Problema do Casamento Máximo em Grafos

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

Este relatório descreve a implementação do algoritmo de Blossom para resolver o problema do Casamento Máximo em Grafos, que consiste em encontrar o maior conjunto de arestas que não compartilham vértices em comum. O trabalho foi desenvolvido em C++ e dividido em duas etapas principais: a geração de grafos aleatórios e a execução do algoritmo de Blossom. Foi utilizada uma lista de adjacência para representação eficiente dos grafos. O algoritmo de Blossom, baseado na contração de ciclos ímpares e em busca em profundidade, foi aplicado para encontrar os emparelhamentos máximos, com complexidade teórica de $O(n^3)$ no pior caso. Os resultados mostraram que o algoritmo é eficiente para grafos de tamanho moderado, mas enfrenta limitações em grafos muito grandes devido ao consumo de memória e tempo de execução.

ACM Reference Format:
Hércules Aparecido Teixeira. 2025. Problema do Casamento Máximo em Grafos. 1, 1 (March 2025), 6 pages.

1 Introdução

O problema do Emparelhamento Máximo em Grafos consiste em encontrar o maior conjunto de arestas que não compartilham vértices em comum. Esse problema tem aplicações em diversas áreas, como alocação de recursos, redes de computadores e biologia computacional. Este trabalho teve como objetivo implementar o algoritmo de Blossom para resolver o problema do emparelhamento máximo, utilizando grafos gerados aleatoriamente para testar a eficiência da solução.

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

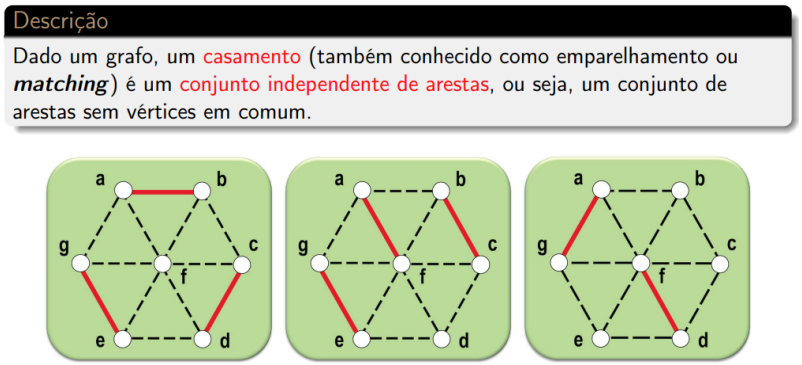

Fig. 1. Exemplo de casamento em grafos. As arestas em vermelho formam um casamento.

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

Além disso, é importante distinguir entre casamento maximal e casamento máximo. Um casamento é considerado **maximal** se a adição de alguma aresta ao conjunto faria com que duas arestas compartilhassem um vértice. Já um casamento é **máximo** quando possui o maior número de arestas possível no grafo.

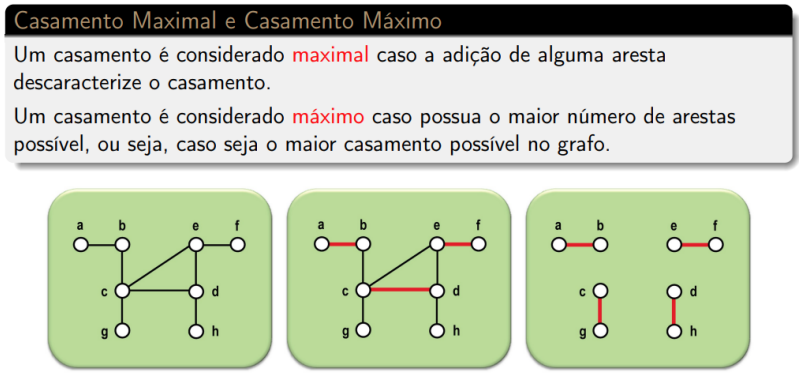

Fig. 2. Casamento maximal e casamento máximo.

O código foi desenvolvido em C++ e dividido em duas partes principais: a geração de grafos aleatórios e a execução do algoritmo de Blossom para encontrar o emparelhamento máximo. O relatório a seguir descreve as decisões de projeto, a análise de complexidade, os resultados obtidos e as conclusões do trabalho.

2 Decisões de Projeto

2.1 Estruturas de Dados

Lista de Adjacência: O grafo foi representado usando uma lista de adjacência, uma escolha eficiente para grafos esparsos, pois armazena apenas as arestas existentes. Essa estrutura permite uma representação compacta e facilita a manipulação de grafos com muitos vértices e poucas arestas.

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 $u \longrightarrow u$). Essa abordagem garante a integridade do grafo e evita redundâncias.

2.2 Geração de Grafos

Número de Vértices e Arestas: Para cada grafo, o número de vértices foi escolhido aleatoriamente entre **1 e N**, e o número de arestas foi definido igual ao número de vértices. Essa decisão foi tomada para equilibrar a complexidade do problema, evitando grafos excessivamente densos que poderiam comprometer o desempenho do algoritmo.

Aleatoriedade: A geração de grafos foi feita utilizando o gerador de números aleatórios Mersenne Twister (`mt19937`), baseado no tempo atual, que garante alta qualidade na distribuição dos números. Essa escolha é crucial para garantir que os grafos gerados sejam representativos e variados, permitindo testes robustos do algoritmo.

2.3 Algoritmo de Blossom

Busca em Profundidade (DFS) e Contração de Ciclos Ímpares: O algoritmo de Blossom foi implementado utilizando uma abordagem baseada em busca em profundidade (DFS) combinada com a contração de ciclos ímpares, permitindo

encontrar e expandir emparelhamentos. A função `buscaProfundidade` verifica se um vértice pode ser emparelhado com outro, garantindo que não haja conflitos. Essa abordagem é eficiente para grafos de tamanho moderado e permite uma implementação direta do conceito de emparelhamento.

2.4 Solução via Algoritmo de Blossom

O Algoritmo de Blossom, proposto por Jack Edmonds, é uma solução eficiente para o problema de encontrar um **emparelhamento máximo** em grafos não direcionados. Esse algoritmo combina características de múltiplos paradigmas de projeto de algoritmos, incluindo **algoritmos gulosos**, **divisão e conquista** e **transformação e redução**, tornando-o uma abordagem poderosa e versátil.

2.4.1 ***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.

2.4.2 ***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.

2.4.3 ***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.

2.4.4 ***Exemplo Prático.*** Considere um grafo com um ciclo ímpar de 5 vértices. O Blossom identifica esse ciclo, contrai-o em um único vértice e resolve o problema de emparelhamento no grafo reduzido. Uma vez que o emparelhamento é encontrado no grafo transformado, o ciclo é descontraído, e o emparelhamento é estendido para o grafo original. Esse processo ilustra como o algoritmo combina estratégias gulosas, divisão e conquista, e transformação e redução para resolver o problema de forma eficiente.

3 Análise de Complexidade

3.1 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.

3.1.1 ***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 $O(n^3)$.

3.2 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.

4 Implementação e Testes

Foram geradas **500 instâncias de grafos**, cada uma com um número aleatório de vértices no intervalo entre **1 e 100.000** e um número de arestas variando de **0 até o número de vértices**. Cada grafo foi salvo em um arquivo individual no formato **grafo_x.txt** para posterior processamento.

Para cada grafo, o algoritmo de emparelhamento foi executado, e os resultados foram armazenados em arquivos separados no formato **emparelhamento_x.txt**. Além disso, o tempo de execução foi medido para cada instância, e os resultados consolidados foram salvos em um arquivo único (**resultados.txt**).

Em um segundo conjunto de testes, foram utilizadas **500 instâncias** com um intervalo de vértices ampliado para **1 a 200.000**.

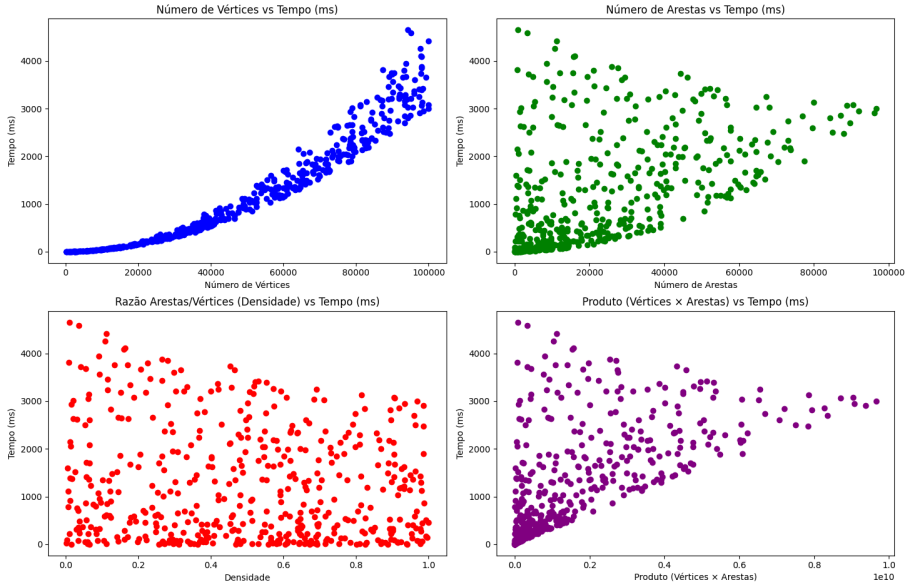

Fig. 3. Gráficos de Dispersão com 500 instancias de até 100.000

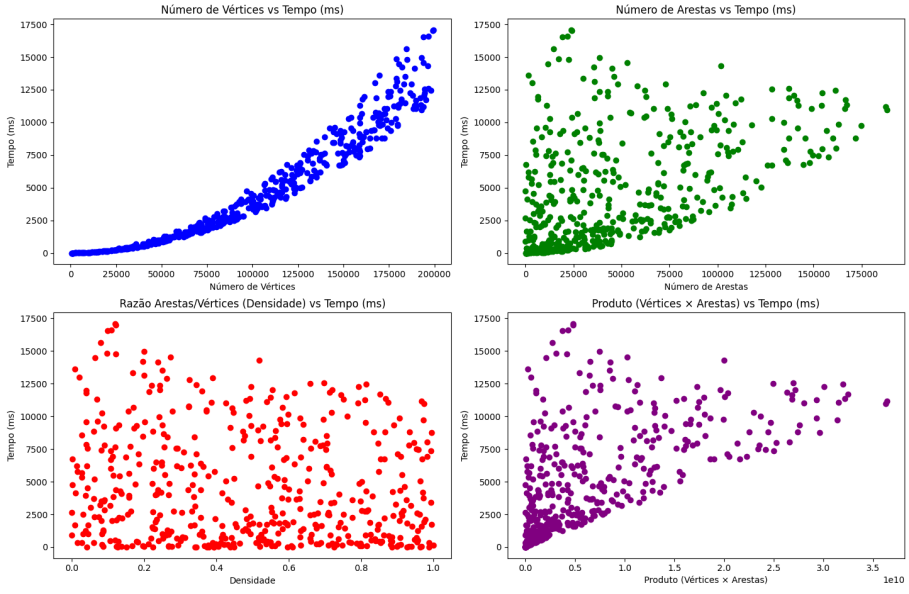

Fig. 4. Gráficos de Dispersão com 500 instancias de até 200.000

5 Conclusão

Este trabalho implementou com sucesso o algoritmo de Blossom para resolver o problema do casamento máximo em grafos. A geração de grafos aleatórios permitiu avaliar a eficiência da solução em diferentes cenários, e os resultados obtidos corroboraram a complexidade teórica esperada de $O(n^3)$. Apesar do alto custo computacional, o algoritmo demonstrou ser eficaz para grafos de tamanho moderado.

Para trabalhos futuros, sugere-se:

- **Otimizações de desempenho:** Explorar melhorias na implementação, como heurísticas para reduzir o número de iterações ou técnicas de poda na busca em profundidade, visando minimizar o tempo de execução e o consumo de memória, especialmente para grafos mais densos com até $O(n(n-1)/2)$ arestas.
- **Comparação com outras abordagens:** Investigar alternativas ao algoritmo de Blossom, incluindo algoritmos gulosos e métodos baseados em fluxo de rede, para avaliar compromissos entre eficiência e precisão na busca por emparelhamentos máximos.
- **Implementação paralela:** Considerar versões paralelizadas do algoritmo para aproveitar arquiteturas multi-core, visando acelerar o processamento de grafos grandes.

Com essas direções, espera-se avançar na aplicação eficiente do emparelhamento máximo em problemas práticos que exigem alto desempenho computacional.