

Problema do Casamento Máximo em Grafos

HÉRCULES APARECIDO TEIXEIRA*, Universidade Federal de Ouro Preto, Brasil

Este relatório apresenta uma abordagem para resolver o problema do casamento máximo em grafos por meio da implementação do algoritmo de Blossom, desenvolvido em C++. O algoritmo busca identificar o maior conjunto de arestas sem vértices compartilhados em grafos, utilizando uma estrutura de lista de adjacência para garantir eficiência em instâncias esparsas. Os resultados experimentais indicam que o algoritmo se mostra eficaz para instâncias de tamanho moderado, apesar dos desafios computacionais impostos pela complexidade teórica $O(n^3)$.

ACM Reference Format:

Hércules Aparecido Teixeira. 2025. Problema do Casamento Máximo em Grafos. 1, 1 (March 2025), 8 pages.

1 Repositório do Projeto

Os arquivos utilizados neste projeto estão disponíveis no seguinte repositório do GitHub: [Repositório no GitHub](#).

2 Introdução

O problema do casamento máximo em grafos consiste em encontrar o maior conjunto de arestas que não compartilham vértices, tendo aplicações relevantes em áreas como alocação de recursos, redes de computadores e biologia computacional. Inicialmente, a solução proposta para esse problema foi uma abordagem gulosa simples, onde cada vértice seria emparelhado com o primeiro vizinho disponível. Entretanto, análises teóricas e experimentais demonstraram que essa estratégia não era capaz de capturar toda a complexidade dos grafos, especialmente em situações onde ciclos ímpares, que podem comprometer um emparelhamento ótimo, estão presentes.

Diante dessa limitação, optou-se por adotar o algoritmo de Blossom, proposto por Jack Edmonds, que combina de maneira inovadora técnicas gulosas com os paradigmas de transformação e redução, além de divisão e conquista. Em essência, o algoritmo de Blossom realiza escolhas locais para ampliar o emparelhamento, mas também identifica e contrai ciclos ímpares (os chamados *blossoms*), reduzindo o problema original a subproblemas mais simples. Após a resolução desses subproblemas, a solução é reconstruída para o grafo original, garantindo que o emparelhamento máximo seja efetivamente alcançado.

Este trabalho foi implementado em C++ e estruturado em duas etapas principais: a geração de grafos aleatórios e a execução do algoritmo de Blossom. Para a representação dos grafos, utilizou-se uma lista de adjacência, que se mostra uma escolha eficiente para grafos esparsos, permitindo uma manipulação compacta e rápida dos dados. A metodologia adotada para o algoritmo de Blossom integrou estratégias gulosas com métodos de transformação, redução e divisão e conquista, proporcionando uma solução robusta para o problema mesmo em instâncias complexas.

Author's Contact Information: Hércules Aparecido Teixeira, hercules.teixeira@aluno.ufop.edu.br, Universidade Federal de Ouro Preto, João Monlevade, Minas Gerais, Brasil.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Nas seções seguintes, detalham-se as decisões de projeto, a análise de complexidade teórica e prática da solução e os resultados experimentais obtidos. Em síntese, os resultados demonstram que o algoritmo de Blossom é eficaz para instâncias de tamanho moderado, embora a complexidade $O(n^3)$ possa representar desafios computacionais para grafos de grande porte.

Descrição

Dado um grafo, um **casamento** (também conhecido como emparelhamento ou *matching*) é um **conjunto independente de arestas**, ou seja, um conjunto de arestas sem vértices em comum.

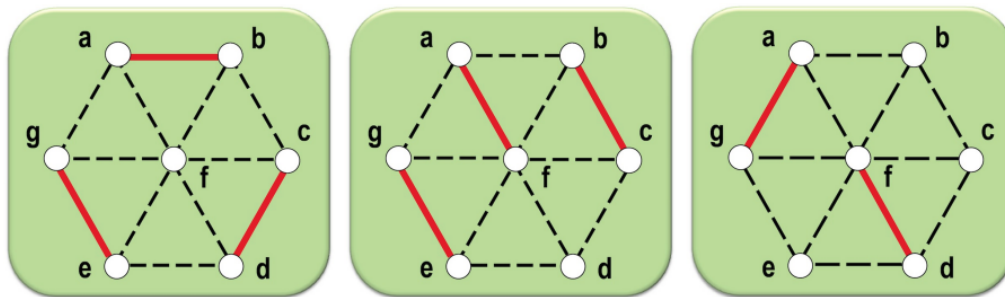


Fig. 1. Exemplo de casamento em grafos. As arestas em vermelho formam um casamento.

Além disso, é importante distinguir entre casamento **maximal** e **máximo**. Um casamento é maximal se a adição de qualquer outra aresta ocasionar sobreposição de vértices; já o casamento máximo é aquele que possui o maior número possível de arestas.

Casamento Maximal e Casamento Máximo

Um casamento é considerado **maximal** caso a adição de alguma aresta descaracterize o casamento.

Um casamento é considerado **máximo** caso possua o maior número de arestas possível, ou seja, caso seja o maior casamento possível no grafo.

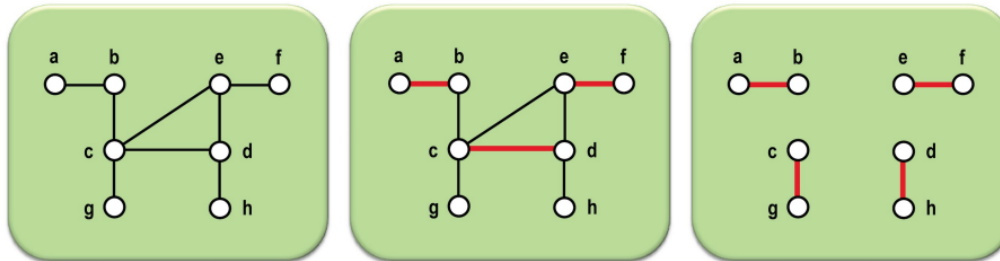


Fig. 2. Casamento maximal e casamento máximo.

3 Decisões de Projeto

3.1 Estruturas de Dados

Para representar os grafos, utilizou-se a **lista de adjacência**, que se mostra ideal para grafos esparsos, pois armazena apenas as arestas existentes. Durante a geração dos grafos, um conjunto foi empregado para evitar duplicação de arestas e assegurar a ausência de laços, mantendo a integridade dos dados.

Conjunto de Arestas: Durante a geração dos grafos, um `set<pair<int, int>` foi utilizado para evitar duplicação de arestas e garantir que não houvesse laços (arestas do tipo $u \rightarrow u$). Essa abordagem garante a integridade do grafo e evita redundâncias.

3.2 Geração de Grafos

A geração dos grafos foi realizada de forma aleatória utilizando o gerador de números Mersenne Twister (mt19937), com semente baseada no tempo atual. O número de vértices foi definido aleatoriamente (por exemplo, entre 1 e 200.000), e o número de arestas variou entre 0 e o número de vértices. Essa escolha visa equilibrar a complexidade das instâncias, evitando grafos excessivamente densos.

3.3 Algoritmo de Blossom

O algoritmo de Blossom resolve o problema do casamento máximo ao identificar e contrair ciclos ímpares em um grafo, reduzindo-o a uma instância mais simples. Após encontrar um emparelhamento máximo no grafo contraído, o ciclo é "descontraído" para reconstruir a solução no grafo original. Este processo combina:

- **Abordagem Gulosa:** O Blossom é essencialmente um algoritmo guloso, pois, em cada passo, ele faz escolhas locais ótimas para maximizar o número de arestas no emparelhamento atual. Por exemplo, o algoritmo tenta

adicionar arestas ao emparelhamento de forma a cobrir o maior número possível de vértices, sem se preocupar com decisões futuras. Essa estratégia garante que, localmente, o emparelhamento seja sempre o maior possível.

- **Divisão e Conquista:** O Algoritmo de Blossom também utiliza técnicas de **divisão e conquista** ao lidar com ciclos ímpares, conhecidos como **blossoms**. Quando um ciclo ímpar é encontrado, o algoritmo "contrai" o ciclo em um único vértice, reduzindo o problema a um grafo menor. Essa contração divide o problema original em subproblemas mais simples, que são resolvidos recursivamente. Após a resolução do subproblema, o ciclo é "descontraído", e a solução é reconstruída. Esse processo de divisão, resolução e reconstrução é uma marca clássica do paradigma de divisão e conquista.
- **Transformação e Redução:** Além disso, o Blossom emprega técnicas de **transformação e redução** para simplificar o problema. A contração de blossoms é um exemplo de transformação: o grafo original é transformado em um grafo menor, onde o problema de emparelhamento é mais fácil de resolver. Após a solução no grafo transformado, o algoritmo aplica uma redução inversa para reconstruir a solução no grafo original. Essa abordagem permite que o Blossom lide eficientemente com estruturas complexas, como ciclos ímpares, que seriam difíceis de tratar diretamente.

4 Análise de Complexidade

4.1 Complexidade do Algoritmo de Blossom

Tempo: A complexidade teórica do algoritmo de Blossom é $O(n^3)$, onde n é o número de vértices. Isso ocorre porque, no pior caso, o algoritmo precisa percorrer todos os vértices e arestas múltiplas vezes, realizando operações de busca em profundidade para cada vértice e lidando com a contração de ciclos ímpares.

Espaço: A complexidade de espaço é $O(n + m)$, onde n é o número de vértices e m é o número de arestas. Essa complexidade é devida ao uso da lista de adjacência para armazenar o grafo e do vetor de emparelhamentos para rastrear os pares de vértices.

4.1.1 Prova da Complexidade do Algoritmo de Blossom. A complexidade $O(n^3)$ do algoritmo de Blossom pode ser demonstrada considerando as operações fundamentais realizadas:

1. A busca por caminhos aumentantes utiliza uma abordagem de busca em profundidade (DFS), que percorre as arestas do grafo em tempo $O(m)$.
2. Cada fase de emparelhamento máximo realiza no máximo $O(n)$ operações.
3. Como o algoritmo de emparelhamento requer no máximo $O(n)$ iterações para processar todas as fases de contração e expansão de ciclos ímpares, o tempo total resulta em $O(n^3)$.

Dessa forma, combinando essas operações, temos a complexidade final de $O(n^3)$.

4.2 Complexidade da Geração de Grafos

Tempo: A geração de grafos tem complexidade $O(m)$, onde m é o número de arestas. Isso ocorre porque cada aresta é inserida no conjunto de arestas e na lista de adjacência, operações que são realizadas em tempo constante.

Espaço: A complexidade de espaço é $O(n + m)$, devido ao armazenamento da lista de adjacência e do conjunto de arestas. Essa complexidade é adequada para a maioria dos casos práticos, permitindo a geração de grafos com até 10.000 vértices e arestas.

5 Implementação e Testes

Foram geradas 500 instâncias de grafos, com número de vértices variando amplamente (até 100.000 e, em testes posteriores, até 200.000). Cada grafo foi salvo individualmente no formato `grafo_x.txt`, e o emparelhamento máximo foi computado e armazenado em arquivos `emparelhamento_x.txt`. Adicionalmente, o tempo de execução de cada instância foi medido e consolidado em um arquivo `resultados.txt`.

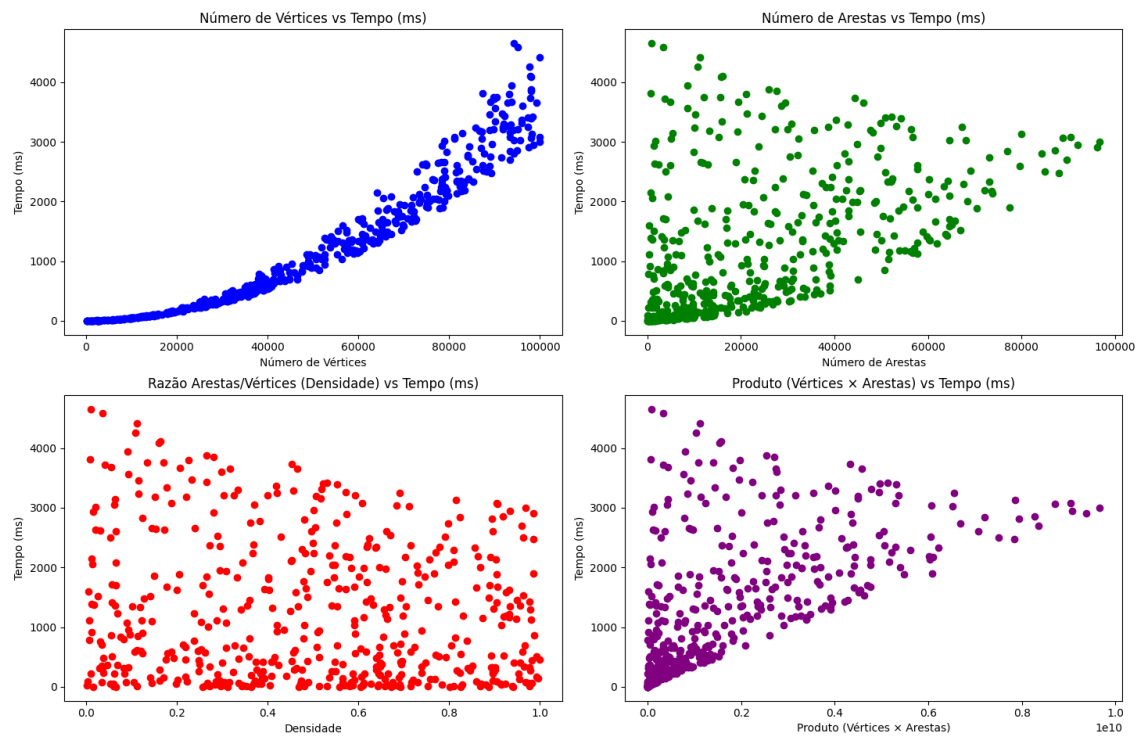


Fig. 3. Gráficos de Dispersão com 500 instâncias de até 100.000 vértices.

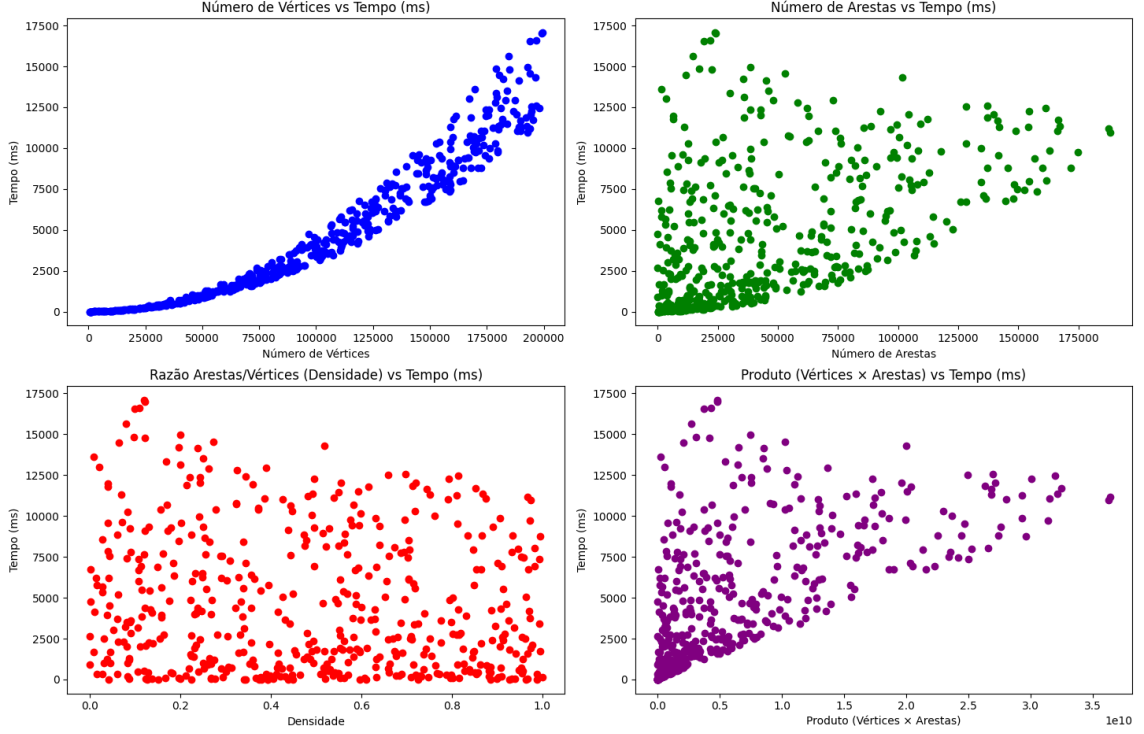


Fig. 4. Gráficos de Dispersão com 500 instâncias de até 200.000 vértices.

Adicionalmente, incluímos uma imagem com o gráfico de dispersão obtido utilizando exclusivamente a metodologia gulosa. Essa abordagem é geralmente mais rápida, pois emparelha cada vértice com o primeiro vizinho livre encontrado, o que resulta em uma execução mais simples e com menor sobrecarga computacional. Em termos de complexidade, para grafos esparsos o algoritmo guloso tende a operar em $O(n + m)$; entretanto, em casos de grafos densos, o pior cenário pode alcançar $O(n^2)$. Embora a abordagem gulosa apresente ganhos de desempenho, ela não garante encontrar o emparelhamento máximo, motivo pelo qual o algoritmo de Blossom, que integra técnicas de transformação, redução e divisão e conquista, foi adotado como solução principal.

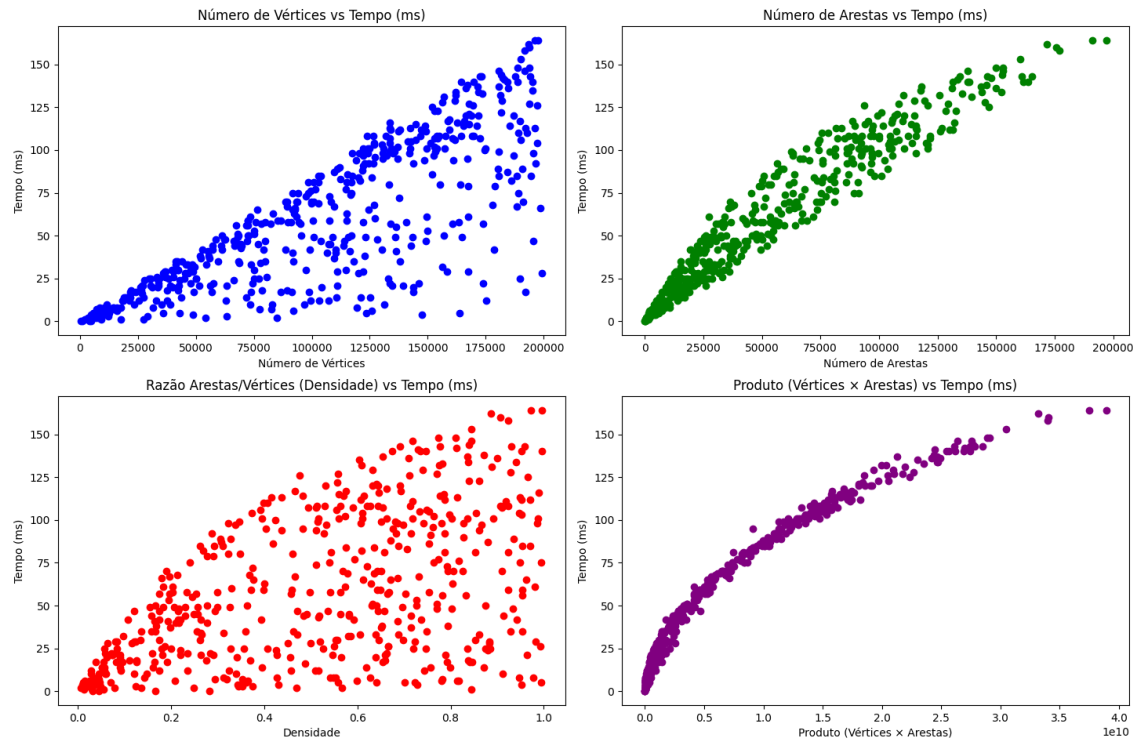


Fig. 5. Gráficos de Dispersão com 500 instâncias de até 200.000 vértices.

6 Conclusão

A implementação do algoritmo de Blossom demonstrou ser uma solução robusta para o problema do casamento máximo em grafos, superando as limitações da abordagem meramente gulosa. Embora a ideia inicial fosse empregar um método guloso, essa estratégia revelou-se insuficiente para garantir a máxima eficiência em todos os casos, motivando a adoção do Blossom – que integra estratégias gulosas com divisão e conquista e transformação e redução. Os testes realizados indicam que o algoritmo funciona bem para grafos de tamanho moderado, embora a complexidade $O(n^3)$ possa representar desafios computacionais para instâncias muito grandes.

Para trabalhos futuros, recomenda-se:

- **Otimizações de Desempenho:** Investigar heurísticas e técnicas de poda para reduzir iterações e aprimorar a eficiência.
- **Comparação com Outras Abordagens:** Explorar métodos alternativos, como algoritmos baseados em fluxo de rede, para avaliar o balanço entre eficiência e precisão.
- **Implementação Paralela:** Desenvolver versões paralelizadas para aproveitar arquiteturas multi-core e acelerar o processamento de grafos grandes.

Com essas melhorias, espera-se ampliar a aplicabilidade do emparelhamento máximo em problemas práticos que demandem alto desempenho computacional.

References

- [1] Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of Mathematics* 17: 449–467.
Disponível em: <https://doi.org/10.4153/CJM-1965-045-4>
- [2] Thomas H. Cormen et al. 2009. *Introduction to Algorithms* (3ª ed.). MIT Press. (Capítulo 25: Emparelhamento em Grafos)
- [3] Elton Máximo Cardoso. 2025. *Notas de Aula: Projeto e Análise de Algoritmo*. Universidade Federal de Ouro Preto.