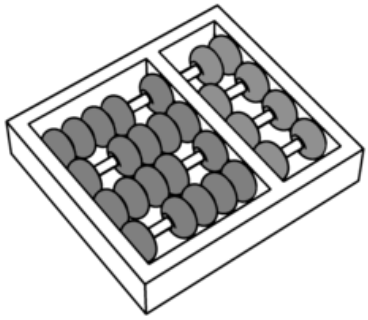
Universidade Estadual de Campinas

Instituto de Computação



Proposta de Dissertação de Mestrado

Partição de Grafos Eulerianos em Circuitos

Candidato: Pedro Olímpio Nogueira de Oliveira Pinheiro

Orientador: Prof. Dr. Cid Carvalho de Souza

Coorientador: Prof. Dr. Zanoni Dias

Resumo

Neste trabalho vamos propor algoritmos para resolver o problema de particionar o conjunto de arestas de um grafo em subconjuntos, de forma que o subgrafo induzido por cada subconjunto de arestas é um circuito, visando maximizar a quantidade de subconjuntos da partição. Projetaremos também algoritmos para uma variação desse problema em que as arestas do grafo possuem cores (pretas ou cinza) e os subgrafos induzidos pelos subconjuntos de arestas são circuitos alternados, isto é, arestas consecutivas no circuito têm cores diferentes. Investigaremos aplicações desses problemas de particionamento ao problema da ordenação por reversão, o qual tem grande relevância na área de biologia computacional no estudo da distância evolutiva entre espécies. Desenvolveremos modelos de programação linear inteira que permitam resolver os problemas de forma exata e algoritmos que utilizem heurísticas construtivas e meta-heurísticas para obter boas soluções. Por fim, vamos realizar experimentos para comparar o desempenho dos algoritmos que apresentaremos.

1 Introdução

O problema de particionar o conjunto de arestas de um grafo em uma classe específica de subgrafos é um problema recorrente na literatura [6, 15, 16]. Neste trabalho trataremos da partição de um grafo em circuitos, ou seja, da separação das arestas do grafo em uma coleção de subconjuntos disjuntos, cuja união é igual ao conjunto de arestas do grafo e cada um deles corresponde a um circuito. Diferente dos trabalhos de Mynhardt e van Bommel [15] e Rodger [16], no problema que abordamos os subconjuntos da partição não precisam ser cópias de um mesmo grafo, mas apenas circuitos de tamanho qualquer. Porém, visamos encontrar a partição com o número máximo de circuitos.

Caprara [4] mostrou que existe uma relação entre o problema de ordenação por reversão, o problema de encontrar a máxima partição em circuitos de um grafo e o problema da máxima partição em circuitos de um grafo com cores nas arestas (pretas e cinza) de maneira que arestas consecutivas nos circuitos tenham cores diferentes.

Ordenação por reversão é um problema da área de biologia computacional com grande relevância no estudo da distância evolucionária entre espécies [11].

Este trabalho segue a seguinte estrutura: na Seção 2 definimos os principais

conceitos necessários para compreensão do trabalho; na Seção 3 descrevemos formalmente os problemas que abordamos; na Seção 4 mostramos quais são os objetivos; na Seção 5 listamos trabalhos relacionados presentes na literatura; na Seção 6 apresentamos a metodologia utilizada no desenvolvimento do trabalho; na Seção 7 apresentamos e avaliamos os resultados preliminares; na Seção 8 mostramos o plano de trabalho.

2 Fundamentação Teórica

Nesta seção descreveremos os principais conceitos utilizados neste trabalho.

As definições relacionadas a grafos são baseadas no livro de Bondy e Murty [2], a menos quando dito ao contrário.

Um **grafo** G é um par ordenado (V, E), onde V é um conjunto de vértices e E é um conjunto de arestas (pares não ordenados de vértices). Se dois vértices $u, v \in V$ possuem uma aresta entre si, ou seja, $(u, v) \in E$, dizemos que u e v são vizinhos ou adjacentes. Os vértices em que uma aresta incide são chamados seus extremos. Neste trabalho vamos considerar que todos os grafos são simples, ou seja, os dois extremos de cada aresta são distintos e não existem duas arestas com os mesmos extremos. A Figura 1 mostra um exemplo de um grafo com quatro vértices $\{a, b, c$ e $d\}$ e seis arestas $\{(a, b), (b, c), (c, d), (d, a), (a, c)$ e $(b, d)\}$.

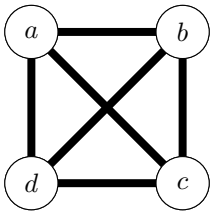


Figura 1: Exemplo de um grafo com quatro vértices e seis arestas.

O **grau** de um vértice é o número de arestas que incidem nesse vértice. O grau de um vértice v é denotado por $d(v)$.

Um grafo $G' = (V', E')$ é um **subgrafo** de um grafo $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$ e as extremidades das arestas de E' estão em V'. Ou seja, G' pode ser obtido a partir de G removendo vértices e arestas. O **subgrafo induzido** por um conjunto de arestas $E' \subseteq E$ de um grafo $G = (V, E)$ é o subgrafo de G cujo conjunto de vértices são

os extremos das arestas em E' e o conjunto de arestas é E'.

Uma **trilha** em um grafo é uma sequência de vértices e arestas $(v_0, e_0, v_1, \ldots, v_{l-1}, e_{l-1}, v_l)$, em que v_i e v_{i+1} são os vértices extremos da aresta e_i. Note que pode haver repetição de vértices, mas não de arestas.

Um **caminho** em um grafo é uma trilha sem repetição de vértices. Um **circuito** em um grafo é uma trilha em que o primeiro vértice é igual ao último. O comprimento de um caminho (ou circuito) é a quantidade de arestas desse caminho (ou circuito).

Os grafos da Figura 2 ilustram um caminho e um circuito no grafo da Figura 1, onde as arestas vermelhas representam as arestas do caminho e do circuito.

(a) (b)

Figura 2: Exemplo de (a) caminho e de (b) circuito.

Um grafo $G = (V, E)$ é **conexo** quando existe um caminho em G, que começa em u e termina em v, para qualquer par de vértices $u, v \in V$.

Uma **trilha euleriana** é uma trilha que passa por cada aresta do grafo exatamente uma vez. Um **circuito euleriano** é um circuito que passa por cada aresta do grafo exatamente uma vez. Um **grafo euleriano** é um grafo em que existe um circuito euleriano.

Uma **partição** das arestas de um grafo $G = (V, E)$ é uma coleção de subconjuntos $\{E_1, E_2, \ldots\}$ de E em que a interseção de qualquer dois subconjuntos é vazia e a união de todos os subconjuntos é igual a E. Uma **partição em circuitos** de um grafo $G = (V, E)$ é uma partição de E em subconjuntos $\{E_1, E_2, \ldots\}$ tal que o subgrafo induzido por cada E_i é um circuito. A Figura 3 mostra um grafo e sua partição em dois circuitos, um com arestas vermelhas e outro com arestas azuis.

Chamaremos de **grafos bicoloridos** os grafos cujas arestas são divididas em dois conjuntos, pretas (P) e cinza (C). Um **caminho alternado** em um grafo bicolorido é um caminho em que duas arestas consecutivas possuem cores diferentes. Um **circuito alternado** é um circuito em que duas arestas consecutivas possuem cores diferentes.

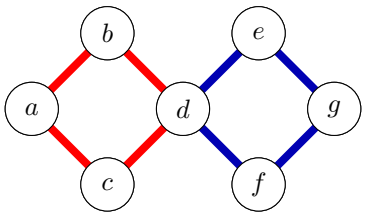


Figura 3: Exemplo de partições em circuitos.

Um **grafo de breakpoint** [1] é um grafo bicolorido $G(\pi)$, construído com base em uma permutação dos $n+2$ primeiros números naturais, que serão os vértices do grafo. Descreveremos a seguir o processo de construção de um grafo de *breakpoint*. Seja $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ uma permutação dos n primeiros inteiros positivos. Adicione o elemento $\pi_0 = 0$ no início da permutação e o elemento $\pi_{n+1} = n+1$ no final. Os vértices do grafo serão os $n+2$ números da permutação estendida. Para cada $i \in \{0, 1, \ldots, n\}$, se $|\pi_i - \pi_{i+1}| \neq 1$ adicione uma aresta preta de π_i para π_{i+1}. Seja π_i^{-1} a posição que o número i está na permutação π, de modo que $\pi_{\pi_i}^{-1} = i$. Para cada $i \in \{0, 1, \ldots, n\}$, se $|\pi_i^{-1} - \pi_{i+1}^{-1}| \neq 1$ adicione uma aresta cinza de π_i^{-1} para π_{i+1}^{-1}. Por exemplo, o grafo da Figura 4 representa o grafo de *breakpoint* para a permutação $\pi = (1, 3, 6, 2, 4, 5)$.

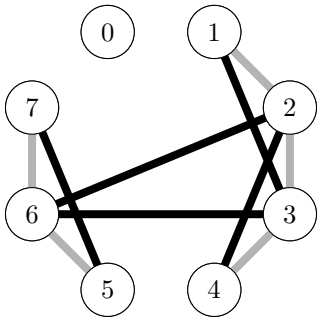


Figura 4: Exemplo de grafo de *breakpoint* para a permutação $\pi = (1, 3, 6, 2, 4, 5)$.

3 Partição em Circuitos

Nesta seção apresentaremos os problemas da Máxima Partição em Circuitos (*Maximum Eulerian Cycle Decomposition* - MAX-ECD), da Máxima Partição em Circuitos Alternados (*Maximum Alternating-Cycle Decomposition* - MAX-ACD) e suas aplicações para o problema de Ordenação por Reversão (*Minimum Sorting by Reversals* - MIN-SBR). Caprara [4] mostrou que todos esses problemas (MAX-ECD, MAX-ACD

e MIN-SBR) são da classe NP-difícil.

O MAX-ECD é definido como: dado um grafo euleriano $G = (V, E)$, encontrar a partição em circuitos de maior cardinalidade, ou seja, particionar G no maior número possível de circuitos. A Figura 5 ilustra dois exemplos de Partição em Circuitos. A primeira partição tem cardinalidade 2, onde o primeiro circuito é composto pelas arestas azuis e o segundo pelas arestas vermelhas, e a segunda tem cardinalidade 4, e é composto pelos circuitos com as arestas vermelhas, azuis, verdes e magentas.

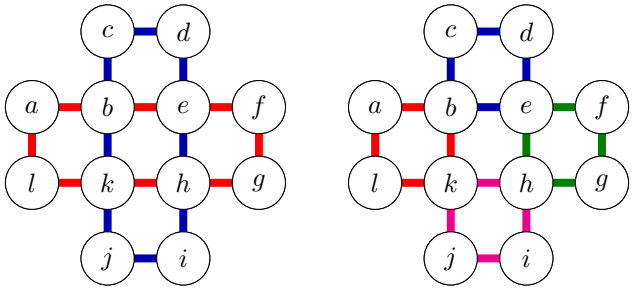


Figura 5: Exemplos de partições em circuitos de um grafo.

O MAX-ACD é definido como: dado um grafo euleriano bicolorido $G = (V, P \cup C)$ que admite uma partição em circuitos alternados, encontrar a partição em circuitos alternados de maior cardinalidade. Na Figura 6, os circuitos que passam pelos vértices (a, b, d, c) e pelos vértices (b, c, f, e) são circuitos alternados, enquanto (a, c, b) e (a, b, e, f, c) são circuitos não alternados.

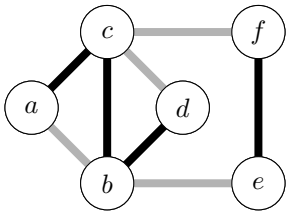


Figura 6: Exemplo de um grafo bicolorido.

Lema 3.1. *Um grafo bicolorido admite uma partição em circuitos alternados se e somente se todos os seus vértices têm uma quantidade igual de arestas pretas e arestas cinza incidentes.*

A prova do Lema 3.1 é semelhante à prova do teorema que diz que um grafo é euleriano se e somente se ele for conexo e se o grau de todos os vértices for par. No caso

do Lema 3.1, a igualdade de arestas pretas e cinza incidentes em um vértice é usada para construir circuitos alternados.

O grafo de *breakpoint* de uma permutação π sempre será um grafo com pelo menos uma partição válida em circuitos alternados, pois a quantidade de arestas pretas incidentes em cada vértice i é igual a quantidade de arestas cinza incidentes no mesmo vértice, que é a quantidade de elementos em $\{i-1, i+1\}$ que não é vizinho de i em π.

Para o MAX-ECD, qualquer solução que possua um circuito que passe mais de uma vez pelo mesmo vértice u pode ser melhorada substituindo-se esse circuito por circuitos que passam apenas uma vez por u. Porém, note que para uma solução ótima do MAX-ACD um circuito pode utilizar vértices repetidos, como o exemplo da permutação $\pi = (4, 1, 2, 3)$, cujo grafo de *breakpoint* só admite uma partição em circuitos alternados com um único circuito que utiliza o vértice 4 duas vezes. A Figura 7 ilustra o grafo de *breakpoint* da permutação π.

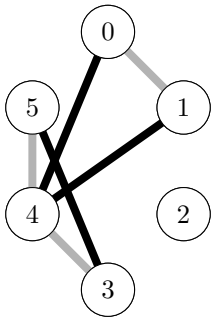


Figura 7: Exemplo de um grafo de *breakpoint* para a permutação $\pi = (4, 1, 2, 3)$ com um circuito alternado que utiliza o mesmo vértice duas vezes.

O MIN-SBR consiste em, dado uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ de $\{1, 2, \ldots, n-1, n\}$ e uma operação de reversão ρ no intervalo (i, j) que inverte a ordem dos elementos entre i e j (incluindo ambos), onde $\pi\rho = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \underline{\pi_j, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_i}, \pi_{j+1}, \ldots, \pi_n)$, encontrar a menor sequência de reversões $\rho_1, \ldots, \rho_k$ que ordene a permutação π em ordem crescente, ou seja, $\pi\rho_1 \ldots \rho_k = (1, 2, \ldots, n-1, n)$.

Caprara [4] mostrou que o MIN-SBR de uma permutação está relacionado com o MAX-ACD de seu grafo de *breakpoint*, assim como o MAX-ACD está relacionado ao MAX-ECD. Descreveremos em seguida como esses problemas se relacionam.

Para uma permutação π dizemos que $d(\pi)$ é o número de reversões necessárias

para ordenar essa permutação, que $b(\pi)$ é o número de arestas pretas no grafo de *breakpoint* de π e que $c(\pi)$ é a quantidade de circuitos na partição com maior número de circuitos alternados no grafo de *breakpoint* de π.

Bafna e Pevzner [1] mostraram que, para qualquer permutação π,

$$d(\pi) \geq b(\pi) - c(\pi). \tag{1}$$

Portanto $b(\pi) - c(\pi)$ é um limitante inferior válido para o número de reversões necessárias para ordenar uma permutação. Caprara [3] mostrou que esse limitante é frequentemente igual ao valor ótimo e que a probabilidade da desigualdade $d(\pi) > b(\pi) - c(\pi)$ ocorrer para uma permutação π de n elementos é $\Theta(1/n^5)$.

Caprara [4] mostrou como o MAX-ECD e o MAX-ACD podem ser reduzidos em tempo polinomial um para o outro. A redução do MAX-ACD para o MAX-ECD não mantém aproximações. Já a redução do MAX-ECD para o MAX-ACD mantém aproximações.

4 Objetivos

Neste trabalho proporemos algoritmos heurísticos e exatos para os problemas MAX-ECD, para grafos obtidos pela redução descrita por Caprara [4] a partir de grafos de *breakpoint* e para grafos eulerianos quaisquer. O mesmo deverá ser feito com o MAX-ACD, para grafos de *breakpoint* e para grafos bicoloridos quaisquer. Iremos propor heurísticas construtivas e algoritmos baseados em meta-heurísticas como, por exemplo, a Busca Tabu.

Criaremos um conjunto de instâncias para execução de experimentos. Até o final do Mestrado, pretendemos disponibilizar este *benchmark* em um sítio de domínio público.

Utilizando os algoritmos propostos por nós para MAX-ECD e MAX-ACD em grafos de *breakpoint* de uma permutação, iremos construir algoritmos para obter soluções para o MIN-SBR para essa permutação e comparar com soluções encontradas na literatura.

5 Trabalhos Relacionados

Lancia e Serafini [12] desenvolveram um modelo de programação linear inteira para resolver o problema de encontrar o maior número de circuitos aresta-disjuntos em um grafo (Empacotamento em Circuitos), que resolve MAX-ECD. Este modelo é dado por:

$$\max \sum_{C \in \mathcal{C}} x_C \tag{2}$$

s.a.:

$$\sum_{C \in \mathcal{C}: e \in C} x_C \leq 1 \qquad \forall e \in E \tag{3}$$

$$x_C \in \{0, 1\} \qquad \forall C \in \mathcal{C} \tag{4}$$

onde $\mathcal{C}$ é um conjunto com todos os circuitos de G e x_C é uma variável binária que vale 1 se o circuito C está na solução e 0 caso contrário. A Restrição (3) garante que não existam dois circuitos que contenham uma mesma aresta.

Lancia e Serafini [12] criaram um modelo de programação linear inteira para MAX-ACD, o qual é dado por:

$$\max \sum_{C \in \mathcal{C}_0} x_C \tag{5}$$

s.a.:

$$\sum_{C \in \mathcal{C}_0: e \in C} x_C \leq 1 \qquad \forall e \in E \tag{6}$$

$$x_C \in \{0, 1\} \qquad \forall C \in \mathcal{C}_0 \tag{7}$$

onde $\mathcal{C}_0$ é o conjunto de todos os circuitos alternados do grafo e x_C é uma variável binária que vale 1 se o circuito C está na solução e 0 caso contrário.

Moreno, Martins e Frota [14] apresentaram um modelo de programação linear inteira para um problema semelhante ao MAX-ECD, o problema *Rainbow Cycle Cover* (RCC). Nesse problema, dado um grafo com arestas coloridas (pode haver mais que

duas cores), deve-se particionar os vértices do grafo no menor número de circuitos sem repetição de vértices, respeitando a restrição que em cada circuito todas as arestas devem ter cores diferentes.

Moreno, Frota e Martins [13] apresentaram um modelo de programação linear inteira para o problema *d-Minimum Branch Vertices* (d-MBV). Esse problema é definido como: dado um grafo $G = (V, E)$ e um inteiro d, encontrar um subgrafo $G' = (V', E')$ conexo com todos os vértices e com $|V| - 1$ arestas (uma árvore geradora) utilizando o mínimo de vértices de grau maior que d em G'.

6 Metodologia

Nesta seção detalharemos as abordagens utilizadas para resolver os problemas MAX-ECD e MAX-ACD.

6.1 Modelos Exatos

Propomos um modelo de programação linear inteira para o MAX-ECD que possui uma quantidade de variáveis e de restrições na ordem de $\Theta(|E|^2)$ e $\Theta(|E|^2 + |V||E|)$, respectivamente.

Definimos um conjunto $I_C = \{1, 2, \ldots, \lfloor \frac{|E|}{3} \rfloor\}$ que indexa o conjunto de circuitos que utilizamos como solução. Observe que a maior partição em circuitos de um grafo ocorre quando todos os circuitos da partição são triângulos (circuitos de comprimento 3) e o seu tamanho é $\lfloor \frac{|E|}{3} \rfloor$. Com isso podemos afirmar que $\lfloor \frac{|E|}{3} \rfloor$ é um limitante superior para o número de circuitos de uma partição.

Para o modelo de programação linear inteira, cada aresta (u, v) do grafo terá duas variáveis associadas em cada circuito c: y_{cuv} e y_{cvu}.

As variáveis binárias x_c, $c \in I_C$, valem 1 se e somente se o circuito c está sendo utilizado na solução. As variáveis binárias y_{cuv} valem 1 se e somente se a aresta $(u, v) \in E$ está no circuito $c \in I_C$. A partir destas definições, chegamos ao seguinte modelo:

$$\max \sum_{c \in I_C} x_c \tag{8}$$

s.a.:

$$\sum_{(u,v)\in E} (y_{cuv} + y_{cvu}) \geq x_c \qquad \forall c \in I_C \tag{9}$$

$$y_{cuv} + y_{cvu} \leq x_c \qquad \forall c \in I_C, \forall (u,v) \in E \tag{10}$$

$$\sum_{c\in I_C} (y_{cuv} + y_{cvu}) = 1 \qquad \forall (u,v) \in E \tag{11}$$

$$\sum_{(u,v)\in E} y_{cuv} = \sum_{(v,u)\in E} y_{cvu} \qquad \forall u \in V, \forall c \in I_C \tag{12}$$

$$\sum_{(u,v)\in E} y_{cuv} \leq x_c \qquad \forall u \in V, \forall c \in I_C \tag{13}$$

$$x_{c+1} \leq x_c \qquad \forall c \in I_C \setminus \{\lfloor |E|/3 \rfloor\} \tag{14}$$

$$x_c \in \{0,1\} \qquad \forall c \in I_C \tag{15}$$

$$y_{cuv} \in \{0,1\} \qquad \forall (u,v) \in E, \forall c \in I_C \tag{16}$$

A Restrição (9) garante que apenas circuitos com alguma aresta serão contados na solução. A Restrição (10) certifica que todas as arestas utilizadas estão em algum circuito usado na solução. A Restrição (11) faz com que cada aresta seja utilizada exatamente uma vez. A Restrição (12) assegura que, em todos os circuitos, o grau de entrada de cada vértice é igual ao grau de saída. A Restrição (13) garante que cada circuito utiliza cada vértice no máximo uma vez.

A Restrição (14) não é necessária no modelo, porém torna o algoritmo de resolução mais eficiente ao reduzir soluções simétricas, pois força que os circuitos utilizados sejam os com menores identificadores $c \in I_C$. As restrições (10) e (13) também não são necessárias no modelo, pois a função objetivo de maximização tende a fazer as variáveis x_c valerem 1, mas tornam o algoritmo de resolução mais eficiente. As Restrições (15) e (16) impõem a integralidade das variáveis do modelo.

Note que as restrições do modelo não garantem conexidade dos circuitos, ou seja, em uma solução não ótima dois circuitos podem estar sendo considerados um só e contribuindo com apenas uma unidade na função objetivo. Porém, a maximação desta função impede que tal fato ocorra na solução ótima, pois isto implicaria na existência de uma solução com um circuito a mais que a solução ótima.

Esse modelo usa um número polinomial de restrições e variáveis, portanto, pode ser computado sem a necessidade de utilizar técnicas de geração de colunas, diferente daquele proposto por Lancia e Serafini [12].

Em sua formulação para o problema RCC, Moreno, Martins e Frota [14] criaram variáveis binárias de decisão para contar os circuitos utilizados e associar arestas aos circuitos, assim como no modelo que apresentamos neste trabalho. Porém, como no problema RCC a partição é nos vértices, os autores adicionaram também variáveis binárias para associar vértices aos circuitos. Além disso, ao contrário do nosso, no modelo daqueles autores são necessárias restrições que garantam que os circuitos encontrados sejam conexos, se não, é possível obter uma resposta inválida em que dois circuitos contribuem com apenas uma unidade na função objetivo. Isso ocorre porque, no RCC, a função objetivo requer a minimização do número de circuitos, diferentemente do MAX-ECD.

A exemplo do que fizemos, no modelo para o problema d-MVB de Moreno, Frota e Martins [13] existem duas variáveis binárias de decisão para cada aresta, uma em cada sentido.

Observe que podemos adaptar o modelo que propomos para o MAX-ECD para o problema MAX-ACD substituindo as Restrições (12) e (13) pelas Restrições (17) e (18). Essas restrições garantem que o grau de entrada de arestas cinza de um vértice v é igual ao grau de saída de arestas pretas e vice-versa.

$$\sum_{(u,v)\in P} y_{cuv} = \sum_{(v,u)\in C} y_{cvu} \qquad \forall u \in V, \forall c \in I_C \tag{17}$$

$$\sum_{(u,v)\in C} y_{cuv} = \sum_{(v,u)\in P} y_{cvu} \qquad \forall u \in V, \forall c \in I_C \tag{18}$$

6.2 Heurísticas Construtivas

Para encontrar soluções para o MAX-ECD, utilizaremos uma estratégia gulosa de encontrar no grafo um dos circuitos de menor comprimento que contém uma aresta escolhida de forma aleatória. Adicionamos esse circuito na solução, o removemos do grafo e repetimos o processo até que todas as arestas sejam removidas. O Algoritmo 1 descreve a heurística. Neste algoritmo consideramos que a rotina `caminho_BFS` da linha

6 devolve um conjunto de arestas que induz um caminho mínimo que comece em u e termine em v. Consideramos que cada conjunto que pertence à solução é um conjunto de arestas que induz um circuito.

Algoritmo 1 Heurística Construtiva

```
1: função ECD-Heurística(V, E)
2:     solução ← ∅
3:     enquanto E ≠ ∅ faça
4:         (u, v) ← aresta aleatória de E
5:         E ← E \ {(u, v)}
6:         C ← caminho_BFS(u, v, V, E)
7:         E ← E \ C
8:         C ← C ∪ {(u, v)}
9:         solução ← solução ∪ {C}
10:    fim enquanto
11:    devolve solução
12: fim função
```

Esse algoritmo pode ser adaptado para resolver o MAX-ACD alterando a escolha da aresta aleatória da linha 4 por uma aresta cinza aleatória e o caminho encontrado na linha 6 por uma trilha alternada mínima que comece e termine com uma aresta preta. O Algoritmo 2 descreve essa heurística.

Algoritmo 2 Heurística Construtiva

```
1: função ACD-Heurística(V, E)
2:     solução ← ∅
3:     enquanto E ≠ ∅ faça
4:         (u, v) ← aresta cinza aleatória de E
5:         E ← E \ {(u, v)}
6:         C ← trilha_alternada_BFS(u, v, V, E)
7:         E ← E \ C
8:         C ← C ∪ {(u, v)}
9:         solução ← solução ∪ {C}
10:    fim enquanto
11:    devolve solução
12: fim função
```

6.3 Busca Tabu

Busca Tabu é uma estratégia para resolver problemas de otimização combinatória. É um algoritmo capaz de utilizar muitos outros métodos, como algoritmos de programação linear e heurísticas especializadas, para superar ótimos locais [7, 8].

Para descrever Busca Tabu, Glover [7, 8] utiliza o seguinte problema genérico:

dado um conjunto X de soluções factíveis, encontrar um elemento $x \in X$ que minimize uma função $c(x)$ (possivelmente não linear).

Em seguida define uma operação s, chamada **movimento**, que dado uma solução $x \in X$ gera uma nova solução $s(x) = x' \in X$. O conjunto de todos os movimentos possíveis será S. O subconjunto de movimentos que podem ser aplicados em uma solução x será chamado $S(x)$.

O Algoritmo 3 descreve a heurística *Hill Climbing* que Glover [7] mostra antes de detalhar Busca Tabu.

Algoritmo 3 Heurística Hill Climbing

1: Escolha uma solução inicial $x \in X$.
2: Escolha algum $s \in S(x)$ de tal modo que

$$c(s(x)) < c(x). \tag{19}$$

Se não existe tal s, então x é um ótimo local. Retorne x.
3: Faça $x \leftarrow s(x)$ e volte ao passo 2.

A principal limitação da heurística *Hill Climbing* é que o ótimo local encontrado pode não ser um ótimo global. A Busca Tabu continua a busca mesmo após encontrar um ótimo local, permitindo movimentos que não melhoram a função objetivo e evitando ótimos locais já visitados [7].

Para isso, um subconjunto T de S é criado, chamado de lista tabu. O conjunto T contém movimentos que foram executados em iterações recentes da heurística e que não são desejados que ocorram novamente ou que sejam desfeitos ainda.

O Algoritmo 4 descreve a heurística Busca Tabu de Glover [7].

Algoritmo 4 Heurística Busca Tabu

1: Escolha uma solução inicial $x \in X$ e faça $x^* \leftarrow x$, onde x^* é a melhor solução encontrada até o momento. Inicie o contador de iterações $k \leftarrow 0$ e a lista tabu $T = \emptyset$.
2: Se $S(x) \setminus T = \emptyset$, então vá para o passo 4. Caso contrário, faça $k \leftarrow k + 1$ e escolha $s_k(x) \in S(x) \setminus T$ de tal forma que

$$s_k(x) = \arg \min_{s \in S(x) \setminus T} \{c(s(x))\}. \tag{20}$$

3: Faça $x \leftarrow s_k(x)$. Se $c(x) < c(x^*)$, faça $x^* \leftarrow x$.
4: Se o critério de parada for satisfeito ou se $S(x) \setminus T = \emptyset$, pare o algoritmo e retorne x^*. Caso contrário, atualize T e continue do passo 2.

Explicaremos a seguir a Busca Tabu que utilizaremos neste trabalho, a qual

pode ser aplicada para o MAX-ECD.

- A **solução factível inicial** será uma solução encontrada pela heurística construtiva descrita no Algoritmo 1.
- A **avaliação do custo de uma solução** será a quantidade de circuitos encontrados.
- Para **exploração da vizinhança de uma solução** teremos duas operações:
  - A primeira operação pode ser executada caso existam dois circuitos com pelo menos três vértices em comum, digamos a, b e c (como na Figura 8, onde as linhas que conectam dois vértices são caminhos que podem conter uma ou mais arestas). Nessa operação substituímos os dois circuitos encontrados (em

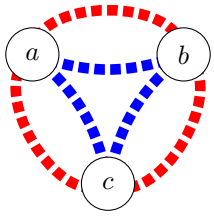


Figura 8: Dois circuitos com três vértices em comum.

azul e vermelho na Figura 8) por três circuitos que usam as mesmas arestas (em azul, vermelho e verde na Figura 9). Note que, caso existam mais do que

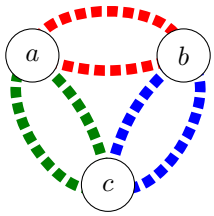


Figura 9: Três circuitos obtidos pela execução da operação.

três vértices em comum nos dois circuitos, o número de circuitos resultantes poderia ser diferente, porém, executamos a operação substituindo os dois circuitos por apenas três circuitos e, se necessário, a executamos novamente esta operação em iterações posteriores. Essa operação será executada sempre que possível, pois melhora a solução atual.

– Na segunda operação são selecionados dois circuitos que possuem dois vértices em comum. Utilizamos os 4 caminhos (2 de cada circuito) para criar dois novos circuitos que vão substituir os dois originais. Nesse passo usamos o maior caminho de cada circuito original para criar um dos circuitos e os caminhos restantes para o outro, visando formar circuitos com mais vértices (para posteriormente tentar executar a primeira operação), como na Figura 10 (linhas azuis representam um circuito e vermelhas o outro). Para essa

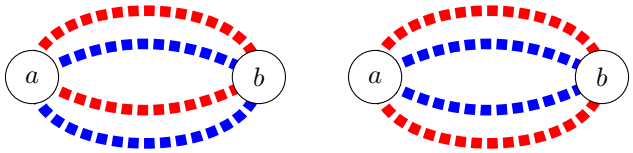


Figura 10: Exemplo de aplicação da segunda operação da Busca Tabu.

operação guardamos uma lista tabu contendo os dois vértices em comum dos circuitos. Essa operação só poderá ser executada se os vértices em comum dos circuitos não estiverem na lista tabu. A quantidade de iterações que o par de vértices ficará na lista tabu dependerá do tamanho da instância.

- Os **critérios de parada** utilizados serão um número limite de iterações e um tempo limite de execução.

6.4 Geração de Instâncias

Os experimentos para o MAX-ECD serão realizados em um conjunto de instâncias geradas aleatoriamente. Utilizaremos o algoritmo de Hakimi [10] com escolhas aleatórias dos graus dos vértices para gerar as instâncias.

Para uma dada sequência de n número inteiros não negativos, o algoritmo de Hakimi gera um grafo de n vértices cujos graus têm uma relação de um para um com os elementos da sequência ou, então, determina que tal grafo não existe. Para gerar a sequência de entrada, usamos uma distribuição de probabilidade para o grau dos vértices e sorteamos n graus, para construir um grafo com n vértices. Como estamos interessados somente em grafos eulerianos, os graus ímpares terão probabilidade zero. Observe que vértices de grau 0 não interferem na solução e podem ser desconsiderados no problema,

ou seja, damos probabilidade zero de um vértice ter grau 0.

Esse algoritmo pode gerar grafos desconexos. Nesse caso, escolhemos duas arestas aleatórias (u, v) e (x, y) de dois componentes conexos (subgrafo conexo maximal) diferentes e as substituímos pelas arestas (u, x) e (v, y), conectando esses dois componentes. Essa operação sempre diminui o número de componentes conexos, pois cada componente conexo é um grafo euleriano, portanto a troca das arestas não desconecta os componentes previamente existentes. Repetimos essa operação enquanto houver mais do que um componente conexo.

Para geração das instâncias para o problema MAX-ACD adaptamos o algoritmo da Hakimi para adicionar todas as arestas pretas e depois todas as arestas cinza.

6.5 Análise dos Resultados

Iremos comparar os resultados obtidos pelos algoritmos descritos um com os outros e com aqueles encontrados na literatura, quando possível.

Para avaliar e comparar o desempenho obtido por cada abordagem para cada instância vamos utilizar a técnica de *performance profile* criada por Dolan e Moré [5]. Utilizaremos essa técnica para analisar a performance de um conjunto de algoritmos A para um conjunto de instâncias I.

O critério que vamos utilizar para comparar os algoritmos é o valor da função objetivo obtido por eles para as instâncias do conjunto I, ou seja, a quantidade de circuitos da partição. Para cada instância $i \in I$ e cada algoritmo $a \in A$, definimos:

$$t_{i,a} = \text{o valor da função objetivo obtido pelo algoritmo } a \text{ para a instância } i. \tag{21}$$

Em seguida comparamos o resultado obtido para uma instância i por um algoritmo a com o resultado do algoritmo que obteve melhor desempenho para aquela instância. Para isto usamos o *performance ratio*

$$r_{i,a} = \frac{\max\{t_{i,b} : b \in A\}}{t_{i,a}}, \tag{22}$$

onde $\max\{t_{i,b} : b \in A\}$ é o resultado do algoritmo que obteve melhor desempenho, pois

estamos considerando um problema de maximização.

Note que $r_{i,a}$ é um valor maior ou igual a 1, que vale 1 se o algoritmo a foi o melhor para a instância i e é maior que 1 se a obteve um valor para a função objetivo menor que algum outro algoritmo para a mesma instância.

O *performance ratio* $r_{i,a}$ avalia a performance do algoritmo a apenas para a instância i. Para fazer uma avaliação do algoritmo a de maneira mais geral, considerando várias instância, definimos a função

$$\rho_a(\tau) = \frac{|\{i \in I : r_{i,a} \leq \tau\}|}{|I|}, \tag{23}$$

que será interpretada como a probabilidade do algoritmo a ter um *performance ratio* menor ou igual a τ, ou seja, a probabilidade do algoritmo a, em uma instância qualquer, obter um resultado maior ou igual ao melhor resultado obtido para essa instância dividido por τ. Chamamos a função $\rho_a(\tau)$ de *performance profile* do algoritmo a.

Após obter as funções *performance profile* de cada algoritmo vamos construir um gráfico em que o eixo das abscissas terá valores de τ, começando em 1, e o eixo das ordenadas terá os valores de $\rho_a(\tau)$ para os algoritmos $a \in A$, variando de 0 a 1.

O gráfico da Figura 11 mostra um exemplo de gráfico de *performance profile*. Nesse exemplo o Método 3 obtém os melhores resultados, conseguindo ter 100% de chance de obter soluções maiores ou iguais à melhor obtida divido por 3. Para resultados maiores ou iguais ao melhor divido por 5, por exemplo, o Método 2 tem 90% de chance de conseguir encontrar uma solução, enquanto o Método 1 tem apenas 60%.

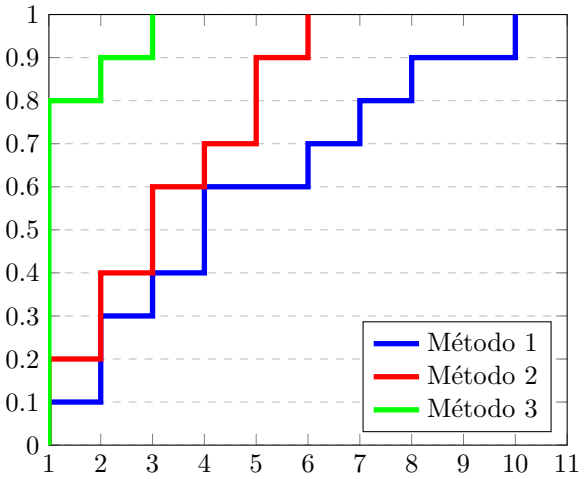


Figura 11: Exemplo de gráfico de *performance profile*.

Usaremos métodos estatísticos para analisar os dados obtidos, como o teste de Wilcoxon [17].

Os critérios que utilizaremos para comparação dos algoritmos serão os tempos de execução e as qualidades das soluções obtidas.

7 Resultados Preliminares

Nesta seção apresentaremos alguns resultados preliminares, os quais referem-se apenas ao MAX-ECD.

Executamos o modelo de programação linear inteira apresentado na Seção 6.1 utilizando o resolvedor Gurobi Optimization [9]. As heurísticas foram implementadas na linguagem de programação C++11. Todos os experimentos foram executados em uma máquina com processador Intel®Core™i7-8700T CPU com 12 núcleos de 2.40 GHz, memória RAM de 8 GB e sistema operacional Ubuntu 18.04.3.

Realizamos experimentos em um conjunto de instâncias criadas usando o método descrito na Seção 6.4. Os tamanhos das instâncias pertencem ao conjunto {10, 20, 40, 80, 160} e as distribuições de probabilidade dos graus dos vértices são:

- Distribuição 1: 50% de chance de ter grau 2 e 50% de chance de ter grau 4;
- Distribuição 2: 25% de chance de ter grau 2, 50% de chance de ter grau 4 e 25% de chance de ter grau 6;

- Distribuição 3: 25% de chance de ter grau 2, 25% de chance de ter grau 4, 25% de chance de ter grau 6 e 25% de chance de ter grau 8.

O critério de parada da Busca Tabu foi um tempo limite de execução de 10 minutos. O tamanho da lista tabu é a quantidade de vértices da instância.

Para obter resultados do modelo de Programação Linear Inteira (PLI) usamos o resolvedor Gurobi com maior prioridade em encontrar soluções factíveis do que buscar uma solução ótima. Também limitamos o tempo em 10 minutos para as execuções do Gurobi.

Para cada par (tamanho, distribuição) geramos 10 instâncias aleatórias diferentes. A Tabela 1 mostra a média do tamanho da partição que cada algoritmo obteve para cada cenário.

Em instâncias que nenhuma solução viável foi encontrada pelo resolvedor Gurobi para o modelo de programação linear inteira atribuímos valor 1 a solução. Dessa forma podemos construir o gráfico de *performance profile*.

Quantidade de Vértices	Distribuição de Probabilidade	Heurística Construtiva	Busca Tabu	PLI
10	1	3,2	3,3	3,3
	2	5,1	5,5	5,5
	3	7,1	7,2	7,4
20	1	5,5	5,8	5,9
	2	8,1	9,4	9,8
	3	12,1	13,3	13,2
40	1	7,9	9,7	8,9
	2	14,2	16,9	15,2
	3	19,9	22,6	12,0
80	1	13,7	16,4	3,6
	2	23,5	28,0	3,4
	3	34,9	40,6	3,6
160	1	22,1	26,8	1,4
	2	39,6	47,0	1,0
	3	59,7	70,8	1,0

Tabela 1: Média dos resultados por algoritmo.

Pela tabela percebemos que a Busca Tabu obteve, em média, resultados melhores do que a heurística construtiva, com grandes ganhos na função objetivo. Por exemplo, nas instâncias com 160 vértices e com a Distribuição 1 de probabilidade dos graus dos vértices houve um aumento de 21,2% na média do valor da função objetivo.

A Busca Tabu obteve resultados melhores do que a heurística construtiva em

96,2% das instâncias testadas, desconsiderando aquelas em que a heurística construtiva obteve solução ótima, provada pelo resolvedor de programação linear inteira. Nestes casos ambos os métodos obtiveram uma solução ótima. Considerando todas as instâncias, a Busca Tabu teve uma melhora média de desempenho de 16,3% em relação à heurística construtiva.

As execuções do resolvedor para o modelo de programação linear inteira só provaram que encontraram soluções ótimas em 28,7% das instâncias, entre elas todas as instâncias com 10 vértices e todas as instâncias com 20 vértices e Distribuição 1. Nas instâncias com 80 e 160 vértices foram obtidos resultados com poucos circuitos. Muitas vezes não foram encontradas soluções válidas, como nas instâncias com 160 vértices e Distribuições 2 e 3 de probabilidade dos graus dos vértices.

A Figura 12 mostra o gráfico de *performance profile* com todos os algoritmos usados. Pelo gráfico podemos dizer que o algoritmo com PLI obteve resultados piores que os outros métodos.

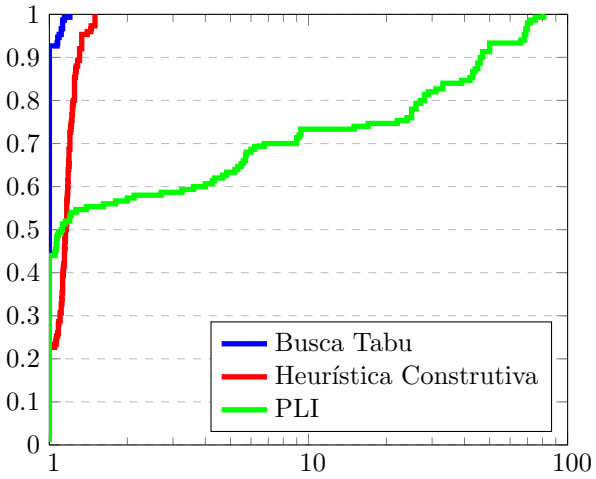


Figura 12: Gráfico de *performance profile*.

A Figura 13 mostra o mesmo gráfico da Figura 12, porém com a escala alterada para melhor comparação da heurística construtiva com a Busca Tabu. Este gráfico mostra que a Busca Tabu consegue obter com 100% de garantia resultados que são maiores ou iguais ao melhor obtido divido por 1,2, enquanto a heurística construtiva só consegue esses resultados com um pouco menos que 70% de garantia. Podemos observar por esse gráfico que o modelo de programação linear inteira obtém a melhor solução em

mais do que 40% das instâncias, porém, seu desempenho cai rapidamente.

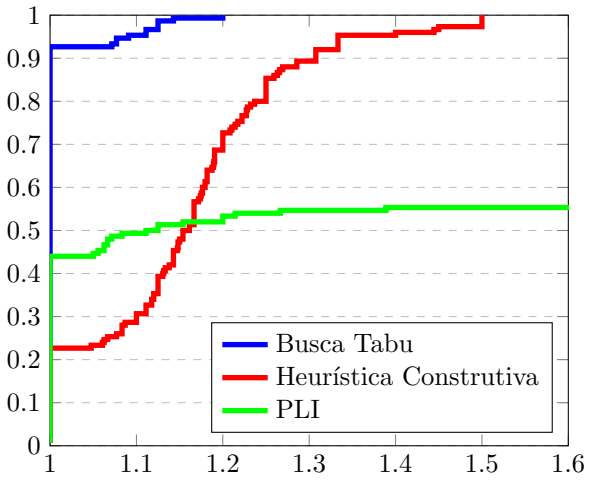


Figura 13: Gráfico de *performance profile* para soluções que são maiores ou iguais ao melhor resultado encontrado divido por 1,6.

Com base nestes resultados concluímos que o algoritmo da Busca Tabu é promissor para o MAX-ECD.

8 Plano de Trabalho

A Tabela 2 apresenta o cronograma de atividades a serem executadas ao longo do programa de mestrado. O tempo alocado e a ordem de execução de algumas atividades podem ser alterados no decorrer do desenvolvimento da pesquisa, uma vez que alguns resultados podem ser mais promissores que outros, fazendo com que algumas atividades sejam realocadas.

	2019										2020												2021	
	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2
1	•	•	•	•	•	•	•	•	•															
2		•	•	•	•	•	•	•	•	•														
3						•	•																	
4		•	•	•	•	•	•																	
5								•																
6													•	•	•	•								
7					•	•	•	•	•	•	•	•	•											
8													•	•	•	•	•	•						
9																		•	•	•	•	•		
10												•	•				•	•			•	•		
11												•	•				•	•			•	•	•	
12																								•

Tabela 2: Cronograma de atividades.

As atividades estão listadas a seguir.

1. Obtenção dos créditos obrigatórios do programa de mestrado;
2. Realização da revisão bibliográfica;
3. Geração de instâncias;
4. Escrita do Exame de Qualificação de Mestrado;
5. Realização do Exame de Qualificação de Mestrado;
6. Participação no Programa de Estágio Docente (PED);
7. Desenvolvimento de algoritmos para o MAX-ECD;
8. Desenvolvimento de algoritmos para o MAX-ACD;
9. Desenvolvimento de algoritmos para o MIN-SBR;
10. Análise dos resultados;
11. Escrita da dissertação;
12. Defesa da dissertação.

Referências

[1] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. ***SIAM Journal on Computing***, 25(2):272–289, 1996.

[2] J. A. Bondy and U. S. R. Murty. ***Graph Theory***. Graduate Texts in Mathematics. Springer, 2008.

[3] A. Caprara. On the tightness of the alternating-cycle lower bound for sorting by reversals. ***Journal of Combinatorial Optimization***, 3(2-3):149–182, 1999.

[4] A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. ***SIAM Journal on Discrete Mathematics***, 12(1):91–110, 1999.

[5] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. ***Mathematical Programming***, 91(2):201–213, 2002.

[6] S. Ganesamurthy and P. Paulraja. $2p$-cycle decompositions of some regular graphs and digraphs. ***Discrete Mathematics***, 341(8):2197–2210, 2018.

[7] F. W. Glover. Tabu Search - Part I. ***INFORMS Journal on Computing***, 1(3): 190–206, 1989.

[8] F. W. Glover. Tabu Search - Part II. ***INFORMS Journal on Computing***, 2 (1):4–32, 1990.

[9] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. `https://www.gurobi.com`, 2019. Acessado em 10/09/2019.

[10] S. L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. i. ***Journal of the Society for Industrial and Applied Mathematics***, 10(3):496–506, 1962.

[11] J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. ***Algorithmica***, 13(1/2): 180–210, 1995.

[12] G. Lancia and P. Serafini. Deriving compact extended formulations via LP-based separation techniques. ***Annals of Operations Research***, 240(1):321–350, 2016.

[13] J. Moreno, Y. Frota, and S. Martins. An exact and heuristic approach for the d-minimum branch vertices problem. ***Computational Optimization and Applications***, 71(3):829–855, 2018.

[14] J. Moreno, S. Martins, and Y. Frota. A note on the rainbow cycle cover problem. ***Networks***, 73(1):38–47, 2019.

[15] C. M. Mynhardt and C. M. van Bommel. Triangle decompositions of planar graphs. ***Discussiones Mathematicae Graph Theory***, 36(3):643–659, 2016.

[16] C. A. Rodger. Graph decomposition. ***Le Matematiche***, 45(1):119–140, 1990.

[17] F. Wilcoxon. Individual comparisons by ranking methods. In ***Breakthroughs in statistics***, pages 196–202. Springer, 1992.