UNIVERSIDADE FEDERAL DE OURO PRETO
DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO
BCC241 - PROJETO E ANÁLISE DE ALGORITMOSS

Relatório do Trabalho Prático

IMPLEMENTAÇÃO EM C++ DE TRÊS PROBLEMAS CLÁSSICOS

Alunos:

FELIPE PÉRET MORAES SASDELLI,

LUKA NASCIMENTO MENIN

Resumo

Este projeto é um trabalho prático para a disciplina de Projeto e Análise de Algoritmos da Universidade Federal de Ouro Preto e implementa três problemas clássicos em C++:

1. Satisfatibilidade (SAT), utilizando backtracking;
2. Clique Máximo, por meio de branch and bound;
3. Conjunto Independente Máximo, obtido por redução a partir do Clique em tempo polinomial.

Incluimos, para fins de teste e avaliação (*benchmarking*) dos algoritmos implementados, algumas instâncias de entrada exemplo para cada problema, assim como geradores de instâncias.

Os códigos das implementações aqui discutidas podem ser encontrados em repositório público no Github: https://github.com/Hotto-Doggu/sat$_c$*lique*$_s$*et*

1 Introdução aos Problemas

1.1 Problema de Satisfabilidade (SAT)

O problema de Satisfabilidade Booleana (SAT) consiste em verificar se, para uma fórmula booleana dada, expressa na forma normal conjuntiva (CNF), existe uma atribuição de valores-verdade às variáveis da fórmula que a torne verdadeira. Este problema é o primeiro a ser provado como **NP-completo** (Cook, 1971), e sua importância se deve ao fato de que muitos problemas de decisão em ciência da computação podem ser reduzidos a uma instância de SAT.

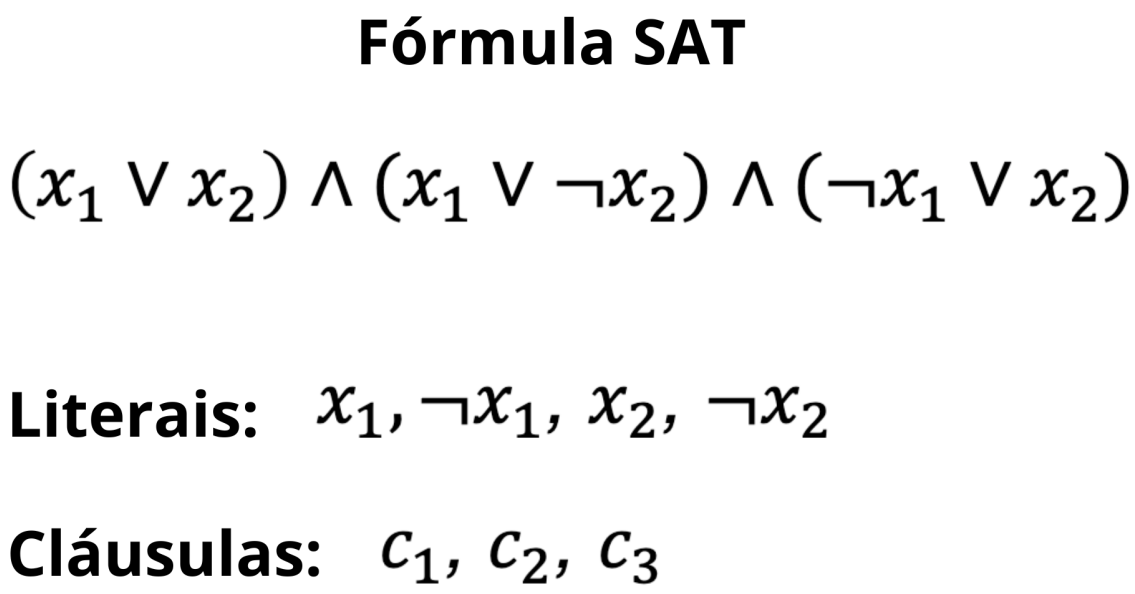

Figura 1: Exemplo de fórmula SAT booleana

1.2 Problema do Clique Máximo

No problema do Clique Máximo, dado um grafo $G(V,E)$ o objetivo é encontrar o maior subgrafo completo, ou seja, um conjunto máximo de vértices tal que todas as arestas possíveis entre eles estejam presentes.

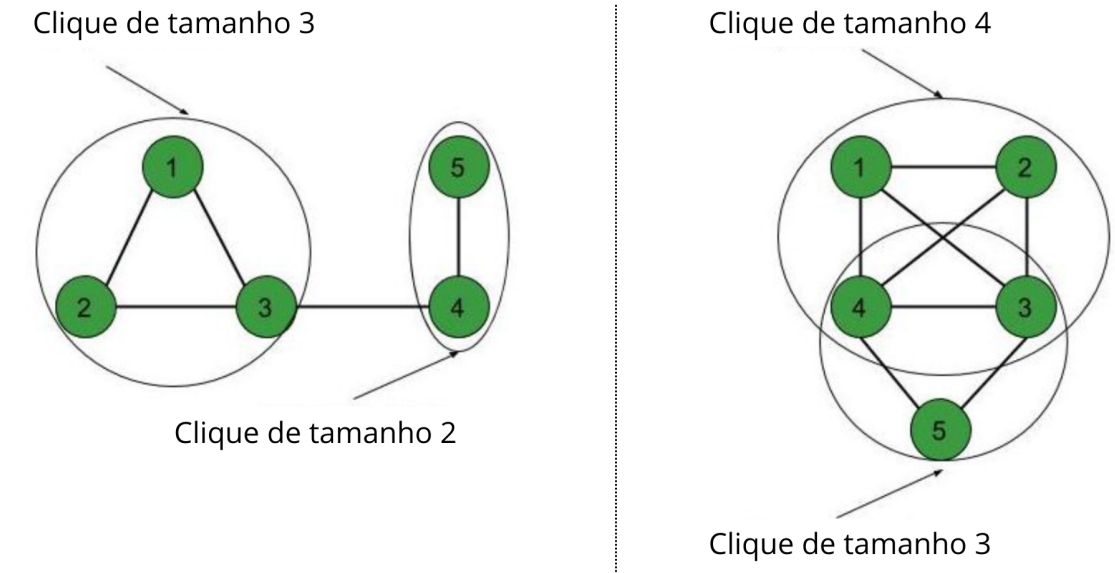

Figura 2: Exemplo de Clique Máximo em dois gráfos

1.3 Problema do Conjunto Independente

O problema do Conjunto Independente envolve encontrar o maior conjunto de vértices em um grafo tal que nenhum par de vértices no conjunto seja adjacente. Assim como o problema do Clique Máximo, o Conjunto Independente é NP-completo, e a solução de ambos está inter-relacionada: um conjunto independente em um grafo G corresponde a um clique no grafo complementar G'.

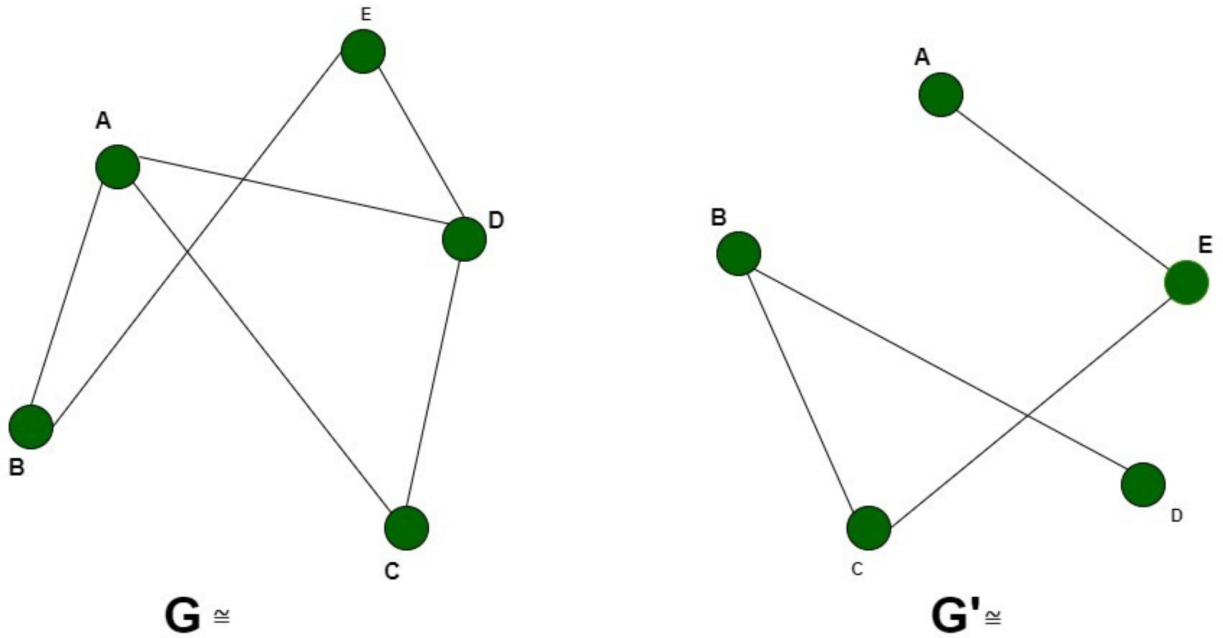

Figura 3: Exemplo de Clique e Conjunto Independente correspondentes, nos gráfos G (original) e G' (complemento)

2 Classificação de Complexidade

Os três problemas discutidos – SAT, Clique Máximo e Conjunto Independente – pertencem à classe dos problemas **NP-completos**. Isso significa que eles satisfazem duas condições:

1. **Pertencem à classe NP:** Problemas em **NP** podem ser verificados em tempo polinomial se uma solução (ou "certificado") for dada. Por exemplo, no problema de clique, dado um subgrafo S, podemos verificar se é um clique máximo em tempo polinomial.

2. **São NP-difíceis:** Qualquer problema na classe **NP** pode ser reduzido em tempo polinomial a um problema **NP-difícil**. No caso do problema de clique, ele é **NP-difícil** porque qualquer problema **NP** pode ser transformado em uma instância do problema de clique, e isso pode ser feito em tempo polinomial.

2.1 NP-Completude do SAT

O problema de Satisfabilidade Booleana (SAT) foi o primeiro problema a ser provado como **NP-completo** através do Teorema de Cook em 1971. A prova de Cook demonstrou que qualquer problema em **NP** pode ser transformado em uma instância do problema de SAT em tempo polinomial.

2.1.1 Estrutura Geral da Prova de Cook:

A ideia central é representar a execução de uma máquina de Turing não-determinística, que resolve um problema em **NP**, como uma fórmula booleana. A fórmula booleana descreve as transições entre estados dessa máquina de Turing e, se ela for satisfeita, implica que a máquina aceita a entrada. Se a fórmula booleana é satisfatível, isso significa que existe uma atribuição de valores-verdade que corresponde a uma sequência válida de transições da máquina, provando que a máquina aceita a entrada em questão. Como qualquer problema em **NP** pode ser descrito em termos de uma máquina de Turing não-determinística, isso mostra que SAT é **NP-completo**.

Além disso, o SAT é **NP-difícil**, pois problemas como o Clique e Conjunto Independente podem ser transformados em uma instância de SAT.

2.1.2 Redução do Clique a SAT:

O problema de Clique consiste em encontrar, em um grafo $G(V,E)$, um subconjunto de vértices $C \subseteq V$ tal que todos os vértices em C estejam conectados entre si, formando um subgrafo completo. Para reduzir esse problema para SAT, podemos formular a seguinte abordagem:

Variáveis Booleanas: Para cada vértice $v_i \in V$ e cada posição j no clique de tamanho k, definimos uma variável booleana $x_{i,j}$, onde$x_{i,j} = 1$ indica que o vértice v_i ocupa a j-ésima posição no clique.

Restrições: Cada posição no clique de tamanho k deve ser ocupada por — exatamente — um vértice, e vértices conectados devem formar arestas. Além disso, apenas vértices que compartilham arestas possam fazer parte do clique.

1. Cada posição no clique j deve ser ocupada por algum vértice:

$$\bigvee_{i=1}^{n} x_{i,j}$$

2. Para garantir que não há dois vértices ocupando a mesma posição j:

$$\bigwedge_{i \neq i'} (\neg x_{i,j} \vee \neg x_{i',j})$$

3. Para cada par de vértices (i, i'), se não houver uma aresta $(v_i, v_{i'})$ no grafo, então eles não podem estar juntos no clique:

$$\bigwedge_{i \neq i'} (\neg x_{i,j} \vee \neg x_{i',j'})$$

   se $(v_i, v_{i'}) \notin E$

Fórmula Resultante: A fórmula CNF resultante é a conjunção das restrições para garantir que cada posição no clique seja ocupada por exatamente um vértice com as restrições para garantir que os vértices no clique estejam conectados.

Se essa fórmula booleana for satisfatível, isso implica que existe um clique de tamanho k no grafo G. A solução da fórmula pode ser interpretada como os vértices que fazem parte do clique.

2.2 NP-Completude do Clique

Pertence a NP: Para verificar se um subgrafo S de um grafo G é um clique de tamanho k, basta verificar em tempo $O(k^2)$ se todas as arestas entre os vértices de S estão presentes no grafo G. Isso pode ser feito de maneira eficiente, já que envolve apenas checar as arestas entre pares de vértices.

É NP-difícil: Para provar que o problema do Clique Máximo é **NP-difícil**, basta mostrar que um problema **NP-completo** já conhecido, como o SAT, pode ser reduzido em tempo polinomial ao problema de clique.

2.2.1 Redução do SAT (3-CNF) a Clique:

A estratégia de redução do 3-SAT (versão em que cada cláusula tem exatamente três literais) para Clique é a seguinte:

Fórmula SAT 3-CNF: Consideramos uma fórmula booleana Φ com m cláusulas $C_1, C_2, \ldots, C_m$, onde cada cláusula contém exatamente três literais. O objetivo é transformar essa fórmula booleana em um grafo G tal que exista um clique de tamanho m se, e somente se, a fórmula Φ for satisfatível.

Construção do grafo: Para cada cláusula C_j da fórmula Φ, criamos um conjunto de três vértices no grafo G, cada um representando um literal da cláusula C_j. Assim, para uma cláusula $C_j = (x_1 \vee x_2 \vee \neg x_3)$, os vértices correspondentes seriam $v_{1,j}, v_{2,j}, v_{3,j}$, onde $v_{1,j}$ representa x_1, $v_{2,j}$ representa x_2 e$v_{3,j}$ representa $\neg x_3$.

Arestas no grafo: As arestas entre os vértices são adicionadas de tal forma que:

1. Vértices de cláusulas diferentes são conectados se, e somente se, eles não representam literais contraditórios (como x_i e $\neg x_i$).

2. Vértices da mesma cláusula não são conectados, para garantir que apenas um literal de cada cláusula pode ser escolhido como parte do clique.

Clique de tamanho m: Um clique de tamanho m nesse grafo corresponderá à seleção de um literal de cada cláusula de forma que nenhum literal selecionado contradiga os outros. Se esse clique existe, a fórmula booleana Φ é satisfatível, pois para cada cláusula há um literal verdadeiro e todos os literais escolhidos são consistentes entre si.

Validação da Redução: Se a fórmula Φ for satisfatível, existe uma atribuição de valores-verdade para os literais tal que pelo menos um literal em cada cláusula é verdadeiro. O conjunto de vértices correspondentes a esses literais formará um clique de tamanho m, já que nenhum par de vértices (literais) contradirá o outro; Se existir um clique de tamanho m no grafo construído, então, para

cada cláusula C_j, há pelo menos um literal selecionado que é consistente com os literais selecionados em outras cláusulas. Assim, podemos definir uma atribuição de valores-verdade para os literais que satisfaz a fórmula Φ.

2.2.2 Cálculo da Complexidade

A complexidade da construção do grafo G a partir da fórmula 3-CNF pode ser decomposta em duas partes:

1. Para cada cláusula, criamos três vértices, resultando em 3m vértices no total, onde m é o número de cláusulas.

2. As arestas são adicionadas entre vértices de diferentes cláusulas, levando em consideração as contradições entre os literais. A verificação de contradições entre literais pode ser feita em tempo $O(1)$ para cada par de vértices, totalizando $O(m^2)$ arestas. Assim, o tempo total para construir o grafo é $O(m^2)$, o que é polinomial em relação ao número de cláusulas da fórmula.

2.3 NP-Completude do Conjunto Independente

O problema do Conjunto Independente pertence à classe **NP**, pois dado um conjunto de vértices S, é fácil verificar em tempo polinomial se S é um conjunto independente, ou seja, se nenhum par de vértices em S compartilha uma aresta no grafo G. Essa verificação pode ser feita em $O(n+m)$, onde n é o número de vértices e m é o número de arestas do grafo.

O problema também é **NP-difícil**, e isso pode ser provado por uma redução polinomial do problema de Clique Máximo.

2.3.1 Redução em Tempo Polinomial do Conjunto Independente para Clique Máximo

A redução do Conjunto Independente para o Clique Máximo é realizada criando o grafo complementar de G. No grafo complementar G', todas as arestas existentes no grafo original G são removidas, e todas as arestas inexistentes são adicionadas. A construção do grafo complementar pode ser feita em tempo $O(n^2)$, onde n é o número de vértices, pois envolve inverter as arestas de G, o que pode ser feito com uma operação de $O(1)$ para cada par de vértices.

Após construir o grafo complementar, o algoritmo de branch and bound pode ser aplicado para resolver o problema do Clique Máximo. Embora esse algoritmo seja exponencial no pior caso, a redução de Conjunto Independente para Clique Máximo ocorre em tempo polinomial, devido à eficiência na construção do grafo complementar.

2.3.2 Cálculo da Complexidade

A complexidade da redução pode ser decomposta em duas partes:

1. Construção do grafo complementar: Para um grafo $G = (V, E)$ com nnn vértices, a construção do grafo complementar G' envolve inverter as arestas, o que pode ser feito em $O(n^2)$.

2. Aplicação do algoritmo de clique máximo: O algoritmo de branch and bound para resolver o Clique Máximo no grafo complementar tem uma complexidade exponencial, como esperado para problemas **NP-completos**. No entanto, a transformação do Conjunto Independente para Clique Máximo é feita em tempo polinomial.

3 Metodologia

3.1 Enunciado do trabalho prático

3.1.1 Satisfabilidade

Dada uma fórmula booleana na forma normal conjuntiva, encontre uma atribuição de valores-verdade às variáveis da fórmula que a torne verdadeira, ou informe que não existe tal atribuição.

O problema da Satisfabilidade deve ser resolvido usando *Backtracking*.

3.1.2 Clique

Dado um grafo, encontre um conjunto máximo de vértices tal que todas as possíveis arestas entre eles estejam presentes. O Clique deve ser resolvido usando *Branch and Bound*.

3.1.3 Conjunto independente

Dado um grafo, o objetivo é encontrar o maior número de vértices independentes, isto é, não existe aresta entre nenhum par deles. E o problema do conjunto independente deve ser resolvido por meio de **redução com custo polinomial** ao problema do Clique, já implementado usando *branch and bound*.

3.2 Descrição da tarefa e entregáveis

Entregar uma documentação contendo as descrições e explicações dos 3 algoritmos, juntamente com as decisões tomadas em suas implementações e os resultados de 3 instâncias de cada problema. Entregar os códigos fontes, as instâncias utilizadas e a documentação, via Moodle. Cada grupo deverá apresentar o trabalho, com 15 min para a apresentação.

4 Implementação

4.1 Especificação dos algoritmos

4.1.1 Problema de Satisfabilidade

Entrada: A entrada será fornecida através de um arquivo de texto no seguinte formato: A primeira linha do arquivo contém um número inteiro n que representa a quantidade de variáveis na fórmula.

As linhas seguintes representam as cláusulas da fórmula booleana. Cada linha terá exatamente n números inteiros que correspondem às variáveis:

- **1** indica que a variável está presente na cláusula e não é negada.
- **0** indica que a variável está presente na cláusula e é negada.
- **-1** indica que a variável não está presente na cláusula.

Saída: A saída consiste dos valores atribuídos a cada variável, caso a fórmula seja satisfeita, ou da não-satisfabilidade da fórmula, e do tempo de execução do algoritmo.

3		
1	1	1
1	0	-1
-1	1	0
0	-1	1
0	0	0

Tabela 1: Exemplo de entrada (fórmula booleana CNF) para o problema SAT.

Atribuições: x1 = True, x2 = False, x3 = True
Tempo de execução: 0.005 segundos

Tabela 2: Exemplo de saída do algoritmo SAT.

4.1.2 Problemas Envolvendo Grafos (Clique e Conjunto Independente)

Entrada: A entrada será fornecida através de um arquivo de texto com o seguinte formato: A primeira linha do arquivo contém um número inteiro *n* que representa a quantidade de vértices do grafo. As *n* linhas seguintes contêm uma matriz de adjacência *nxn*, onde:

- **1** indica a presença de uma aresta entre dois vértices.
- **0** indica a ausência de uma aresta entre dois vértices.

9								
0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	0	0	1	0	1	0	1
0	0	0	0	0	0	1	1	0

Tabela 3: Exemplo de entrada (matriz de adjacências) para os problema Clique e Conjunto Independente.

Saída: A saída consistirá nos vértices que fazem parte da solução do problema (clique máximo ou conjunto independente) e o tempo de execução do algoritmo.

Clique Máximo: [2, 3, 4, 5]
Tempo de execução: 0.013 segundos

Tabela 4: Exemplo de saída do problema Clique.

4.2 Código

Explicação das soluções algorítmicas em pseudocódigo.

4.2.1 Satisfabilidade

O algoritmo usa *backtracking*, tentando atribuir valores booleanos (*True* ou *False*) às variáveis e verificando se as cláusulas são satisfeitas com essas atribuições. A função *isSatisfied* verifica se uma atribuição específica satisfaz todas as cláusulas.

Algorithm 1: Satisfabilidade (SAT)

Input: Número de variáveis e lista de cláusulas
Output: Atribuição de variáveis que satisfaz a fórmula ou "não há solução"

```
1  Função solveSATProblem(cláusulas, num_variáveis):
2      Inicializa uma atribuição vazia com tamanho igual ao número de variáveis;
3      solveSAT(0, num_variáveis);
4      if satisfiable é verdadeiro then
5          Exibe a atribuição que satisfaz a fórmula;
6      else
7          Informa que não há solução;
8  Função solveSAT(índice, num_variáveis):
9      if índice é igual ao número de variáveis then
           // Verifica se a fórmula está satisfeita usando
              isSatisfied(cláusulas)
10         if a fórmula está satisfeita then
11             satisfiable ← verdadeiro;
12         return;
13     else
14         Atribui 0 à variável no índice atual e chama solveSAT(índice + 1,
             num_variáveis);
15         if satisfiable é verdadeiro then
16             return;
17         Atribui 1 à variável no índice atual e chama solveSAT(índice + 1,
             num_variáveis);
18 Função isSatisfied(cláusulas):
19     for cada cláusula em cláusulas do
20         Inicializa clauseSatisfied como falso;
21         for cada literal na cláusula do
22             Determina o valor da variável correspondente ao literal;
23             if o literal é verdadeiro then
24                 Marca clauseSatisfied como verdadeiro;
25                 Interrompe o laço;
26         if clauseSatisfied é falso then
27             return falso;
28     return verdadeiro;
```

4.2.2 Clique

Utilizamos o método *branch and bound* para explorar subconjuntos de vértices que formam cliques. O limite (*bound*) serve para podar a exploração de subconjuntos que não podem levar a uma solução maior do que a já encontrada.

Algorithm 2: Clique Máximo (max_clique)

Input: Grafo *G* representado por sua matriz de adjacência
Output: Conjunto de vértices que formam o maior clique encontrado no grafo

```
1  Função max_clique(grafo, num_vértices):
2      Inicializa best_clique como vazio e max_size como 0;
3      Inicializa uma lista de candidatos contendo todos os vértices;
       // Chama a função branch_and_bound com clique inicial vazio e
          todos os vértices como candidatos
4      branch_and_bound(grafo, clique vazio, candidatos);
       // Retorna o melhor clique encontrado
5      return best_clique;
6  Função branch_and_bound(grafo, clique_atual, candidatos):
       // Se tamanho do clique_atual for maior que o melhor clique até
          agora
7      if tamanho de clique_atual > max_size then
8          Atualiza best_clique com clique_atual e max_size com o novo tamanho;
9      for cada vértice v em candidatos do
10         Inicializa uma nova lista de candidatos conectados ao vértice atual v;
11         for cada vértice w após v em candidatos do
12             if v e w estão conectados no grafo then
13                 Adiciona w à nova lista de candidatos;
14         Adiciona v ao clique_atual;
15         branch_and_bound(grafo, clique_atual, nova lista de candidatos);
16         Remove v de clique_atual;
```

4.2.3 Conjunto independente

A solução para o conjunto independente é obtida transformando o grafo original em seu grafo complemento. Depois disso, o algoritmo do clique máximo é aplicado ao grafo complemento, visto que um conjunto independente no grafo original corresponde a um clique no complemento.

Algorithm 3: Conjunto Independente Máximo (max_independent_set)

Input: Grafo G representado por sua matriz de adjacência
Output: O maior conjunto independente encontrado no grafo

Função `max_independent_set(`*grafo, num_vértices*`)`**:**
    Gera o grafo complementar G' usando `complement_graph(`*grafo, num_vértices*`)`;
    `// Inicializa uma lista de candidatos contendo todos os vértices`
    `branch_and_bound(`*grafo_complementar, clique vazio, candidatos*`)`;
    `// Retorna o melhor clique encontrado no grafo complementar como conjunto independente`
    return *best_clique*

Função `complement_graph(`*grafo, num_vértices*`)`**:**
    Inicializa uma matriz complementar de tamanho $n \times n$;
    `// cada par de vértices` (i, j) `no grafo original` G
    for *cada vértice* v_i **do**
        for *cada vértice* v_j **do**
            `// Se não houver aresta entre` i `e` j
            if $i \neq j$ **then**
                Insere aresta no grafo complementar G';
    return *grafo complementar* G';

Função `branch_and_bound(`*grafo, clique_atual, candidatos*`)`**:**
    if *tamanho de* *clique_atual* > *max_size* **then**
        Atualiza `best_clique` com `clique_atual` e `max_size` com o novo tamanho;
    for *cada vértice v em* *candidatos* **do**
        Inicializa uma nova lista de candidatos conectados a v;
        for *cada vértice w após v em* *candidatos* **do**
            if *v e w estão conectados no grafo* **then**
                Adiciona w à nova lista de candidatos;
        Adiciona v ao `clique_atual`;
        `branch_and_bound(`*grafo, clique_atual, nova lista de candidatos*`)`;
        Remove v de `clique_atual`;

5 Resultados

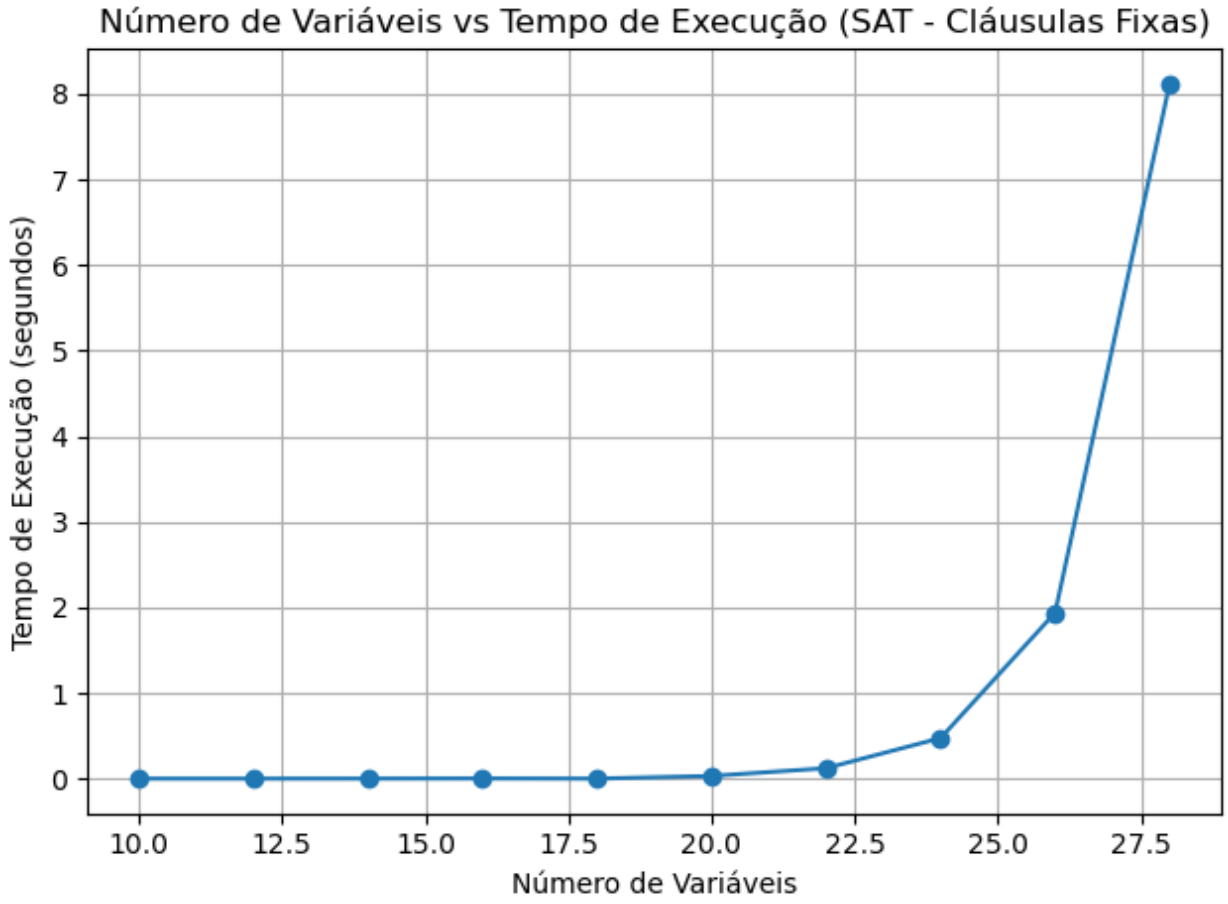

Figura 4: Gráfico de tempos de execução para as dez entradas geradas pro problema SAT

1. **SAT (Backtracking)**: O gráfico para o problema SAT mostra o tempo de execução em função do número de variáveis, mantendo um número fixo de cláusulas. O método de *Backtracking* utilizado para resolver o SAT possui complexidade exponencial, aproximadamente $O(2^n)$, onde n é o número de variáveis. Isso ocorre porque, no pior caso, o algoritmo precisa explorar todas as combinações possíveis de variáveis para encontrar uma solução que satisfaça todas as cláusulas. Podemos ver que o tempo de execução permanece baixo até cerca de 20 variáveis, mas cresce rapidamente depois disso, atingindo cerca de 8 segundos para 28 variáveis. Esse comportamento reflete a dificuldade inerente do problema SAT e a ineficiência do *Backtracking* para instâncias maiores.

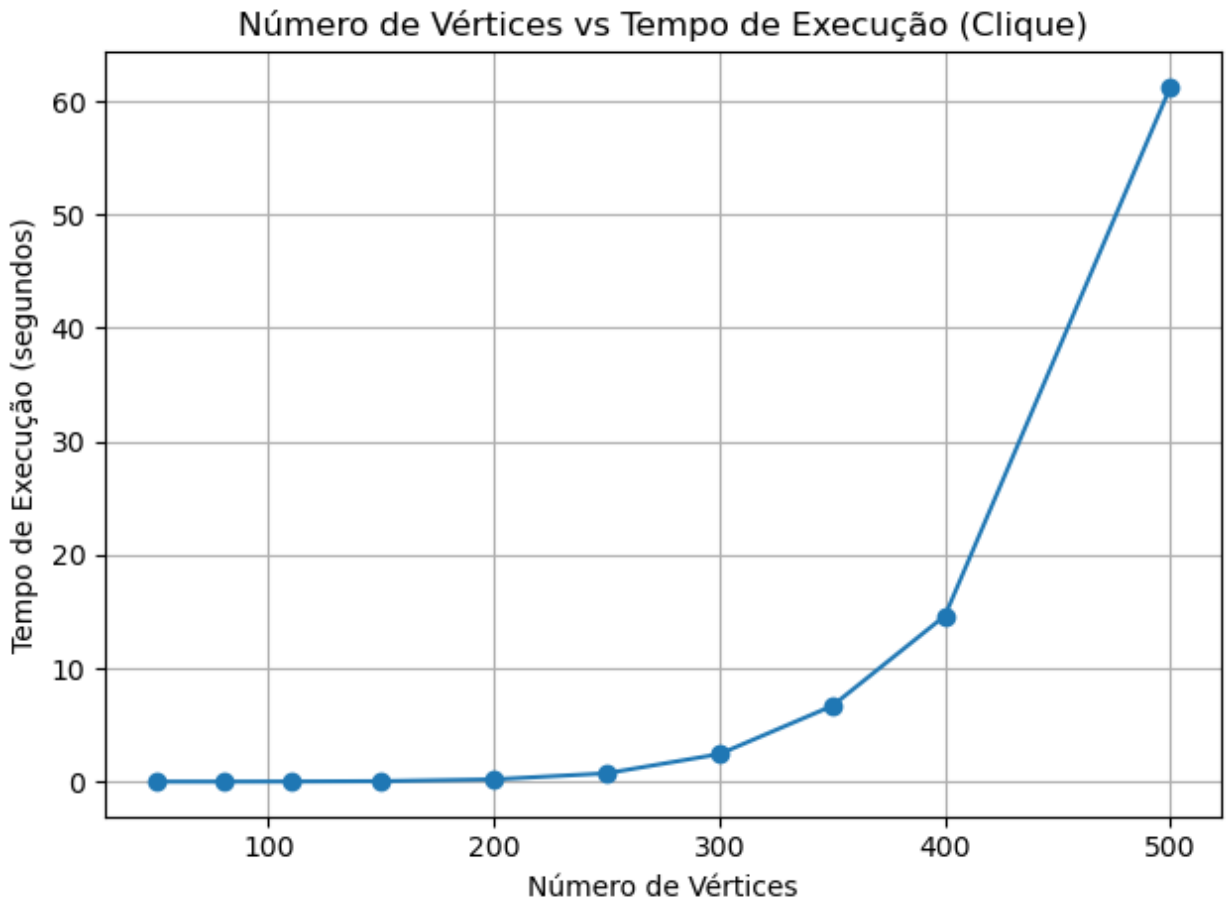

Figura 5: Gráfico de tempos de execução para as dez entradas geradas pro problema Clique Máximo

2. **Clique (Branch and Bound)**: O gráfico mostra que o tempo de execução do algoritmo *Branch and Bound* para resolver o problema do Clique aumenta rapidamente conforme o número de vértices cresce. Esse aumento é esperado, pois a complexidade do *Branch and Bound* para esse problema é exponencial no pior caso, aproximadamente $O(2^n)$, onde n é o número de vértices. O método tenta construir soluções parciais e poda ramos que não podem conter a solução ótima, mas, mesmo com essas podas, o número de subconjuntos potenciais que o algoritmo precisa explorar cresce exponencialmente à medida que o número de vértices aumenta. Por isso, o tempo de execução se mantém baixo para instâncias menores, mas cresce significativamente após 300 vértices, chegando a ultrapassar 60 segundos para 500 vértices.

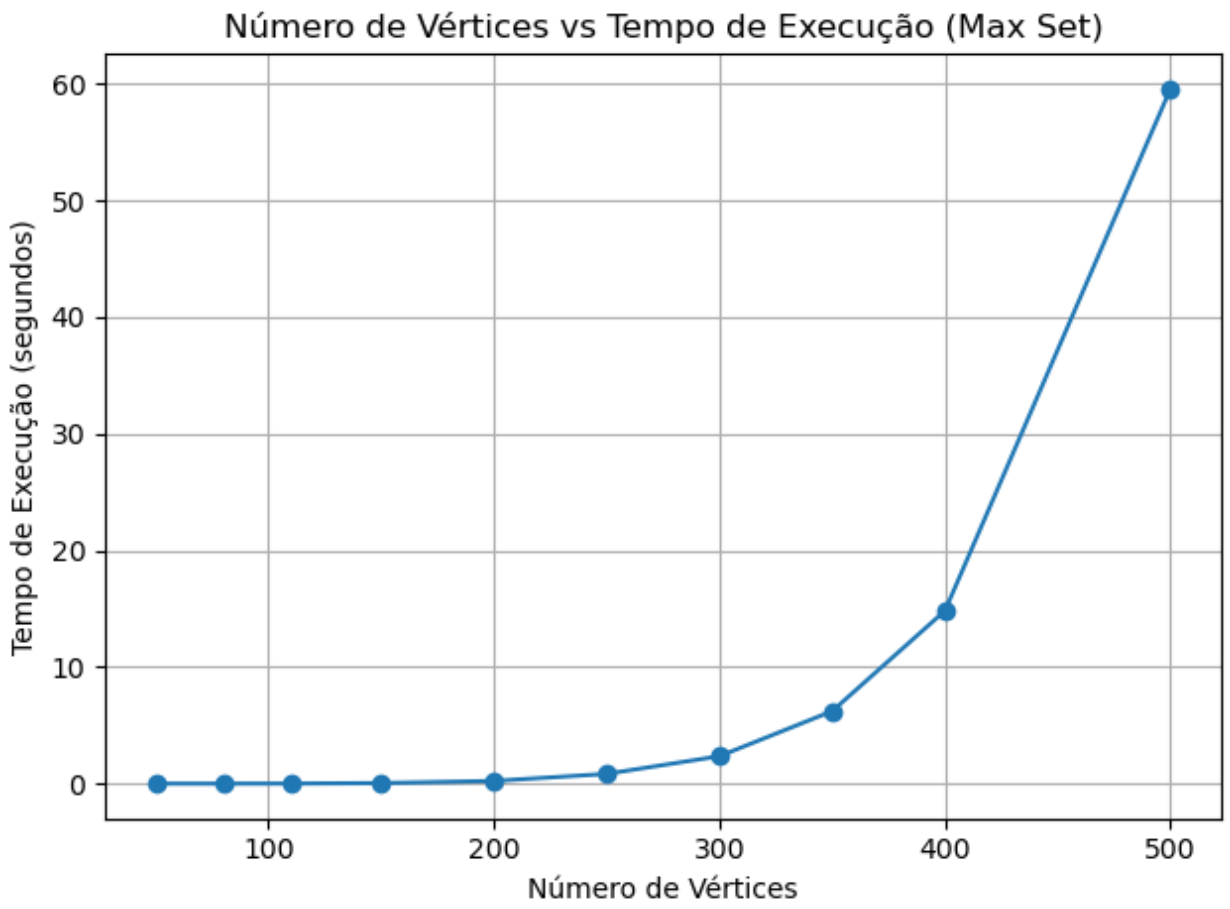

Figura 6: Gráfico de tempos de execução para as dez entradas geradas pro problema Conjunto Independente Máximo

3. **Conjunto Independente Máximo (Redução do Clique)**: O problema do Conjunto Independente Máximo foi resolvido utilizando uma redução para o problema do Clique. A complexidade da resolução permanece a mesma, uma vez que a transformação entre os dois problemas não altera a ordem de complexidade. Como o Clique é um problema **NP-completo**, o Conjunto Independente Máximo também possui complexidade exponencial no pior caso, cerca de $O(2^n)$. Isso é evidenciado pelo comportamento do gráfico, que é quase idêntico ao do Clique. O tempo de execução é baixo para instâncias com menos de 200 vértices, mas cresce rapidamente à medida que o número de vértices se aproxima de 500.

Referências

Bomze, I. M., Budinich, M., Pardalos, P. M., & Pelillo, M. (1999). The maximum clique problem. *Handbook of Combinatorial Optimization*, *4*, 1–74.

Cook-levin theorem. (n.d.). https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem. (Accessed: 2024-09-22)

Design and analysis of algorithms. (n.d.). https://www.geeksforgeeks.org/design-and-analysis-of-algorithms/. (Accessed: 2024-09-22)

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103).

Knuth, D. E. (2015). *The art of computer programming, volume 4, fascicle 6: Satisfiability*. Addison-Wesley Professional.

Papadimitriou, C. H. (1994). *Computational complexity*. Addison-Wesley.

Reduction from clique to independent set. (n.d.). https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/clique_to_independentSet.html. (Accessed: 2024-09-22)