$ e as outras linha executam em tempo polinomial. Portanto é um algoritmo polinomial.

Teo. PATH $\in$ P

Demonstração (continuação)

- Agora vemos mostrar que esse algoritmo decide PATH.
- Primeiro suponha que $\langle G, u, v, k \rangle \in$ PATH.
- Então existe um uv-caminho P tal que $|E(P)| \leq k$.
- Sabemos que $k \geq |E(P)| \geq dist(u, v) = d[v]$. Portanto, o Alg-PATH $(\langle G, u, v, k \rangle) = 1$, o que é o comportamento esperado.
- Agora suponha que Alg-PATH $(\langle G, u, v, k \rangle) = 1$.
- Então $\langle G, u, v, k \rangle$ é uma codificação válida de uma instância de PATH (pela linha 1)
- Como o algoritmo retorna 1, sabemos que o teste da linha 3 deu verdadeiro. e $d[v] \leq k$
- Pela correção de BFS-dist, G contém um uv-caminho P tal que $|E(P)| \leq k$.

□

Certificado

Considere uma linguagem L.

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$ de forma que verificar y permite concluir que $\langle x \rangle \in L$.
  - Normalmente y é a codificação de uma solução para x.
- Chamamos y de <u>certificado</u> para x.

Certificado para Path

- tome uma instância $x = \langle G, u, v, k \rangle$ de Path
  - se $x \in$ PATH, existe uv-caminho P de comprimento $\leq k$ em G
  - se $x \notin$ PATH, não existe uv-caminho P de comprimento $\leq k$ em G
- no primeiro caso, $y = \langle P \rangle$ é um certificado de que $x \in$ PATH

Verificador

Um **verificador** para uma linguagem L é um algoritmo que recebe uma instância x e uma sequência de bits y tal que

- Se $x \in L$, ele devolve 1 para algum certificado y
- Se $x \notin L$, ele devolve 0 independentemente de y

Verifica-PATH $(\langle G, u, v, k \rangle, \langle P \rangle)$

1 Se $\langle P \rangle$ não é a codificação de um caminho P, devolva 0

2 Se P não é um uv-caminho, devolva 0

3 Se P tem mais de k arestas, devolva 0

4 senão devolva 1

Tempo de Verificação

Queremos executar o verificador em tempo polinomial:

1. O tempo do verificador deve ser polinomial em $|x|$ e $|y|$
2. O tamanho do certificado $|y|$ deve ser polinomial em $|x|$

- Queremos diferenciar as tarefas de decidir e verificar
- Para certos problemas, decidir uma instância é difícil, mas pode ser que verificar uma solução seja fácil.

Ciclo Hamiltoniano

Um <u>ciclo hamiltoniano</u> em um grafo G é um ciclo que passa por todos os vértices.

- Se G contém um ciclo hamiltoniano, então dizemos que G é hamiltoniano

Problema do Ciclo Hamiltoniano

HAM-CYCLE = $\{\langle G \rangle : G$ é um grafo hamiltoniano$\}$

Escolha para certificado

- Ciclo Hamiltoniano $C \subseteq G$
- Uma sequência de vértices

Podemos decidir o problema do ciclo Hamiltoniano:

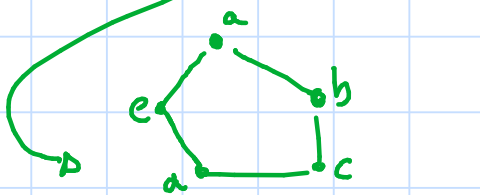
- Algoritmo trivial gasta $O(V!)$
- Os melhores algoritmos têm tempo $O(n^2 2^n)$

Verificar um ciclo C

- Podemos fazer em tempo linear

Verificando uma Solução

```
Verifica-HAM-ciclo (<G>, <C>)
1   Se <G> e <C> não são codificações válidas
    de um grafo G e de um ciclo C, devolva 0
2   Se C não contém todos os vértices de G,
    devolva 0
3   Para i ← 1 até C.comprimento - 1
4       u ← C[i]
5       v ← C[i+1]
6       Se uv ∉ E(G)
7           devolva 0
8   Devolva 1
```



C = a, b, c, d, e, a

1 2 3 4 5 6

Linguagem Verificada

Um algoritmo V verifica uma linguagem L se

$$L = \{ x \in \{0,1\}^* : \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1 \}$$

Em outras palavras

- se $x \in L$, então existe um certificado y tal que $V(x,y) = 1$
- se $x \notin L$, então não existe um certificado y tal que $V(x,y) = 1$

Classe NP

A classe NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe um algoritmo V que verifica L em tempo polinomial.

Em outras palavras:

- Existe um algoritmo $V(x,y)$ que verifica L.
- Esse algoritmo executa em tempo polinomial em $|x|$ e $|y|$.
- Para cada $x \in L$, existe um certificado y polinomial em $|x|$.

A classe NP contém os problemas de decisão que admitem um certificado para instâncias sim que podem ser verificados de de forma rápida.

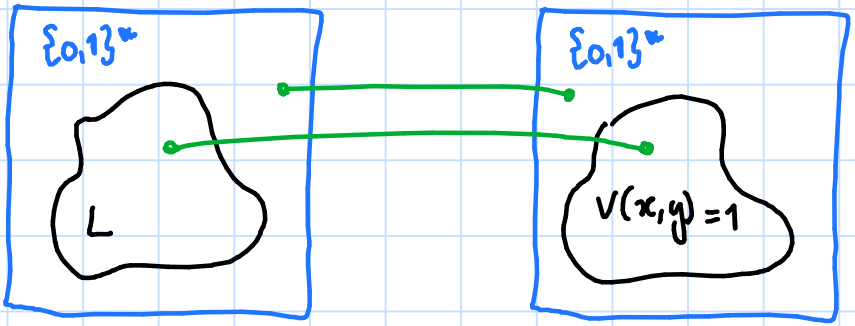
Determinando se o Problema é NP

- Dado um problema L, devemos seguir esses passos

1. identifique um certificado de tamanho polinomial para L
2. Construa um algoritmo verificador $V(x,y)$ polinomial
3. Demonstre que

$x \in L$ se e somente se $V(x,y) = 1$

onde y é uma cadeia de bits.



$x \in L \Rightarrow V(x,y)=1$

$(L \subseteq V(x,y)=1)$

$x \in V(x,y)=1 \Rightarrow x \in L$

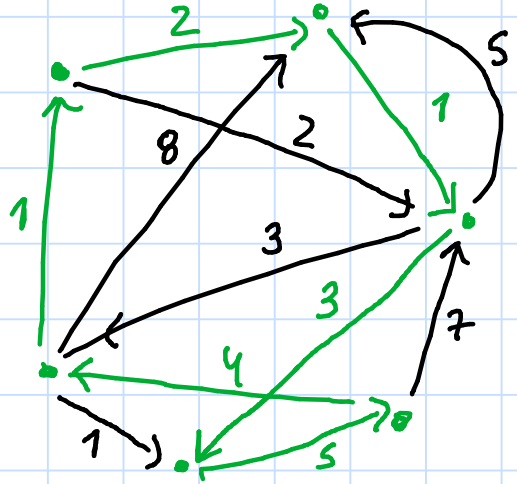
$(V(x,y)=1 \subseteq L)$

Problema TSP (Popularmente conhecido como o problema do caixeiro viajante)

Entrada: $\langle D, w, k \rangle$, onde D é um digrafo, $w: E(D) \to \mathbb{R}$ e $k \in \mathbb{R}$

Saída: sim, se existe um ciclo Hamiltoniano C em D tal que $w(D) \leq k$.

não, caso contrário.



$K = 22$

$w(C) = 1 + 2 + 1 + 3 + 5 + 4$

$= 16$

$$TSP = \left\{ \langle G, w, k \rangle : \begin{array}{l} \text{o grafo } G \text{ contém um ciclo} \\ \text{Hamiltoniano } c \text{ tal que } w(c) \leq k \end{array} \right\}$$

Teo. $TSP \in NP$

próximo slide

Lema A TSP $\in$ NP

Demonstração

Podemos usar como certificado para o TSP a sequência dos vértices de um circuito Hamiltoniano $C = u_1, u_2, u_3, \ldots, u_n, u_1$ tal que $w(C) \leq k$. Assim, a verificação pode ser feita em tempo polinomial pelo seguinte algoritmo

```
Verifica-TSP(<D,w,k>, <C>) {
1  Se <D,w,k> não for uma codificação válida de
   uma instância do TSP, devolva 0
2  Se <C> não for uma codificação válida de um
   ciclo gerador C de D, devolva 0
3  Se w(C) > k devolva 0
}  Devolva 1
```

□

Lema A TSP ∈ NP

Demonstração (continuação)

- Agora, vamos mostrar que o algoritmo de fato verifica a linguagem TSP
- Seja $\langle G, w, k \rangle \in$ TSP.
- Então existe um ciclo Hamiltoniano $C \subseteq G$ tal que $w(C) \leq k$
- Assim, Verifica-TSP$(\langle G, w, k \rangle, \langle C \rangle) = 1$.

Veja que exibimos uma cadeia de bits que faz com que o verificador retorne 1

- Agora, suponha que Verifica-TSP$(x, y) = 1$ para as cadeias $x \in \{0,1\}^*$, $y \in \{0,1\}^*$.
- Pela linha 1 do Algoritmo $x = \langle D, w, k \rangle$ é uma codificação válida de uma palavra de TSP

Lema A TSP $\in$ NP

Demonstração (continuação)

- Pela linha 2 do algoritmo $y = \langle C \rangle$ é uma codificação válida de um ciclo gerador C de D.
- Como o algoritmo não retornou 0 na linha 3, sabemos que $w(C) \leq K$
- Assim, D é um grafo que possui um ciclo gerador C tal que $w(C) \leq K$. Portanto $\langle G, w, K \rangle \in$ TSP

□

$P \subseteq NP$

Teo $P \subseteq NP$

Demonstração

- Seja $L \in P$. Então existe um algoritmo A polinomial que decide L
- Vamos construir um verificador para L

verificaL (x, y)
devolva $A(x)$

- Se $x \in L$, seja $y = \varepsilon$, e note que verificaL$(x,y) = 1$, já que A decide L
- Se verificaL$(x,y) = 1$, então $A(x) = 1$ e, portanto, $x \in L$.

P vs NP

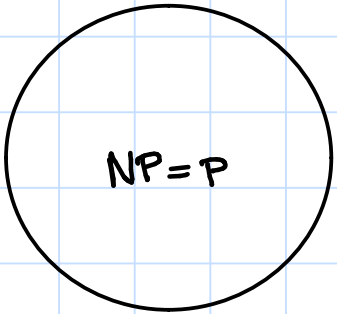
Teo $P \subseteq NP$

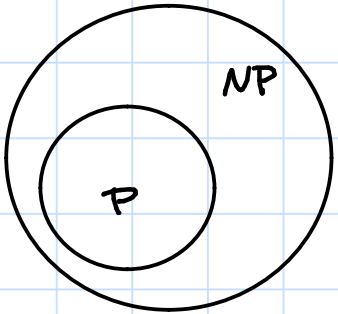
Não sabemos se $NP \subseteq P$, i.e., se $P = NP$

- i.e. não sabemos se os problemas que são fáceis de decidir são os problemas que são fáceis de verificar.
- Existe um prêmio de 10^6 dólares para quem resolver esse problema.

Classes de Complexidade

Possíveis configurações





Reduções de Karp

Estamos interessados em reduções em que:

1. A transformação da entrada f leve tempo polinomial
2. <u>não</u> há transformação da saída

Redução de Karp

A linguagem L_1 é reduzível em tempo polinomial para L_2 se

- existe algoritmo $f: \{0,1\}^* \to \{0,1\}^*$
- $f(x)$ leva tempo polinomial em $|x|$
- $x \in L_1 \iff f(x) \in L_2$

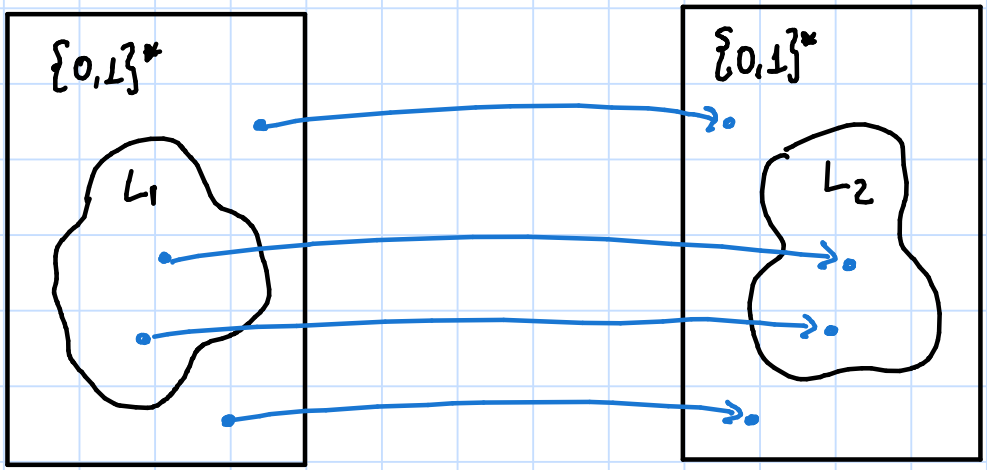
lemos "L_1 reduz em tempo polinomial a L_2"

escrevemos $L_1 \leq_p L_2$ (ou $L_1 \leq_{poli} L_2$)

Redução de Karp

A linguagem L_1 é redutível em tempo polinomial para L_2 se

- existe algoritmo $f: \{0,1\}^* \to \{0,1\}^*$
- $f(x)$ leva tempo polinomial em $|x|$
- $x \in L_1 \iff f(x) \in L_2$



Teo Considere linguagens $L_1, L_2 \subseteq \{0,1\}^*$ tais que $L_1 \leq_p L_2$
Se $L_2 \in P$, então $L_1 \in P$.

Demonstração

- Suponha que $L_2 \in P$ e seja ALG-L_2 um algonitmo que decide L_2 em tempo polinomial
- Seja f um algonitmo que reduz L_1 para L_2 em tempo polinomial
- Seja $ALG\text{-}L_1(x) = ALG\text{-}L_2(f(x))$
- Como f é uma redução temos que $x \in L_1$ se csomente se $f(x) \in L_2$
- Portanto, ALG-L_1 decide L_1 em tempo polinomial. □

```
ALG-L1( x )
    <z> ← f( x ) ] Poli
    Devolva ALG-L2( z ) ] Poli
```

Classe NP-difícil

A classe NP-difícil é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que $L' \leq_p L$ para todo $L' \in NP$.

A classe NP-completo é o conjunto de linguagens $L \subseteq \{0,1\}^*$ tais que $L \in NP$ e $L \in$ NP-difícil.

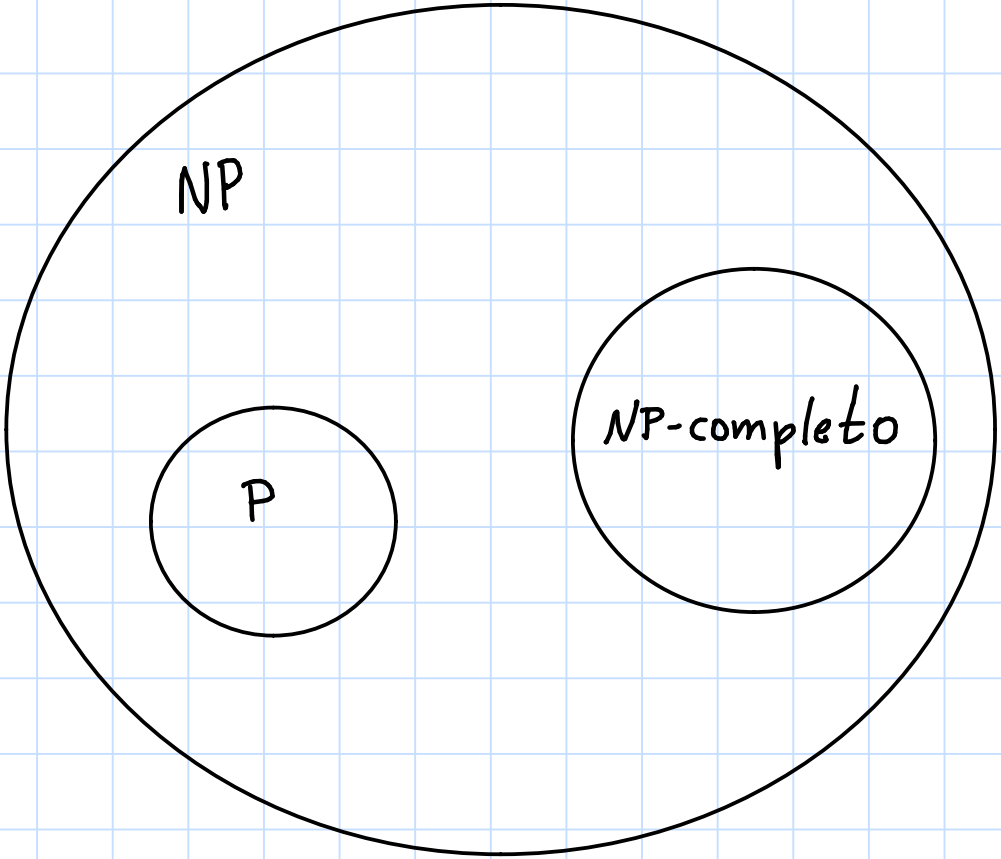
Teo. Se existe um algoritmo que decide $L \in$ NP-completo em tempo polinomial, então $P = NP$.

Demonstração

- Suponha que existe $L \in P \cap$ NP-completo
- Como $L \in$ NP-completo, para toda linguagem $L' \in NP$, temos que $L' \leq_p L$
- Como $L \in P$, temos que $L' \in P$ pelo teorema anterior.
- Logo $NP \subseteq P$ e, portanto, $NP = P$. □

Cor. Se existe um problema $L \in NP$ tal que $L \notin P$, então NP-completo $\cap P = \emptyset$.

Como acreditamos que é



Por que acreditamos que não há algoritmo rápido para problemas NP-difíceis?

- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin)
- bem antes, vários desses problemas já eram estudados
- Desde então, mostrou-se que inúmeros problemas importantes também são NP-completos ou NP-difíceis
- Vários pesquisadores estudaram seus problemas NP-completos preferidos, mas ninguém descobriu qualquer algoritmo polinomial.
- basta que um algoritmo NP-difícil tenha algoritmo de tempo polinomial para que <u>todos</u> os problemas em NP tenham algoritmos de tempo polinomial.

O que fazer se o problema for NP-difícil?

Se soubermos que o problema é NP-difícil, podemos concentrar nossos esforços em busca de

- Algoritmo para instâncias pequenas
  - backtracking
  - enumeração
  - Programação semidefinida
  - Programação Linear Inteira
  - Algoritmos Parametrizados
  - Programação por Restrição
- Soluções aproximadas
  - Heurísticas
  - Metaheurísticas
  - Algoritmos de Aproximação
    $OPT(I) \leq A(I) \leq 2OPT(I)$
- Algoritmos eficientes e exatos, para casos particulares do problema.

A classe NP-Completo não é vazia

- Será que realmente existe um problema NP-completo?
  - Cook e Levin responderam que sim (independentemente)
    - Eles mostraram que SAT é NP-Completo.

Teo de Cook-Levin
SAT é NP-completo

Satisfabilidade

Considere uma fórmula booleana

- Contém um conjunto de variáveis booleanas
- é escrita usando os seguintes operadores:
  1. negação ($\neg$)
  2. Conjunção ($\wedge$)
  3. Disjunção ($\vee$)
  4. Implicação ($\rightarrow$)
  5. Equivalencia ($\leftrightarrow$)

Problema da Satisfabilidade (SAT)

Entrada: uma fórmula booleana

Saída: sim, se existe uma atribuição de valores lógicos às variáveis da fórmula tal que a fórmula dê verdadeiro.
não, caso contrário.

Fórmula Satisfazível

Uma fórmula é satisfazível se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo

$$f = ((x_1 \to x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

atribuição: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$

$$\begin{aligned} f &= ((0 \to 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \\ &= 1 \wedge 1 \\ &= 1 \end{aligned}$$

Linguagem Correspondente

A linguagem SAT é aquela que contém fórmulas booleanas com atribuição verdadeira.

$SAT = \{\langle f \rangle : f \text{ é uma fórmula booleana satisfazível}\}$

Como Mostrar que um Problema é NP-difícil?

temos duas possibilidades para mostrar que um problema Q é NP-difícil:

1. Mostrar que $L \leq_p Q$ para todo $L \in NP$
2. Mostrar que $L \leq_p Q$ para algum $L \in$ NP-Difícil.

Normalmente usamos a segunda opção

Como Mostrar que um Problema é NP-Completo?

Para provar que um problema Q é NP-completo, fazemos:

1. Provamos que $Q \in NP$.
2. Provamos que $Q \in$ NP-difícil.

Teo Considere uma linguagem Q e seja $L \in$ NP-difícil. Se $L \leq_p Q$, então $Q \in$ NP-difícil.

Demonstração

Como L é NP-difícil, para todo $L' \in NP$, vale que $L' \leq_p L$

Assim $L' \leq_p L$ e $L \leq_p Q$, o que implica $L' \leq_p Q$

Portanto $Q \in$ NP-difícil □

Além disso, se $Q \in NP$, então Q é NP-Completo.

Fórmula 3-CNF

Dizemos que uma fórmula booleana ϕ sobre um conj. de variáveis $V = \{x_1, x_2, \ldots, x_n\}$ é 3-CNF se

$$\phi = c_1 \wedge c_2 \wedge c_3 \wedge \cdots \wedge c_m,$$

onde $c_i = (l_1^i \vee l_2^i \vee l_3^i)$, para todo $i = 1, 2, \ldots, m$, e l_j^i é um literal tal que $l_j^i = x$ ou $l_j^i = \neg x$, onde $x \in V$.

→ chamamos isso de cláusula

Exemplo

$$\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

cláusula

literais

Problema 3-SAT

Entrada: ϕ, onde ϕ é uma fórmula 3-CNF contendo literais de variáveis booleanas $x_1, x_2, \ldots, x_n$.

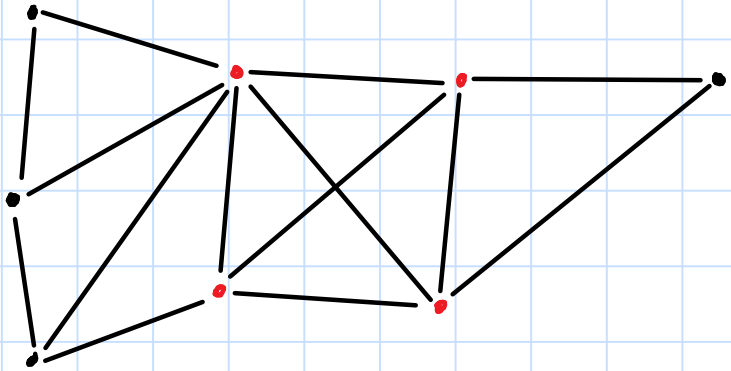
Saída: Sim existe uma atribuição de valores $x_1, x_2, \ldots x_n$ tal que ϕ seja satisfatível. Não caso contrário.

Teo 3-SAT é NP-completo.

Problema Clique

Entrada: $\langle G, k \rangle$, onde G é um grafo e k é um inteiro positivo

Saída: sim se existe uma clique $S \subseteq V(G)$ tal que $|S| \geq k$ e não, caso contrário.



$k = 3$

Teo Clique é NP-completo

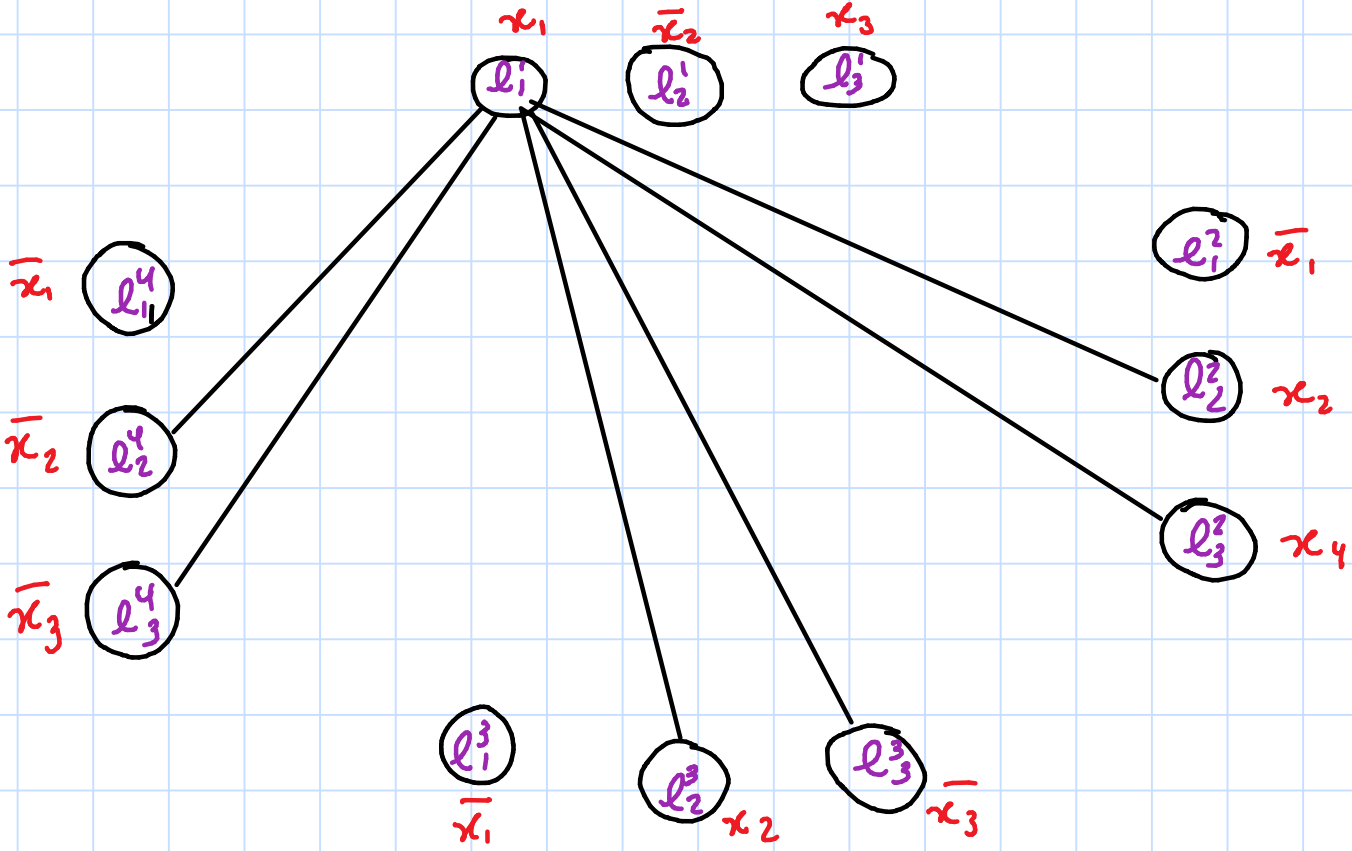
Demonstração

- Primeiro mostraremos que CLIQUE está em NP e depois faremos uma redução polinomial de 3-SAT para CLIQUE
- Clique está em NP pois, dados $\langle G, k \rangle$ e um centificado $\langle S \rangle$, onde $S \subseteq V(G)$ é uma clique tal que $|S| \geq k$, é fácil escrever um algoritmo para verificar se S é realmente tal clique: basta verificar se há uma aresta entre todos os pares de vértices em S, que pode ser feito em $O(V^2 \cdot E)$, e contar o número de elementos de S, que pode ser feito em $O(V)$.

Quando for provar que um problema está em NP no meio da prova de NP-completo, você pode ser menos detalhista, para a prove não ficar muito longa

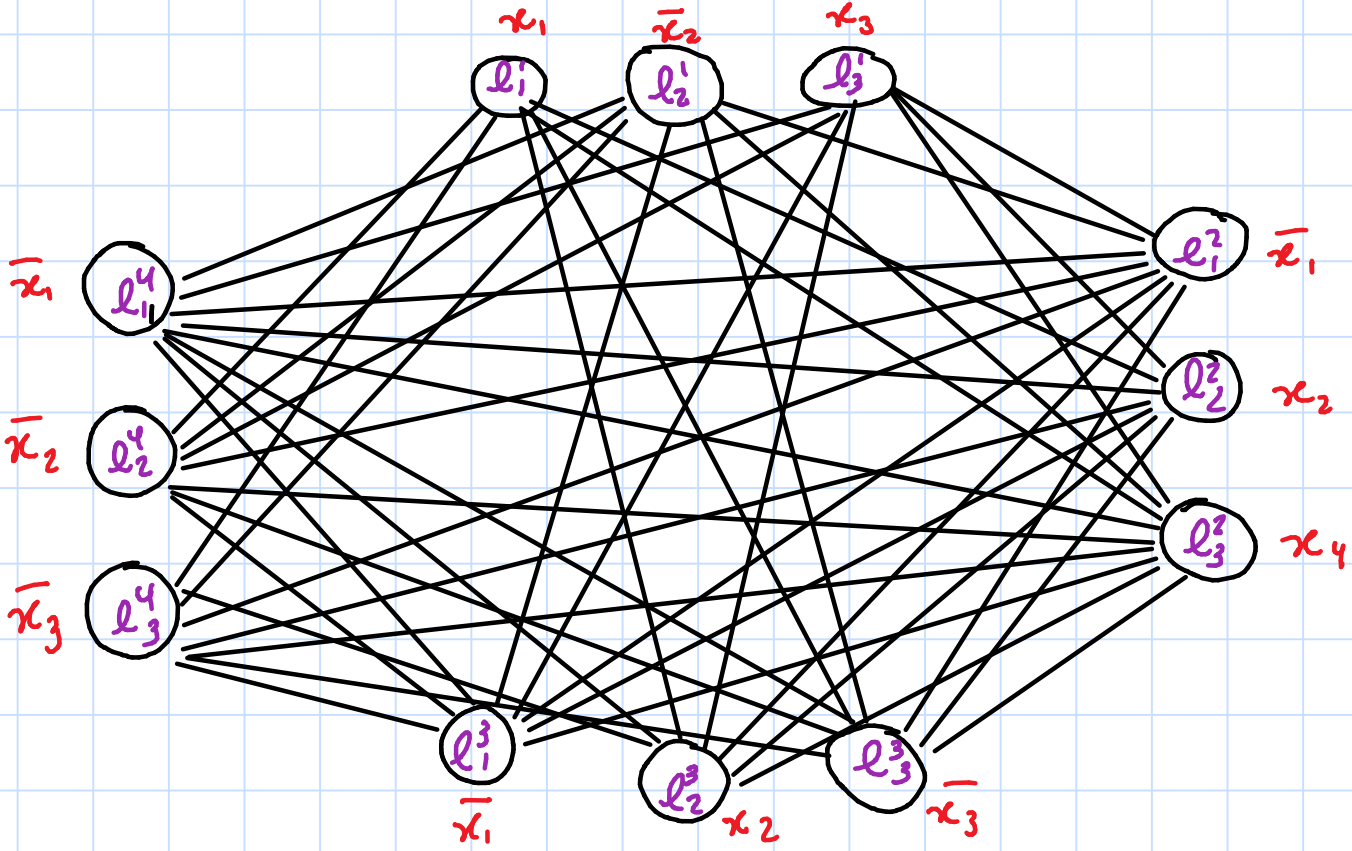
Dado $\phi = \underbrace{(x_1^1 \vee \bar{x}_2^1 \vee x_3^1)}_{c_1} \wedge \underbrace{(\bar{x}_1^2 \vee x_2^2 \vee x_4^2)}_{c_2} \wedge \underbrace{(\bar{x}_1^3 \vee x_2^3 \vee \bar{x}_3^3)}_{c_3} \wedge \underbrace{(\bar{x}_1^4 \vee \bar{x}_2^4 \vee \bar{x}_3^4)}_{c_4}$

construa



Dado $\phi = \underbrace{(x_1^1 \vee \bar{x}_2^1 \vee x_3^1)}_{c_1} \wedge \underbrace{(\bar{x}_1^2 \vee x_2^2 \vee x_4^2)}_{c_2} \wedge \underbrace{(\bar{x}_1^3 \vee x_2^3 \vee \bar{x}_3^3)}_{c_3} \wedge \underbrace{(\bar{x}_1^4 \vee \bar{x}_2^4 \vee \bar{x}_3^4)}_{c_4}$

construa



Teo Clique é NP-completo

Demonstração (continuação)

- Agora considere uma fórmula 3-CNF ϕ com m cláusulas e conjunto de variáveis $V = \{x_1, x_2, \ldots, x_m\}$, ou seja, $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, onde $c_i = (l_1^i \vee l_2^i \vee l_3^i)$ para $i = 1, \ldots, m$ e l_j^i é uma variável de V ou a negação de uma variável de V para todo $i = 1, \ldots, m$ e $j = 1, 2, 3$.

- Vamos construir um grafo $G(\phi)$ com $3m$ vértices: para cada cláusula adicionamos 3 vértices, um para cada literal. Assim

$$V(G(\phi)) = \bigcup_{i=1}^{m} \{l_1^i, l_2^i, l_3^i\}$$

Teo Clique é NP-completo

Demonstração (continuação)

- Sejam l_j^i e l_p^k dois vértices de $G(\phi)$. Vamos adicionar uma aresta entre esses dois vértices em $G(\phi)$ se $i \neq k$ e l_j^i não for a negação do literal l_p^k (ou seja, se (a) $l_j^i = l_p^i$ ou (b) $l_j^i = x$ e $l_p^k = \bar{x}$, então não colocamos a aresta).
- Nossa redução será

```
f(<φ>) {
        Seja m o número de literais em φ
        Devolve <G(φ), m>
}
```

Teo Clique é NP-completo

Demonstração (continuação)

- Não é difícil perceber que f executa em tempo polinomial.
- Resta verificar se $\langle \phi \rangle \in$ 3-SAT sse $f(\langle \phi \rangle) \in$ CLIQUE
- Suponha que $\langle \phi \rangle \in$ 3-SAT.

C_1 C_2 C_3 C_4

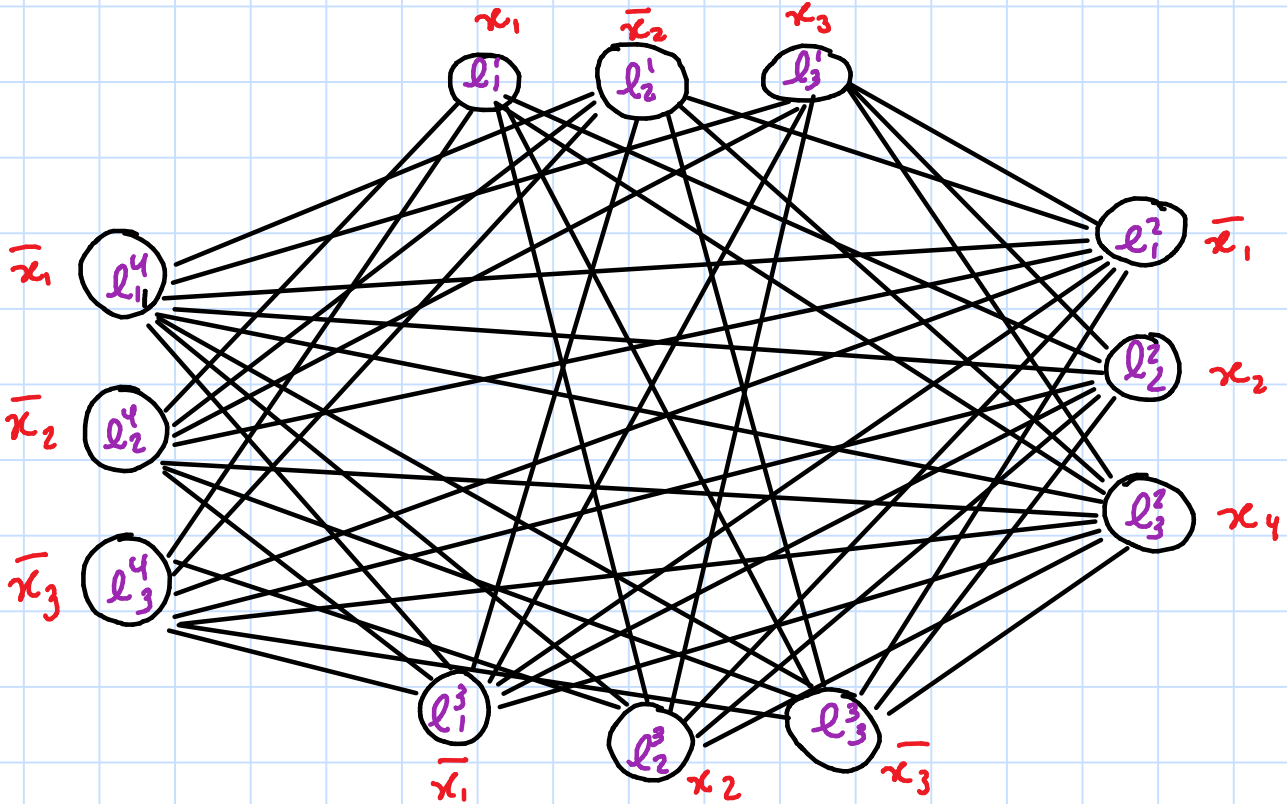
$\phi = (\underset{1}{x_1} \vee \underset{0}{\overline{x_2}} \vee \underset{0}{x_3}) \wedge (\underset{0}{\overline{x_1}} \vee \underset{1}{x_2} \vee \underset{1}{x_4}) \wedge (\underset{0}{\overline{x_1}} \vee \underset{0}{x_2} \vee \underset{1}{\overline{x_3}}) \wedge (\underset{0}{\overline{x_1}} \vee \underset{0}{\overline{x_2}} \vee \underset{1}{\overline{x_3}}) = 1$

$x_1 = 1$

$x_2 = 1$

$x_3 = 0$

$x_4 = 1$



C_1 C_2 C_3 C_4

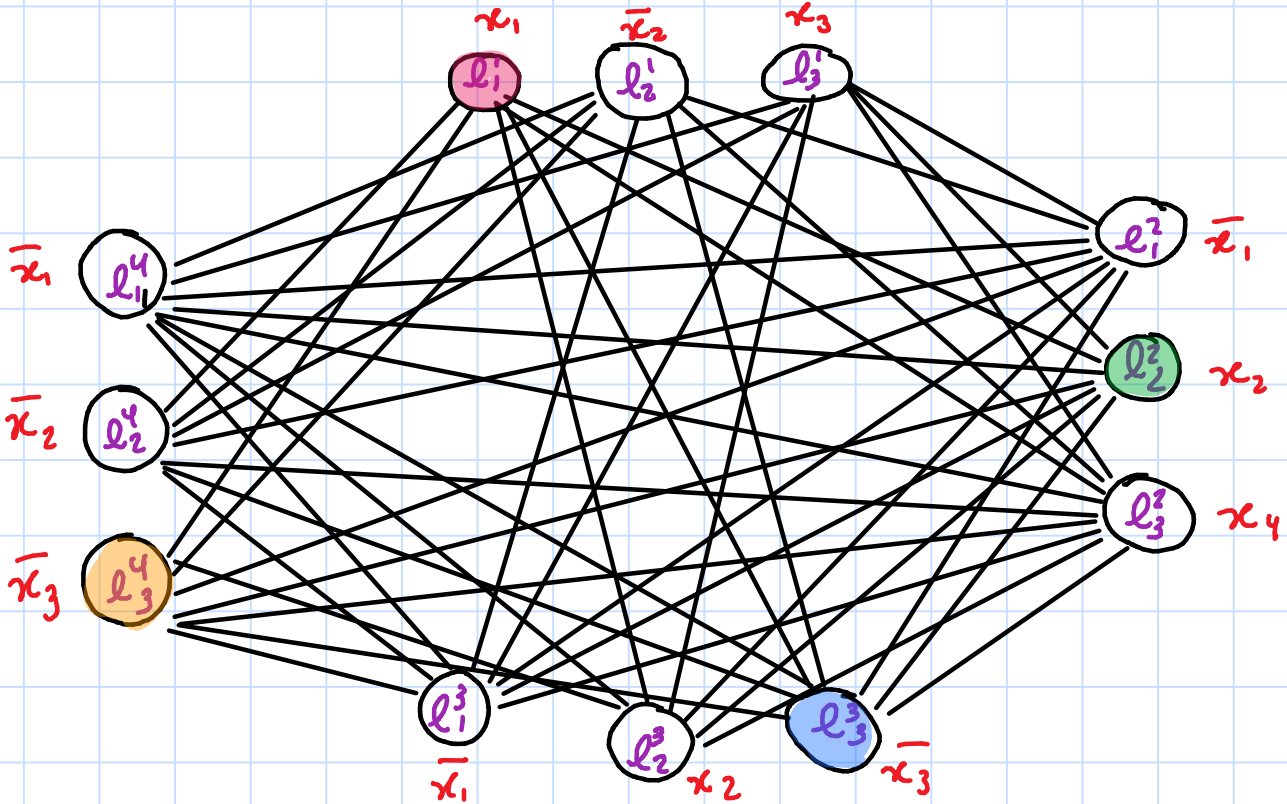
$$\phi = (\underbrace{x_1}_{1} \vee \underbrace{\bar{x}_2}_{0} \vee \underbrace{x_3}_{0}) \wedge (\bar{x}_1 \vee \underbrace{x_2}_{1} \vee \underbrace{x_4}_{1}) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \underbrace{\bar{x}_3}_{1}) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \underbrace{\bar{x}_3}_{1}) = 1$$

$x_1 = 1$

$x_2 = 1$

$x_3 = 0$

$x_4 = 1$



C_1 C_2 C_3 C_4

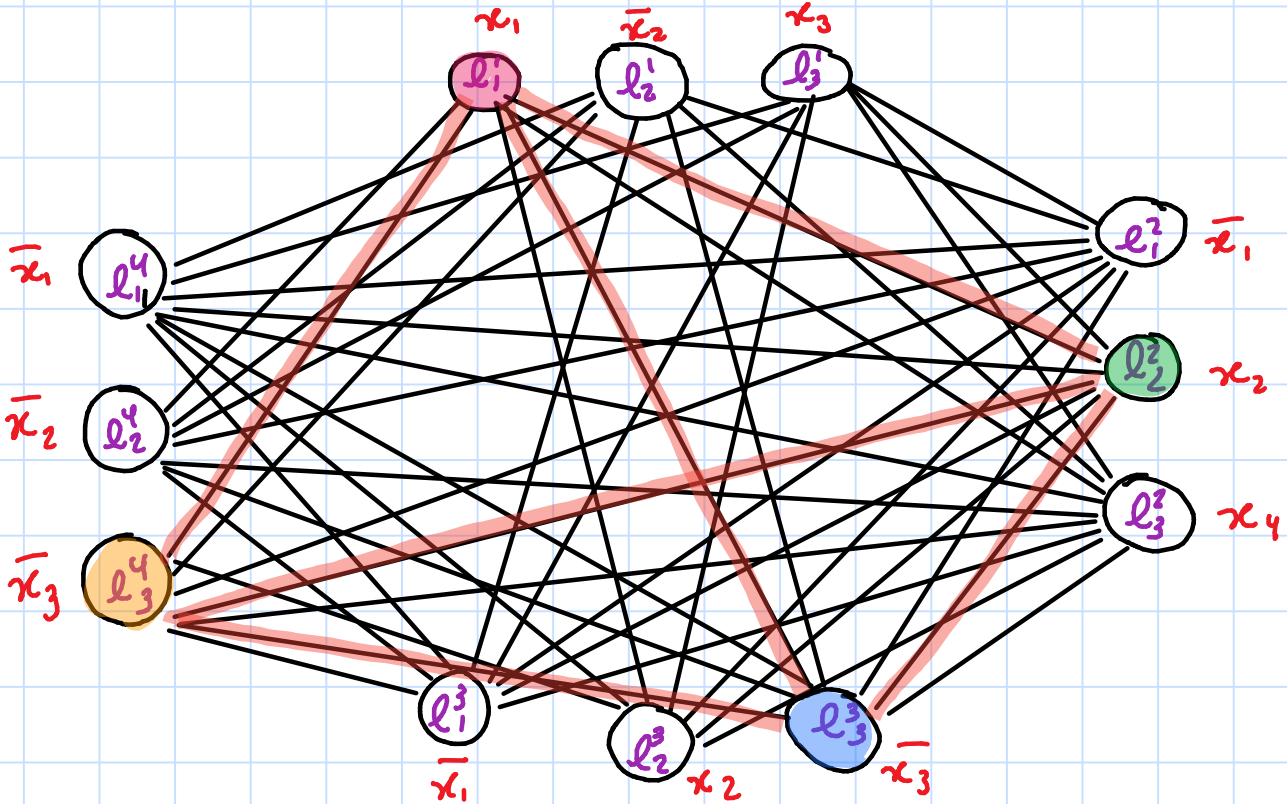
$\phi = (\underbrace{x_1}_{1} \vee \underbrace{\bar{x}_2}_{0} \vee \underbrace{x_3}_{0}) \wedge (\underbrace{\bar{x}_1}_{0} \vee \underbrace{x_2}_{1} \vee \underbrace{x_4}_{1}) \wedge (\underbrace{\bar{x}_1}_{0} \vee \underbrace{x_2}_{0} \vee \underbrace{\bar{x}_3}_{1}) \wedge (\underbrace{\bar{x}_1}_{0} \vee \underbrace{\bar{x}_2}_{0} \vee \underbrace{\bar{x}_3}_{1}) = 1$

$x_1 = 1$

$x_2 = 1$

$x_3 = 0$

$x_4 = 1$



C_1 C_2 C_3 C_4

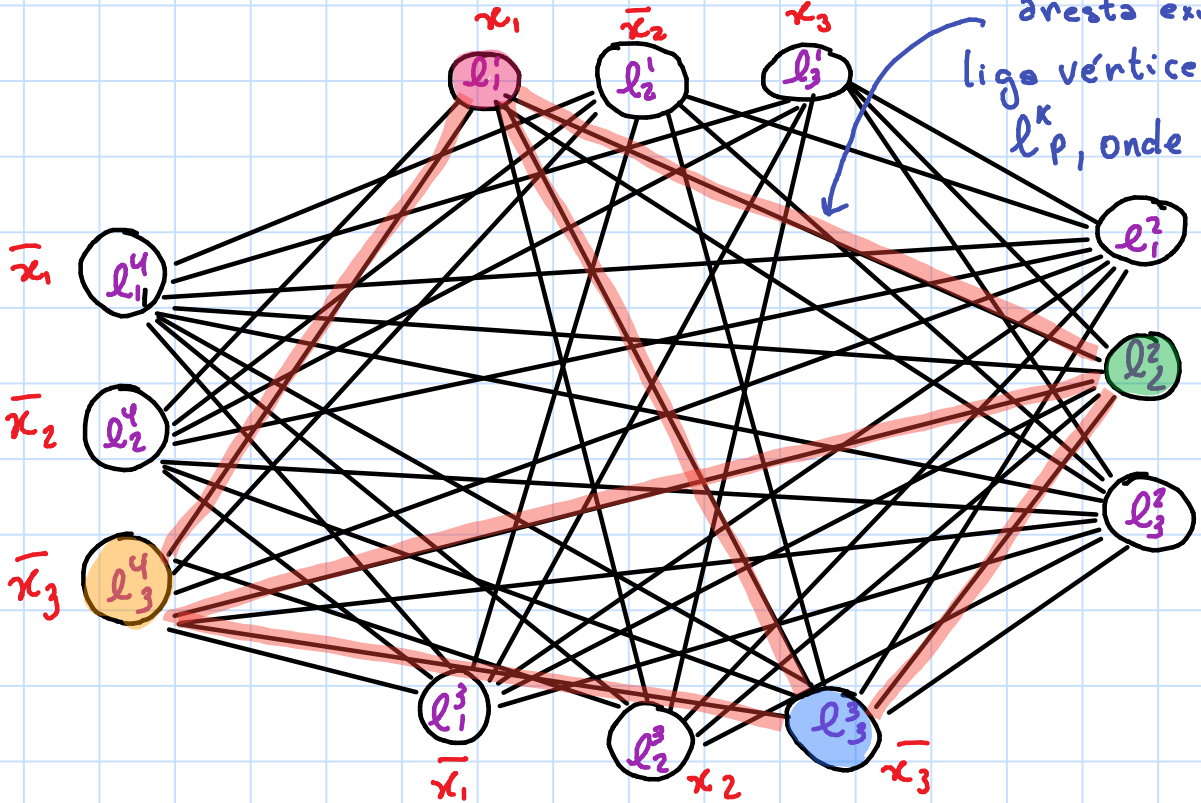
$\phi = (\underbrace{x_1}_{1} \vee \underbrace{\overline{x_2}}_{0} \vee \underbrace{x_3}_{0}) \wedge (\overline{x_1} \vee \underbrace{x_2}_{1} \vee \underbrace{x_4}_{1}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \underbrace{\overline{x_3}}_{1}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \underbrace{\overline{x_3}}_{1}) = 1$

$x_1 = 1$

$x_2 = 1$

$x_3 = 0$

$x_4 = 1$



aresta existe pois liga vértices l^i_j e l^k_p, onde $i \neq k$ e l^i_j não é a negação de l^k_p.

Teo Clique é NP-completo

Demonstração (continuação)

- Seja $\psi : V \to \{0,1\}$ uma atribuição de valores lógicos à ϕ de forma que ϕ seja satisfazível
- Então, ao menos um Literal em cada cláusula C_i deve ter avaliado para 1. Para cada cláusula C_i, seja t_i um literal dessa cláusula que tenha avaliado para 1.
- Seja $S = \{t_1, t_2, \ldots, t_m\}$
- Note que $|S| = m$. Agora, vamos argumentar que S é uma clique em $G(\phi)$
- Como todos os elementos de S são de cláusulas distintas,

Teo Clique é NP-completo

Demonstração (continuação)

pela construção de $G(\phi)$, a única possibilidade para não haver uma aresta entre dois vértices $t_i, t_j \in S$ seria se $t_i = x$ e $t_j = \bar{x}$, mas neste caso, não seria possível que esses dois literais tivessem avaliado para 1. Assim, S é uma clique em $G(\phi)$ e $f(\langle\phi\rangle) = \langle G(\phi), m\rangle \in$ CLIQUE.

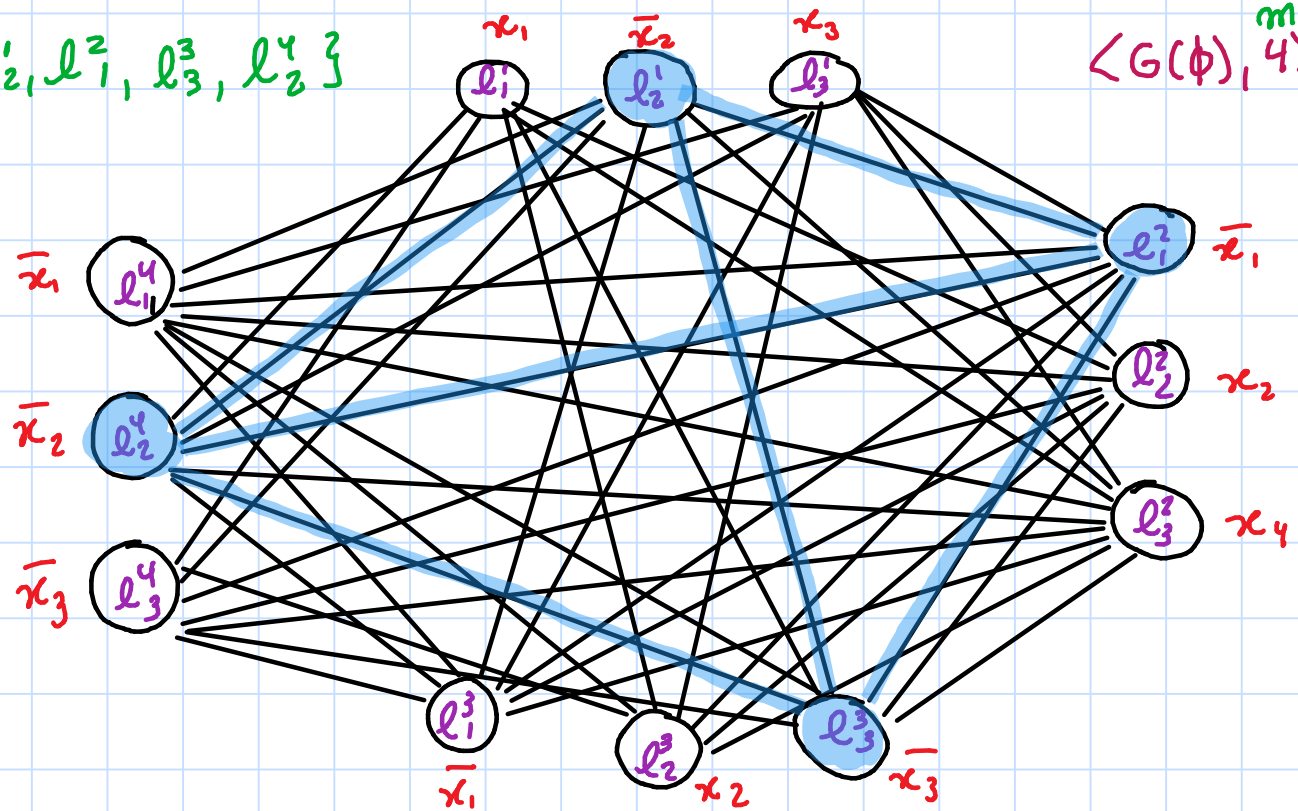
- Agora, suponha que $f(\langle\phi\rangle) = \langle G(\phi), m\rangle \in$ CLIQUE.

C_1 C_2 C_3 C_4

$\phi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{\ell_1^1\ \ell_2^1\ \ell_3^1} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee x_4)}_{\ell_1^2\ \ell_2^2\ \ell_3^2} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}_{\ell_1^3\ \ell_2^3\ \ell_3^3} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})}_{\ell_1^4\ \ell_2^4\ \ell_3^4}$

$S = \{ \ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4 \}$

$\langle G(\phi), m \rangle$ $m = 4$



C_1 C_2 C_3 C_4

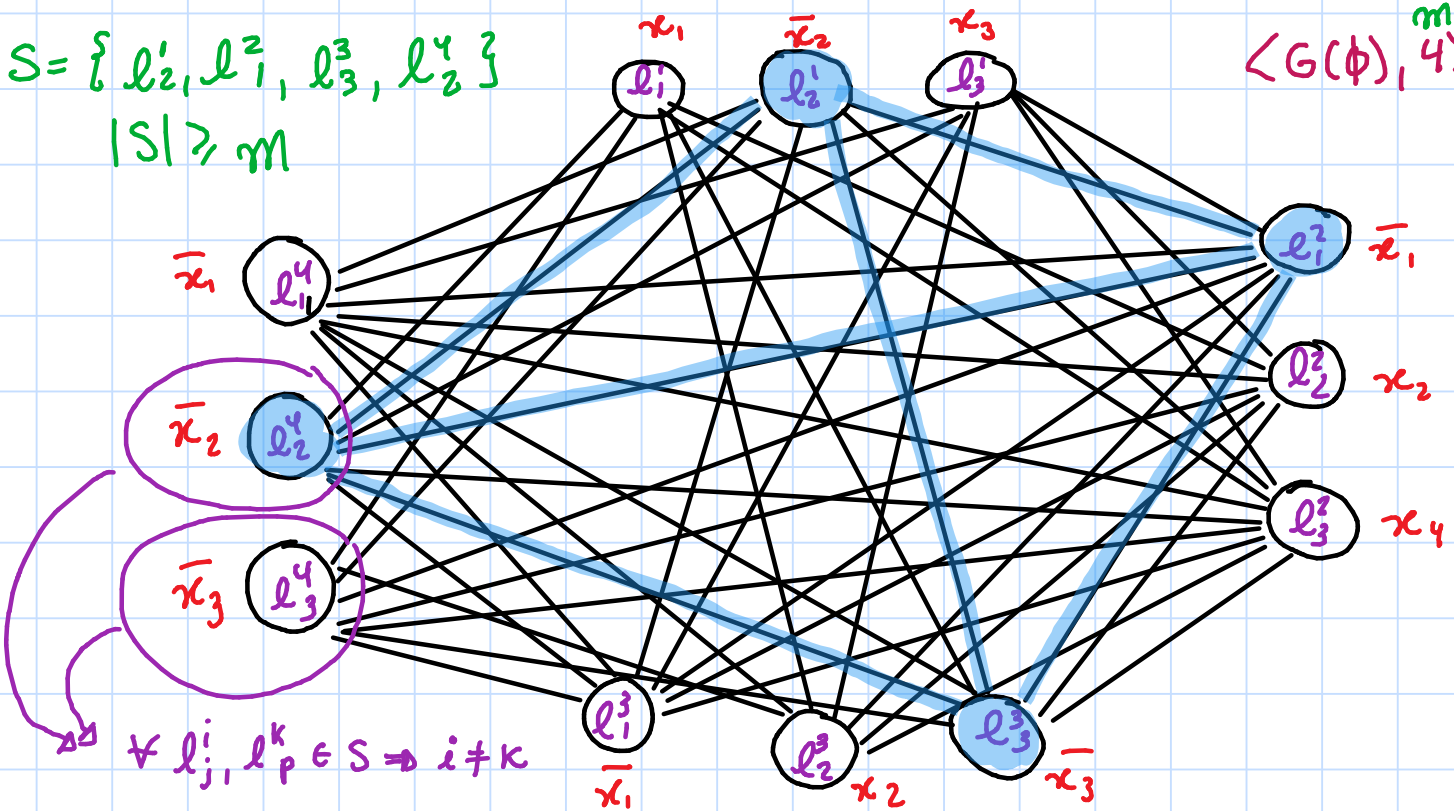
$\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$

$\ell_1^1\ \ell_2^1\ \ell_3^1 \quad \ell_1^2\ \ell_2^2\ \ell_3^2 \quad \ell_1^3\ \ell_2^3\ \ell_3^3 \quad \ell_1^4\ \ell_2^4\ \ell_3^4$

$S = \{\ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4\}$

$|S| \geq m$

$\langle G(\phi), m \rangle$



$\forall\ \ell_j^i, \ell_p^k \in S \Rightarrow i \neq k$

C_1 C_2 C_3 C_4

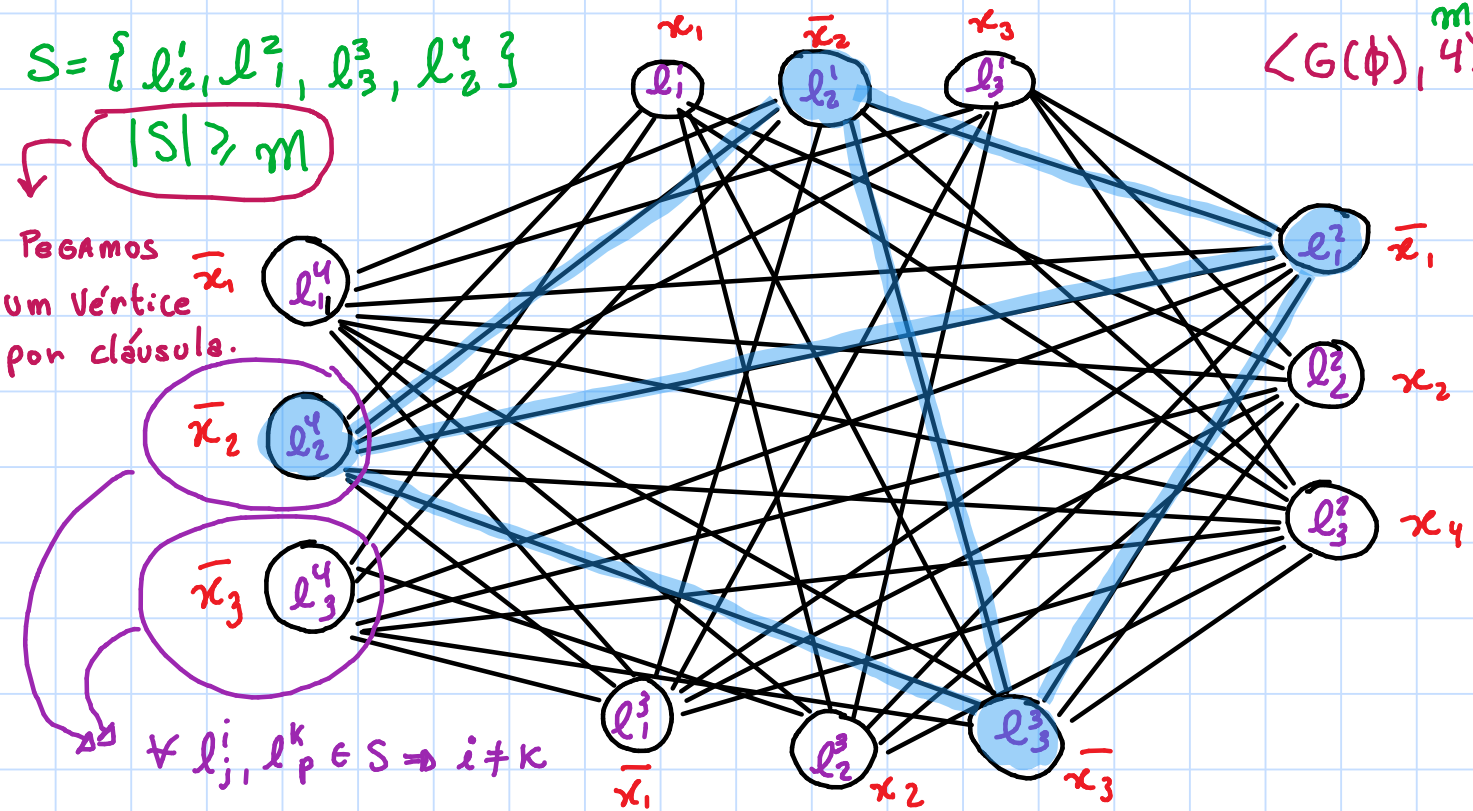
$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

$\langle G(\phi), m \rangle$

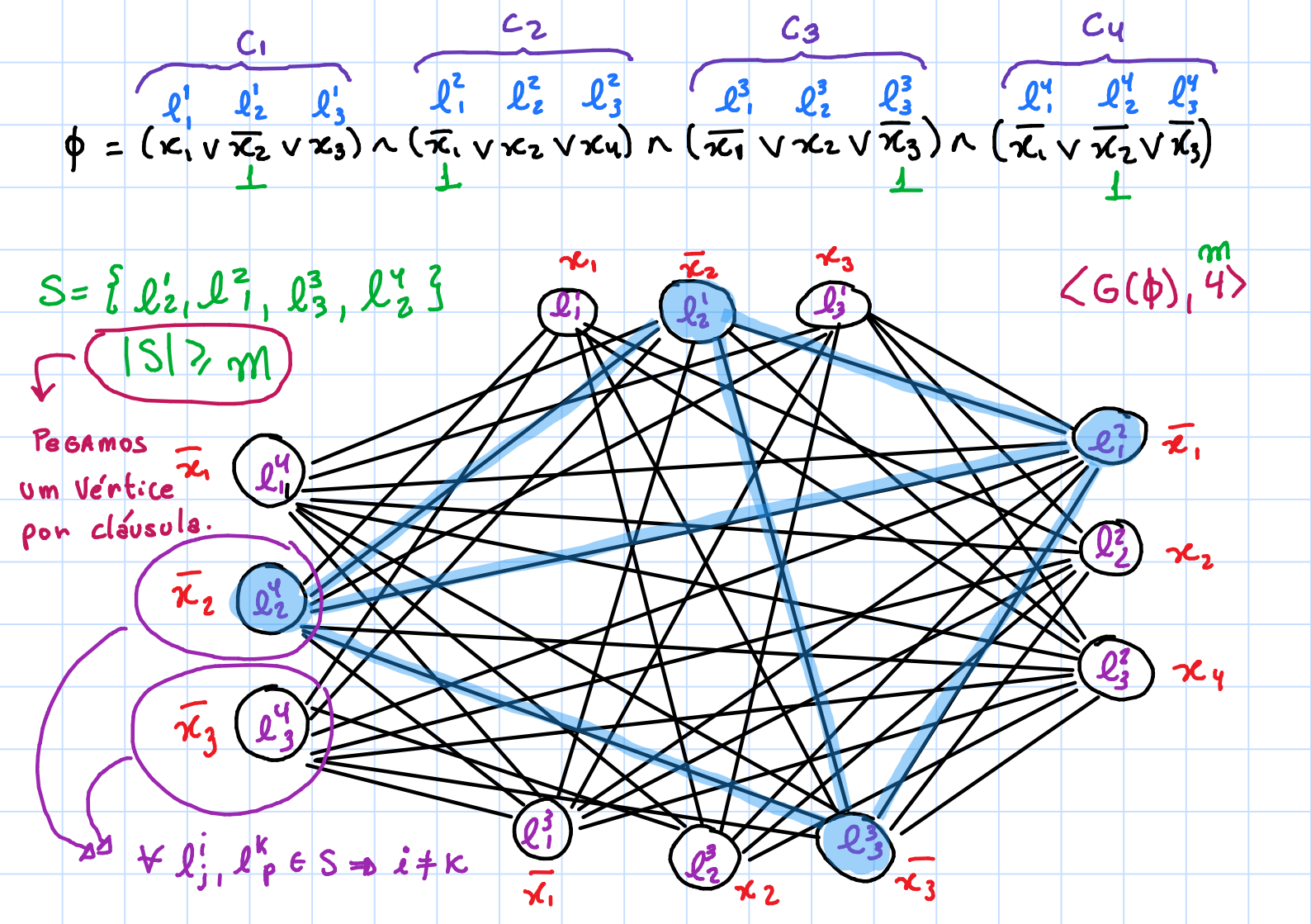
$S = \{ \ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4 \}$

$|S| \geq m$

Pegamos um vértice por cláusula.



$\forall \ell_j^i, \ell_p^k \in S \Rightarrow i \neq k$


C1
C2
C3
C4
$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$
1
1
1
1
$S = \{ \ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4 \}$
$|S| \geq m$
$\langle G(\phi), m \rangle$
4
Pegamos um vértice por cláusula.
$\forall \ell_j^i, \ell_p^k \in S \Rightarrow i \neq k$
x_1
$\overline{x_2}$
x_3
$\overline{x_1}$
x_2
x_4
$\overline{x_3}$
$\overline{x_1}$
x_2
$\overline{x_3}$

C_1 C_2 C_3 C_4

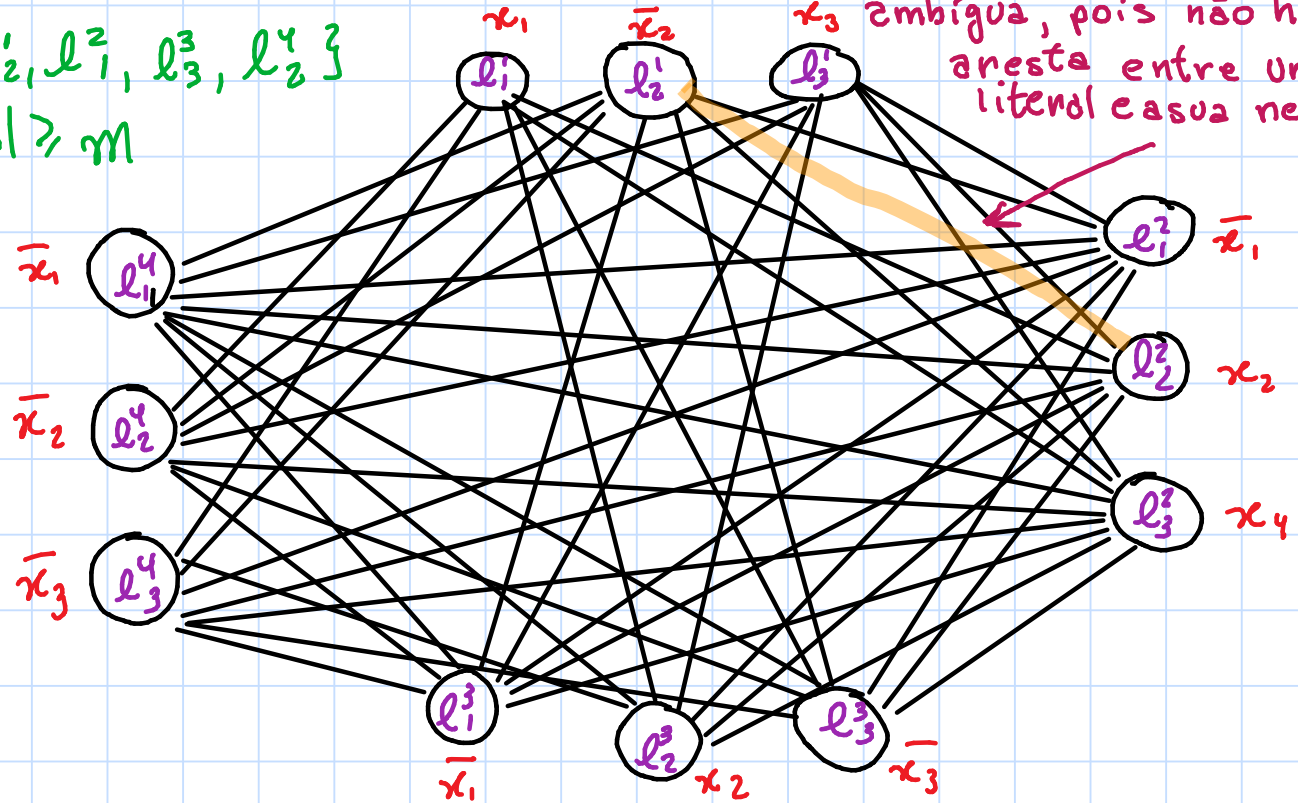
$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

$\ell_1^1\ \ell_2^1\ \ell_3^1 \quad \ell_1^2\ \ell_2^2\ \ell_3^2 \quad \ell_1^3\ \ell_2^3\ \ell_3^3 \quad \ell_1^4\ \ell_2^4\ \ell_3^4$

← não há risco de fazer uma atribuição ambígua, pois não há aresta entre um literal e a sua neg.

$S = \{\ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4\}$

$|S| \geq m$



$$\phi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{C_1} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee x_4)}_{C_2} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}_{C_3} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})}_{C_4}$$

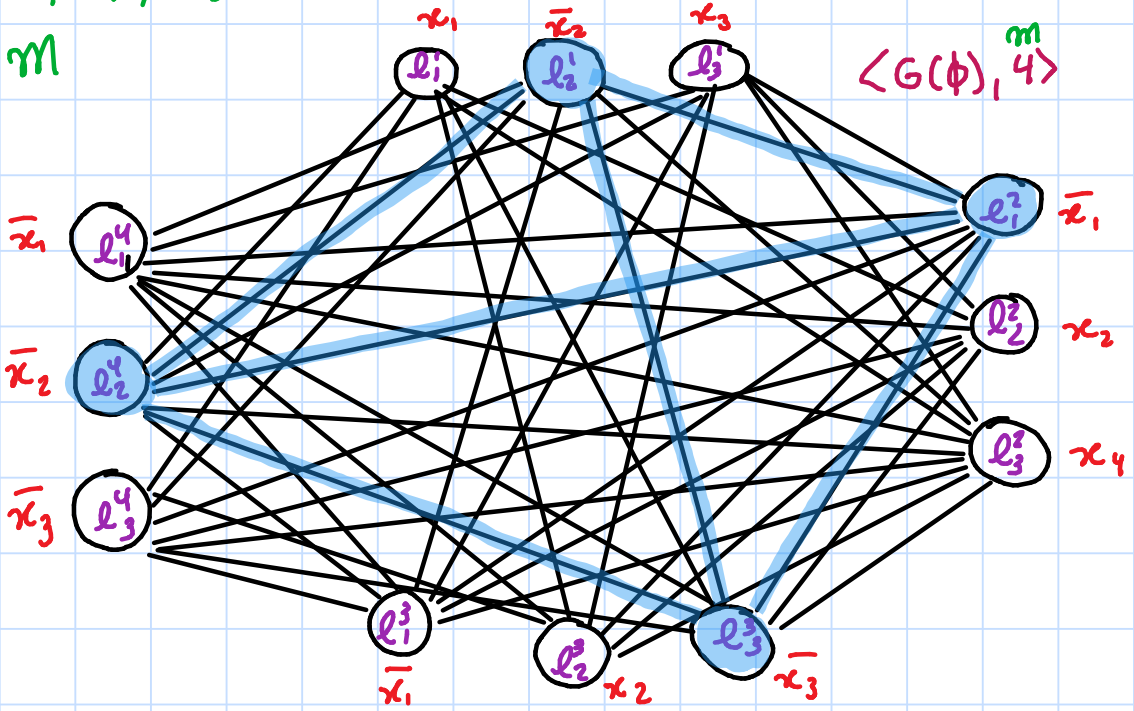
$\ell_1^1 \quad \ell_2^1 \quad \ell_3^1 \qquad \ell_1^2 \quad \ell_2^2 \quad \ell_3^2 \qquad \ell_1^3 \quad \ell_2^3 \quad \ell_3^3 \qquad \ell_1^4 \quad \ell_2^4 \quad \ell_3^4$

1 1 1 1

$S = \{ \ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4 \}$

$|S| \geq m$

x	$\psi(x)$
x_1	0
x_2	0
x_3	0
x_4	0 ou 1



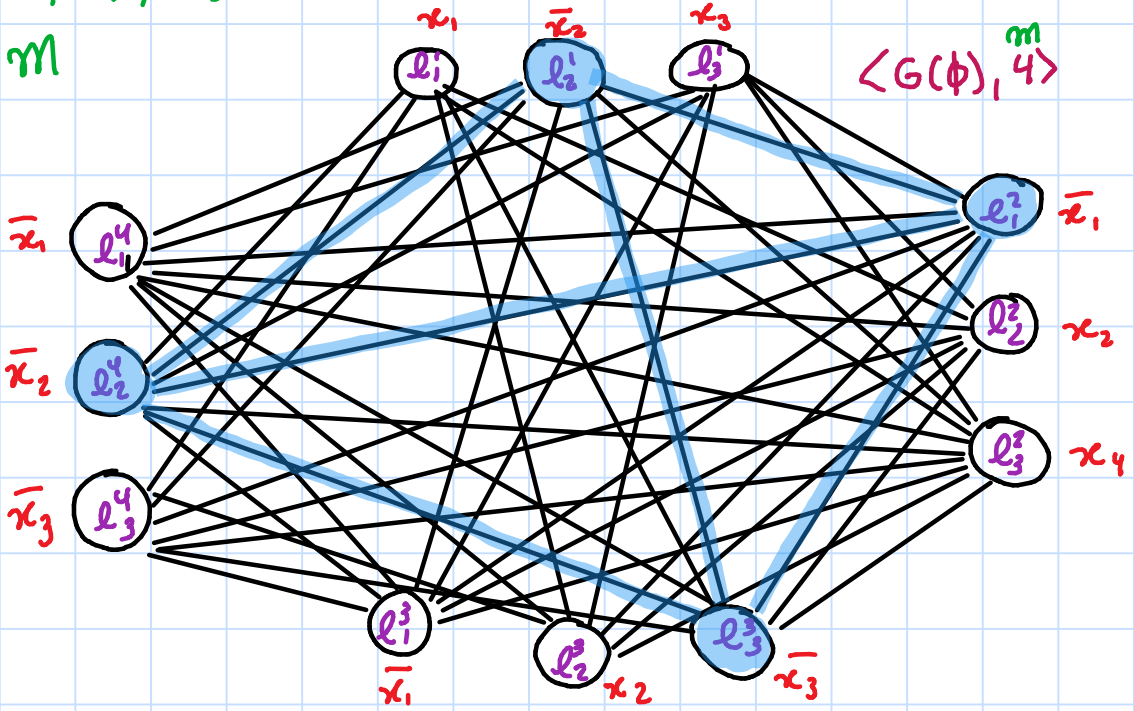
$$\phi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{C_1} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee x_4)}_{C_2} \wedge \underbrace{(\overline{x_1} \vee x_2 \vee \overline{x_3})}_{C_3} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})}_{C_4} = 1$$

$\ell_1^1\ \ell_2^1\ \ell_3^1$: $0\ 1\ 0$ — $\ell_1^2\ \ell_2^2\ \ell_3^2$: $1\ 0\ 1$ — $\ell_1^3\ \ell_2^3\ \ell_3^3$: $1\ 0\ 1$ — $\ell_1^4\ \ell_2^4\ \ell_3^4$: $1\ 1\ 1$

$S = \{ \ell_2^1, \ell_1^2, \ell_3^3, \ell_2^4 \}$

$|S| \geq m$

x	$\psi(x)$
x_1	0
x_2	0
x_3	0
x_4	1



Teo Clique é NP-completo

Demonstração (continuação)

- Seja $S = \{w_1, w_2, \ldots, w_k\}$ uma clique em $G(\phi)$ tal que $k \geq m$.
- Como não há arestas entre vértices que representam literais da mesma cláusula, sabemos que cada $w_i \in S$ pertence a uma cláusula distinta. Como temos m cláusulas e $k \geq m$, concluímos que $k = m$.
- Portanto, podemos supor, sem perda de generalidade, que $w_i \in C_i$, para todo $i = 1, \ldots, m$
  - Seja $\psi : V \to \{0,1\}$ definida da seguinte forma

$$\psi(x) = \begin{cases} 1 & \text{se } \exists\, w_i \in S \text{ tal que } w_i = x \\ 0 & \text{se } \exists\, w_i \in S \text{ tal que } w_i = \bar{x} \\ \perp & \text{caso contrário} \end{cases}$$

Teo Clique é NP-completo

Demonstração (continuação)

- Como não há arestas entre literais l_j^i e l_k^p de cláusulas distintas ($i \neq p$) tais que $l_j^i = x$ e $l_k^p = \bar{x}$ e como S é uma clique, temos que ψ é uma atribuição válida e não ambígua.

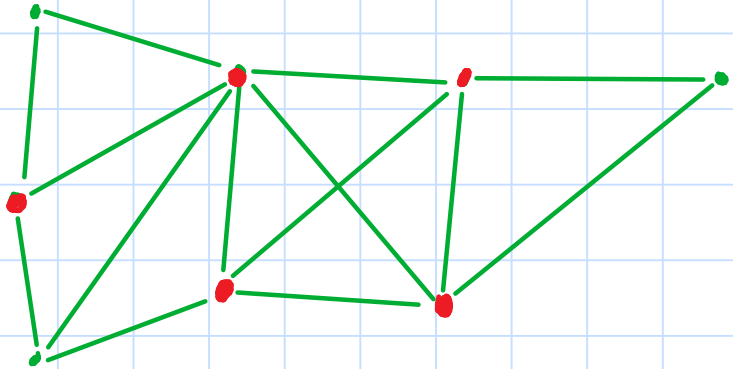
- Note que a atribuição ψ faz com que o literal w_i avalie como verdadeiro na cláusula C_i. Assim, ψ é uma fórmula satisfazível e, portanto, $\langle \phi \rangle \in$ 3-SAT.

D

Problema VERTEXCOVER

Entrada: $\langle G, k \rangle$, onde G é um grafo e k é um inteiro

Saída: sim, se existe um conjunto $S \subseteq V(G)$ tal que $|S| \leq k$ e $\forall\, uv \in E(G)$, vale que $S \cap \{u, v\} \neq \emptyset$; não, caso contrário.



$k = 5$

Teo VERTEXCOVER $\in$ NP-completo

Demonstração

- Primeiro mostraremos que vertexcover está em NP e depois faremos uma redução polinomial de Clique para VertexCover

- VertexCover está em NP pois podemos usar a cobertura de vértices S como certificado. Dado S podemos verificar em tempo polinomial se cada uma das arestas de G foi coberta por S usando o seguinte algoritmo Verificador.

Teo VERTEX COVER $\in$ NP-completo

Demonstração (continuação)

```
Verifica-VertexCover (<G, k>, <S>) {
1   Se <G,k> não é uma codificação válida para
    um grafo G e um inteiro k, retorna 0
2   Se <S> não é uma codificação válida para
    S ⊆ V(G), retorna 0
3   sum ← 0
4   Para cada u ∈ S
5       sum ← sum + 1
6   se sum > k, retorna 0
7   Para toda uv ∈ E(G)
8       se S ∩ {v,u} = ∅
9           Retorna 0
10  Retorna 1
}
```

Teo VERTEX COVER $\in$ NP-completo

Demonstração (continuação)

- É fácil perceber que verifica-VertexCover executa em tempo polinomial
- Portanto VertexCover $\in$ NP
- Agora resta mostrar que Clique $\leq_p$ VertexCover para mostrar que VertexCover $\in$ NP-difícil.

Essa redução é semelhante à redução da versão de otimização do VertexCover para a versão de otimização do clique e por isso é deixada de exercício.

Alerta: Já fizemos na direção oposta, mas a mesma redução funciona para essa direção também!

□

Problema CicloHamiltoniano

Entrada: $\langle D \rangle$, onde D é um digrafo

Saída: sim, se existe um ciclo hamiltoniano em D
não, caso contrário

Teo CicloHamiltoniano $\in$ NP-completo

Ideia da Prova

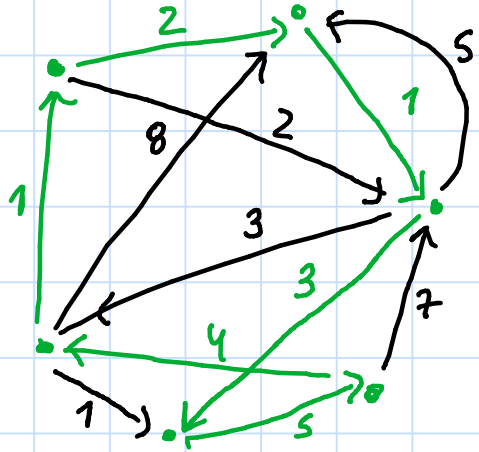
- Primeiro prova que cicloHamiltoniano $\in NP$ e depois mostra que VertexCover $\leq_p$ cicloHamiltoniano

Problema TSP (Popularmente conhecido como o problema do caixeiro viajante)

Entrada: $\langle D, w, k \rangle$, onde D é um digrafo, $w: E(D) \to \mathbb{R}$ e $k \in \mathbb{R}$

Saída: sim, se existe um ciclo Hamiltoniano C em D tal que $w(D) \leq k$.

não, caso contrário.



$K = 22$

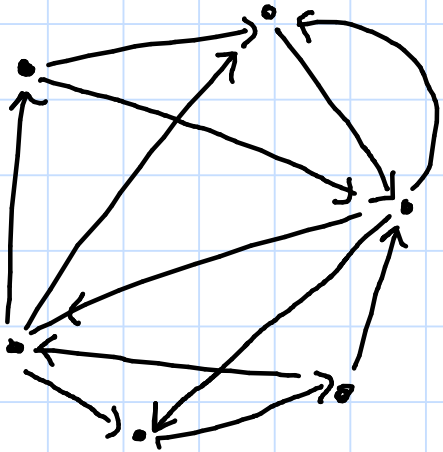
$w(C) = 1 + 2 + 1 + 3 + 5 + 4$

$= 16$

Lema B CicloHamiltoniano $\leq_p$ TSP

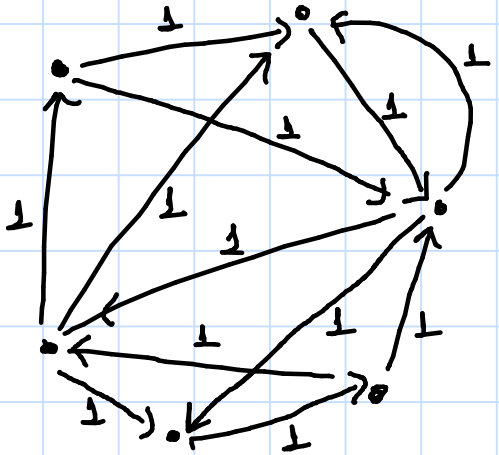
Demonstração

CicloHamiltoniano



$\langle D \rangle$

TSP



$f(D) = \langle D, w, |V(D)| \rangle$

Lema B $\quad$ CicloHamiltoniano $\leq_p$ TSP

Demonstração

Seja f o procedimento definido como:

$f(\langle D \rangle)\{$

Seja $w : E(D) \to \mathbb{R}$ definida como

$w(e) = 1 \quad \forall \; e \in E(D)$

Seja $K = |V(D)|$

Retorne $\langle D, w, k \rangle$

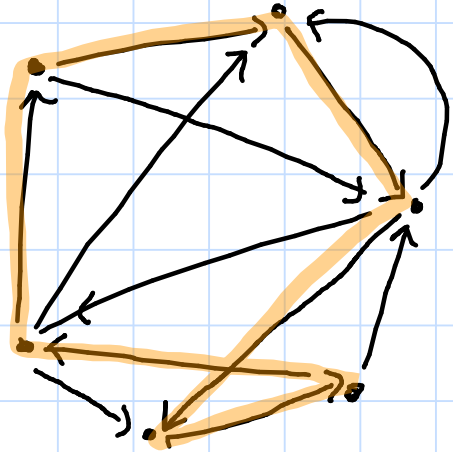
$\}$

É fácil perceber que f toma tempo polinomial. Agora, vamos mostra que f é uma redução.

Lema B CicloHamiltoniano $\leq_p$ TSP

Demonstração

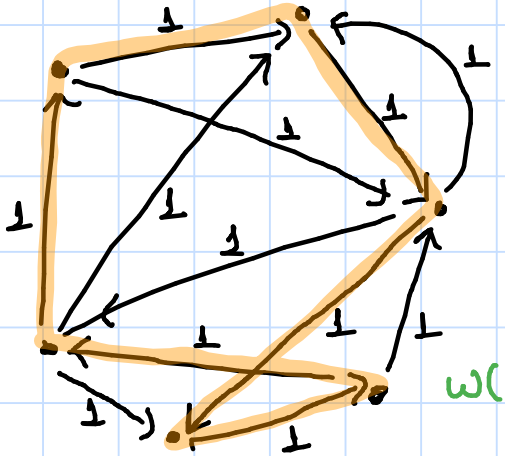
CicloHamiltoniano



C

$\Rightarrow$

TSP



$\omega(C) = 6$

$\langle D \rangle$

$f(D) = \langle D, w, 6 \rangle$

$|V(D)|$

Lema B CicloHamiltoniano $\leq_p$ TSP

Demonstração (continuação)

- Suponha que $\langle D \rangle \in$ ciclohamiltoniano e seja $C \subseteq D$ um ciclo hamiltoniano de D.
- Seja $\langle D', w, k \rangle = f(\langle D \rangle)$.
- Note que $C \subseteq D' = D$ é um ciclo Hamiltoniano tal que

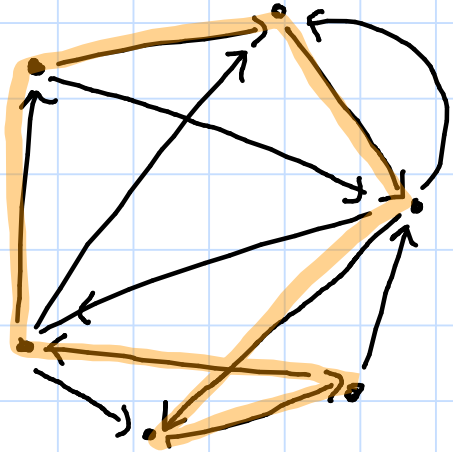
$$w(C) = \sum_{e \in E(C)} w(e) = \sum_{e \in E(C)} 1 = |E(C)| = |V(D')| = k.$$

- Portanto, $\langle D', w, k \rangle \in$ TSP.

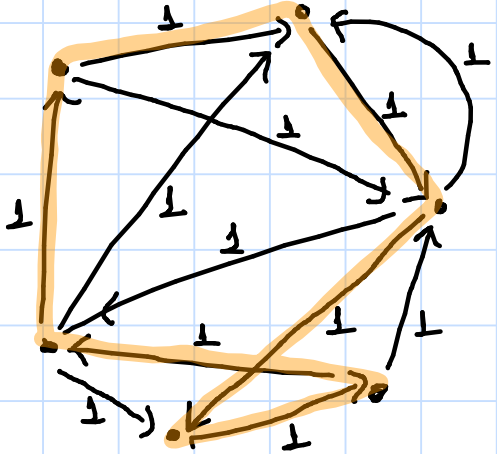
Lema B CicloHamiltoniano $\leq_p$ TSP

Demonstração

CicloHamiltoniano



TSP



C $\Longleftarrow$

$\langle D \rangle$

$f(D) = \langle D, w, 6 \rangle$

$|V(D)|$

Lema B CicloHamiltoniano $\leq_p$ TSP

Demonstração (continuação)

- Seja $f(\langle D \rangle) = \langle D', w, k \rangle \in TSP$ e seja $C \subseteq D'$ um ciclo hamiltoniano tal que $w(C) \leq k$.
- Note que $C \subseteq D' = D$ e $V(C) = V(D') = V(D)$.

.Portanto D é cicloHamiltoniano. □

Teo TSP é NP-difícil

Demonstração

Consequência direta do Lema B □

Teo TSP é NP-completo

Demonstração

Consequência direta dos Lemas A e B □

Lema A TSP ∈ NP