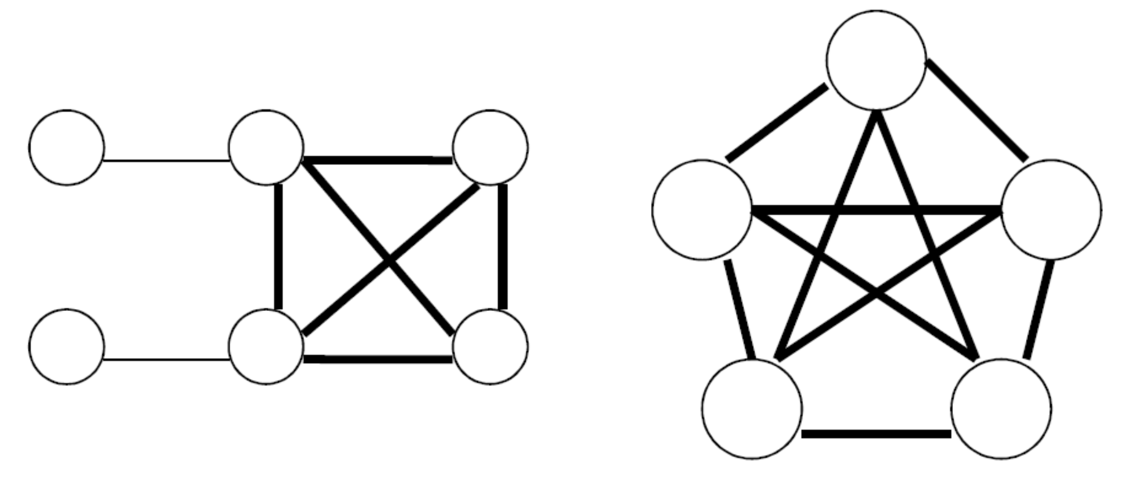
CLIQUE

Gustavo Jandt Feller – UFRGS – Ciência da computação
Neimar Bitencourt Braga – UFRGS – Ciência da computação

Problema do Clique

. O problema do clique consiste em dado um G(V,E), grafo não direcionado, sem pesos nas arestas e um numero natural k <= |V|, existe em G um subgrafo completo com K vértices?



A figura mostra um grafo de 6 vértices com um 4-clique e um grafo de 5 vértices, o qual é um 5-clique. O problema CLIQUE pode ser definido como um problema de otimização, encontrar o maior clique em determinado grafo, o de listar os cliques de um grafo e, na sua versão de decisão, o de decidir se um grafo contém ou não um clique de um determinado tamanho.

Como Provar que um Problema é NP-Completo

1 – Provar que o problema é NP fazendo sua verificação em tempo polinomial.

2 – Provar que é possível reduzir a ele em tempo polinomial qualquer outro problema NP.

Provando que o clique é NP

.Um verificador para CLIQUE pode ser definido da seguinte forma: o algoritmo recebe um conjunto de vértices V, com k elementos, para fazer o teste (usando G e k já definidos). O primeiro passo deve testar se os vértices contidos em V também são vértices de G. O segundo passo deve testar se G contém arestas ligando todos esses vértices entre si. O algoritmo retorna verdadeiro se ambos os passos retornarem verdadeiro. Podemos descrever esse algoritmo com o seguinte pseudocódigo:

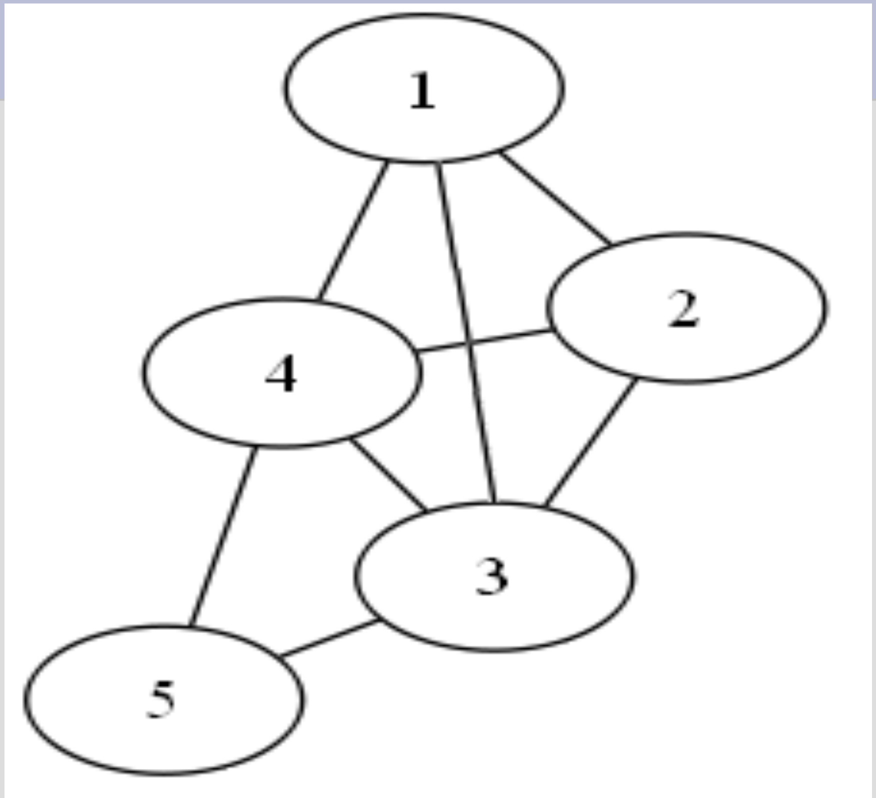
VerificadorClique (grafo G, certificado V, inteiro K):

```
if(k <= 0) return false; //k é um número inválido.
•//Seja q a quantidade de vértices de G
int aux = 0;
for(int i = 0; i <q; i++){
        if(V[i] pertence a G){
                aux++;
        }
}//fim do for
//Se nem todos os elementos de V pertencerem a G
if(aux != k) return false;
```

```
/*Percorrer a lista e verifica se existem arestas para todos os vértices em V. Caso não haja aresta para um deles, return false.*/
for(int a = 0; a < k; a++){
    /*verifica se V[a] esta conectado a todos os vértices de V (excluindo ele mesmo)*/
    answer=ConectadoATodos(G,V[a],V);
    if(answer==false)
        return false;
}
/*Se o algoritmo chegar nesse ponto é porque existe um k-clique em G com os vértices contidos em V*/
return true;
```

. Exemplo de Verificação:

Grafo G como abaixo:



K = 4 e Certificado = {1,2,3,4}

O algoritmo resulta em true, porque os vértices dados no certificado formam um grafo completo.

Analisando a Complexidade do Algoritmo de Verificação

. Analisando a complexidade desse algoritmo notamos que: os laços for do primeiro passo fornecem complexidade O (k*q), já que dentro deles só há instruções com complexidade O(1). As outras instruções do primeiro passo também são todas O(1). No segundo passo o laço for tem complexidade O(V*V). Então a complexidade total desse algoritmo é $O(n^2)$, que é visivelmente polinomial, provando portanto que CLIQUE é um problema NP.

Provando que o CLIQUE pertence a NP-DIFÍCIL

. Para provar que um problema p1 é NP-DIFÍCIL, precisamos reduzir um problema p2, de maneira que p2 seja um problema NP-DIFÍCIL. Para realizar essa redução de p2 em p1, precisamos de um algoritmo de redução em tempo polinomial, que dada uma instância de p2, a transformamos em uma instância de p1. Se p1 já foi provado ser NP, e conseguirmos provar que ele também é NP-DIFÍCIL, então p1 será NP-COMPLETO. Provaremos que o Clique é NP-DIFÍCIL reduzindo o problema da Satisfabilidade (SAT) a ele.

```
ReducaoSATClique (equacao S)
    // G(V,E) é um grafo com V vértices e E arestas
  // S = clausula1 □ clausula2 □ ... □ clausula i;
  //clausula i= var X \/ var Y ...


    // Parte 1
  For(i=0; i<S.size; i++) //para toda clausula de S
     For(j=0; j<C[i].size; j++) //para toda variável da clausula
        {
         V = V + S[i].clausula[j]; //é adicionado a V
     }
```

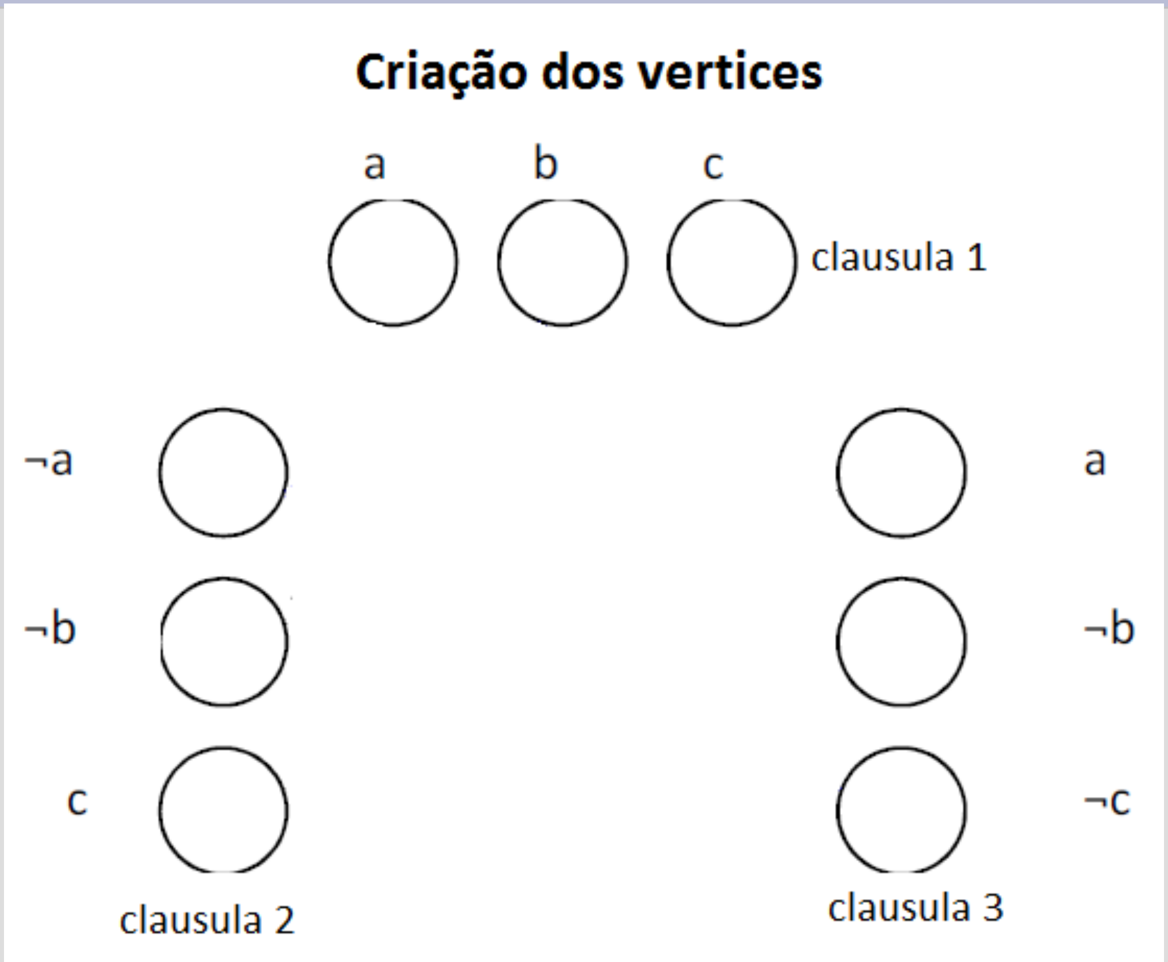
```
    //parte 2
For(i=0; i<V.size; i++)
        For(j=0; j<V.size; j++)
        {
            /*se as clausulas forem diferentes e a variável avaliada
          nao for a sua negação ex: X e ¬X, a aresta é adicionada.
          */
         if(V[i].clausula □ V[j].clausula and V[i].var   □ □V[j].var){
              E = E + (V[i], V [j]);//adiciona a aresta
            }
        }
return G(V,E)
```

Analisando a Complexidade do Algoritmo de Redução

. A parte 1 que consiste na inclusão de todos literais em V, tem complexidade $O(s*c)$ (onde s é a quantidade de cláusulas e c a maior quantidade de ocorrência de literais), considerando n a quantidade total de ocorrência de literais, então a primeira parte tem complexidade $O(n)$. E a parte 2, que consiste na inclusão das arestas, tem complexidade $O(v^2)$, onde v é a quantidade de vértices formado no grafo, ou seja, é a ocorrência de literais. Com isso, temos complexidade $O(n^2)$.

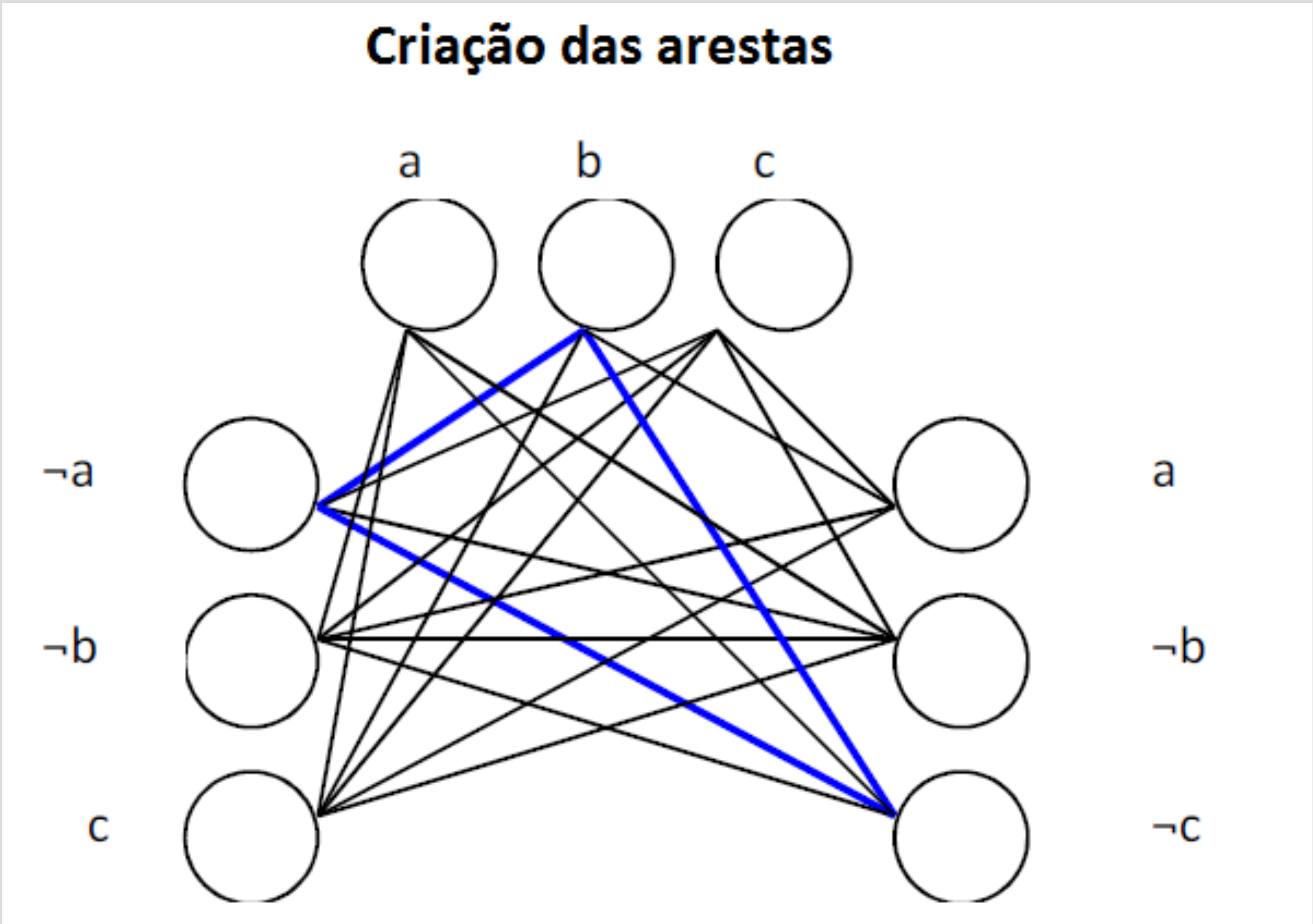
Exemplo de redução:

S=(a ∨ b ∨ c) ^ (¬a ∨ ¬b ∨ c) ^ (a ∨ ¬b ∨ ¬c)



Exemplo de redução:

$S=(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee \neg c)$



Conclusão

. Mostramos que o problema Clique possui um algoritmo de verificação em tempo polinomial, ou seja, provamos que ele pertence a NP, e mostramos um algoritmo de redução de SAT para Clique, também em tempo polinomial, ou seja, provamos que ele também pertence a NP-DIFÍCIL, logo, pertencendo a esses dois conjuntos, provamos que o problema do Clique é NP-COMPLETO.