Notas em Matemática Aplicada

ISSN 2175-3385
E-ISSN 2236-5915

Volume 100, 2024

Corpo Editorial

Sandra Mara Cardoso Malta (Editor Chefe)
Laboratório Nacional de Computação Científica - LNCC
Petrópolis, RJ, Brasil

Eduardo V. O. Teixeira (Editor Executivo)
University of Central Florida - UCF
Orlando, FL, EUA

Lilian Markenzon
Universidade Federal do Rio de Janeiro - UFRJ
Rio de Janeiro, RJ, Brasil

Marcelo Sobottka
Universidade Federal de Santa Catarina - UFSC
Florianópplis, SC, Brasil

Paulo F. de Arruda Mancera
Universidade Estadual Paulista Júlio de Mesquita Filho- UNESP
Botucatu, SP, Brasil

Sandra Augusta Santos
Universidade Estadual de Campinas - UNICAMP
Campinas, SP, Brasil

Tânia Schmitt
Universidade de Brasília - UnB
Brasília, DF, Brasil

Sociedade Brasileira de Matemática Aplicada e Computacional

2024

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC em 2003, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa a publicar, também, livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex, com as figuras em .eps, .pdf e etc.** e ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo. O idioma pode ser Português ou Espanhol.

Veja todos os títulos publicados nesta série na página
http://https://proceedings.science/notas-sbmac

Sociedade Brasileira de Matemática Aplicada e Computacional

2024

REDUÇÃO DE LARGURA DE BANDA DE MATRIZES

Sanderson L. Gonzaga de Oliveira
sanderson.oliveira@unifesp.br

Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2024

Coordenação Editorial da Série: Sandra M. C. Malta

Editora: SBMAC

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Copyright ©2024 por *Sanderson L. Gonzaga de Oliveira*. Direitos reservados, pela SBMAC. A publicação nesta série não impede o autor de publicar parte ou a totalidade da obra por outra editora, em qualquer meio, desde que faça citação à edição original.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Oliveira, Sanderson L. Gonzaga de
Redução de largura de banda de matrizes [livro eletrônico] / Sanderson L. Gonzaga de Oliveira. -- São Carlos : SBMAC, 2024. -- (Notas em matemática aplicada ; 100)
PDF

Bibliografia.
ISBN 978-65-86388-31-2

1. Matemática - Estudo e ensino 2. Matrizes (Matemática) I. Título. II. Série.

24-205457 CDD-510.7

Índices para catálogo sistemático:

1. Matemática : Estudo e ensino 510.7

Tábata Alves da Silva - Bibliotecária - CRB-8/9253

Para Luísa, Tatiana,
e à memória de meus pais, Selma e Luiz,
dedico.

Agradecimentos

Agradeço ao comitê responsável pela série Notas em Matemática Aplicada da SBMAC a oportunidade de divulgação deste trabalho. Agradeço especialmente à professora Sandra Malta (LNCC) a dedicação e esforço como Editora Chefe da série. Agradeço a Guilherme O. Chagas por gentilmente ceder figuras do nosso livro em conjunto (volume 75 da série) para melhorar os exemplos mostrados neste texto. Agradeço a Alexandre A. A. M. de Abreu a criação de figura para este livro, bem como a leitura e comentários de partes do texto. Agradeço também aos revisores anônimos as valiosas sugestões para que o texto fosse melhorado.

Conteúdo

Prefácio

A principal aplicação do problema de redução de largura de banda de matrizes está intrinsecamente relacionado ao problema de numerar pontos de malhas computacionais para a solução de equações diferenciais parciais por métodos de discretização, como o método dos elementos finitos. Por causa da importância do tópico, possivelmente, centenas de métodos heurísticos para redução de largura de banda foram propostos desde a década de 1960.

Em 2014, foi publicado o volume 75 da série Notas em Matemática Aplicada da SBMAC, intitulado Introdução a heurísticas para redução de largura de banda de matrizes. Diversos métodos heurísticos foram apresentados no volume 75 da série como introdução ao tema. Nesses 10 anos, revisões foram realizadas mostrando que muitos daqueles algoritmos foram superados por outros métodos.

Assim, no presente texto, com o amadurecimento acadêmico do autor no tema, alguns dos principais métodos heurísticos para redução de largura de banda de matrizes são apresentados neste livro. São apresentados 12 métodos heurísticos para redução de largura de banda. Seis desses métodos são baseados em conceitos de teoria dos grafos: Cuthill-Mckee (1969), RCM (1971), GPS (1976), KP-band (2011), RBFS-GL (2018), RLK (2020). Também são apresentados cinco algoritmos meta-heurísticos: NCHC (2006), FNCHC (2007), VNS-band (2010), DRSA (2015) e FNCHC+ (2022). Ainda, é apresentada uma hiper-heurística: ACHH (2020).

O livro está dividido em quatro capítulos. No capítulo 1, há uma introdução ao tópico e noções básicas são apresentadas para a compreensão dos demais capítulos, como conceitos elementares em teoria dos grafos.

Métodos heurísticos importantes, baseados em teoria dos grafos, são apresentados no capítulo 2. Incluindo a busca em largura, são apresentados sete (CM, RCM, RBFS-GL, KP-band, RLK, GPS) métodos heurísticos baseados em conceitos da teoria dos grafos no capítulo 2. Esses são métodos rápidos e práticos para serem utilizados no pré-processamento de matrizes para aceleração de resolutores de sistemas de equações lineares.

No capítulo 3, é mostrada uma hiper-heurística baseada na meta-heurística otimização por colônia de formigas. Essa hiper-heurística evolui três métodos heurísticos baseados em conceitos da teoria dos grafos (RCM, KP-band e RLK). Além desses três métodos, a hiper-heurística também pode selecionar a heurística RBFS-GL. Assim, a hiper-heurística seleciona ou gera métodos heurísticos baseados em conceitos da teoria dos grafos. Esses métodos heurísticos resultantes da hiper-heurística são tão rápidos quanto os métodos rápidos utilizados como base da evolução. Consequentemente, os métodos heurísticos resultantes da hiper-heurística forneceram resultados melhores no pré-processamento de matrizes para aceleração de resolutores de sistemas de equações lineares em comparação com os métodos anteriores.

Algoritmos meta-heurísticos, que reduzem bastante a largura de banda, são apresentados no capítulo 4. Por serem projetados por meta-heurísticas, inerentemente, são mais lentos que, por exemplo, os métodos heurísticos baseados na busca em largura, apresentados nos capítulos 2 e 3. No capítulo 4, são apresentados cinco algoritmos meta-heurísticos, baseados em três meta-heurísticas diferentes. Com isso, são abordadas quatro meta-heurísticas neste texto.

Em suma, uma coleção de métodos heurísticos projetados por técnicas variadas é apresentada. Supõe-se que o leitor poderá obter uma introdução adequada ao assunto com os métodos heurísticos apresentados.

Claramente, o presente texto não esgota o assunto. Este livro é apenas uma introdução a métodos heurísticos para redução de largura de banda de matrizes. Desde já, agradecemos as sugestões para a melhoria do texto que venham a ser encaminhadas para o autor.

São José dos Campos, abril de 2024.

Sanderson L. Gonzaga de Oliveira

Capítulo 1

Introdução à redução de largura de banda de matrizes

1.1 Introdução

O problema de minimização de largura de banda de uma matriz simétrica A é aproximar, ao máximo possível, todos os coeficientes não nulos[1] da diagonal principal de A. Isso significa que minimizar a largura de banda é encontrar a menor largura de banda de uma matriz simétrica.

O problema de minimização de largura de banda de matrizes surgiu, possivelmente, na década de 1950. Engenheiros de estruturas analisavam a resolução computacional de sistemas de equações lineares com matrizes de grande porte e já tinham a preocupação com a largura de banda das matrizes (por exemplo, veja o artigo de Livesley [89]).

O problema de minimização de largura de banda de matrizes pertence à classe $\mathcal{NP}$-difícil, conforme foi demonstrado por Papadimitriou [98]. O autor mostrou que o problema de determinar se os vértices de um grafo têm uma ordenação que produza uma largura de banda de tamanho B ou menor é $\mathcal{NP}$-Completo. Garey et al. [34] provaram que o problema pertence à classe $\mathcal{NP}$-completo mesmo para árvores com grau máximo 3. Petit [99] revisou resultados da resolução de vários problemas em grafos: em minimização de largura de banda, pesquisadores identificaram muitas classes particulares de grafos em que esse problema permanece pertencente à classe $\mathcal{NP}$-Difícil.

Alguns conceitos elementares são importantes para o acompanhamento do texto. Basicamente, algoritmo é uma sequência finita de instruções rigorosas para resolver um problema computacional. Neste contexto, considere método como sinônimo de algoritmo e um método heurístico, ou simplesmente heurística, é um algoritmo projetado para

[1] Tipicamente, nulos são representados por zeros.

retornar, com baixo custo de execução, uma solução aproximada à solução ótima de um problema de otimização ao ser aplicado a instâncias de grande porte do problema. Nesses problemas, quando pertencentes à classe $\mathcal{NP}$-Difícil, um algoritmo para encontrar solução ótima do problema teria um tempo de execução impeditivo para instâncias suficiente grandes. Em um problema de otimização, busca-se maximizar ou minimizar uma função objetivo. Basicamente, instância é um caso específico do problema. Neste contexto, matriz é uma instância para o problema de redução de largura de banda.

Os métodos que realizam a redução de largura de banda podem ser divididos em métodos exatos, algoritmos aproximativos, métodos heurísticos e métodos híbridos. Na seção 1.2, métodos exatos, algoritmos aproximativos e métodos híbridos são abordados brevemente.

Neste texto, o foco é em métodos heurísticos seriais para redução de largura de banda de matrizes de grande porte. Para métodos heurísticos para a renumeração de matrizes retangulares, veja a seção 7.13 da obra de Kaveh [75, p. 260-264]. Para sub-ordenações de blocos de vértices, veja a seção 7.14 do livro de Kaveh [75, p. 265-267]. Isso pode ser útil, por exemplo, para particionamento de grafos em resoluções paralelas de sistemas de equações lineares.

Como é claramente impraticável verificar todas as $n!$ sequências possíveis associadas com uma matriz de ordem n suficiente grande, desde a década de 1960, foram propostos, provavelmente, centenas de métodos heurísticos para o problema de redução de largura de banda de matrizes. Everstine [28], em 1979, referenciou 49 métodos heurísticos para reordenação de linhas e colunas de matrizes neste contexto. Gibbs [39], em 1980, afirmou que a lista de Everstine [28] não foi exaustiva. Gibbs [39] supôs que existiriam de 50 a 100 métodos heurísticos para reordenação de linhas e colunas de matrizes neste contexto até aquele ano. Com a grande quantidade de métodos heurísticos para redução de largura de banda, têm-se evidências da importância desse problema [57, 9].

Do ponto de vista da teoria dos grafos, pode-se considerar uma permutação de linhas e colunas em uma matriz simétrica como equivalente a se renumerar os vértices do grafo correspondente sem que as adjacências sejam alteradas. Como exemplo, na figura 1.1, tem-se a representação de um grafo por meio de uma matriz de adjacências. Cada linha da matriz A corresponde a um vértice no grafo da figura 1.1, ou seja, para $1 \leq i \leq 10$, a linha i da matriz corresponde ao vértice i do grafo.

Para o grafo mostrado na figura 1.2, utilizou-se uma ordem diferente da anterior para numerar os vértices do grafo e obteve-se uma representação matricial também diferente. Nota-se que a disposição

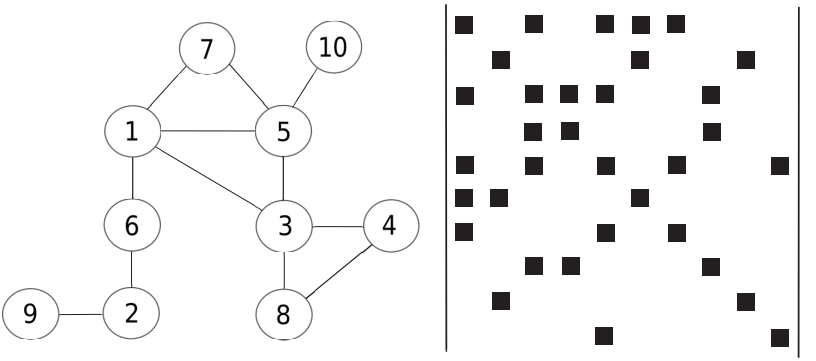

Figura 1.1: Representação de grafo por uma matriz de adjacências [56]. A diagonal principal (veja a definição 1.6, na página 6) da matriz é representada por clareza.

dos coeficientes não nulos na representação matricial do grafo depende da ordem em que os vértices do grafo são numerados. É da ordem dessa numeração de que trata este texto. Para isso, há métodos heurísticos que definem essa ordem e o vértice inicial da numeração [56].

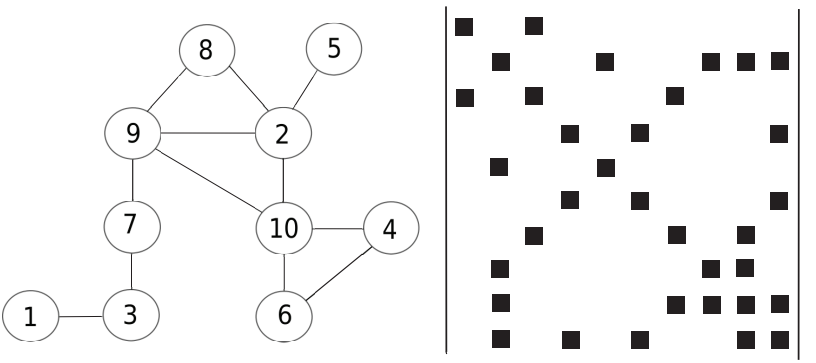

Figura 1.2: Grafo mostrado na figura 1.1 com a numeração dos vértices alterada [56].

Este texto está organizado da seguinte forma. Os principais conceitos para o entendimento deste texto são apresentados na seção 1.3. Na seção 1.4, são comentados os métodos heurísticos apresentados ao longo deste texto. Na seção 1.5, comenta-se sobre armazenamento de matrizes.

Métodos heurísticos rápidos para reduções de largura de banda de matrizes são apresentados no capítulo 2. Esses métodos heurísticos são

baseados em níveis de vértices a partir de um vértice inicial, do grafo subjacente da matriz A.

Uma hiper-heurística baseada na meta-heurística otimização por colônia de formigas é apresentada no capítulo 3. Algoritmos meta-heurísticos, ou seja, métodos heurísticos baseados em meta-heurísticas, são apresentados no capítulo 4.

Os pseudocódigos estão mostrados de uma forma a se manter a simplicidade na apresentação. Claramente, esquemas e estruturas sofisticadas podem ser utilizadas em uma fase de implementação.

1.2 Algoritmos exatos, aproximativos e híbridos

Métodos por força bruta encontram a menor largura de banda de matrizes por *backtracking* no conjunto de $n!$ permutações possíveis de vértices [108, p. 399]. Exemplo de método exato pode ser estudado no trabalho de Harary [70].

Sudborough e colaboradores [95, 66] propuseram algoritmos por programação dinâmica com complexidade $\mathcal{O}(n^b)$ em tempo e espaço, com $1 \leq b \leq n$. Esses métodos são impraticáveis se b não for pequeno.

Del Corso e Manzini [21] propuseram dois algoritmos exatos baseados em busca em profundidade. O primeiro algoritmo, chamado de MB-ID (*Minimum Bandwidth by Iterative Deepening*) tenta encontrar uma rotulação dos vértices de largura de banda $\beta = \left\lceil \frac{k}{2} \right\rceil$, em que k é o grau máximo do grafo. Caso essa ordenação não possa ser encontrada, uma ordenação de largura de banda $\beta + 1$ é procurada, sucessivamente, até que se encontre a solução ótima. O segundo algoritmo, denominado MB-PS (*Minimum Bandwidth by Perimeter Search*), é uma variação do primeiro algoritmo que implementa uma busca em profundidade conhecida como busca em perímetro. Um conjunto de rotulações parciais, chamado de perímetro, é gerado, e obtém-se uma solução final combinando duas rotulações parciais. Os autores indicaram que a técnica pode reduzir o custo computacional em até 100 vezes, se comparada com a busca em profundidade simples; entretanto, o consumo de memória pode ser alto para perímetros grandes. Os autores realizaram seus experimentos com instâncias geradas aleatoriamente de até 100 vértices.

Feige e Kilian [29] apresentaram um algoritmo com complexidade $\mathcal{O}^*(10^n)$ e com ocupação de espaço polinomial. $\mathcal{O}^*$ é a notação $\mathcal{O}$ padrão omitindo fatores polinomiais.

Caprara e Salazar-González [8] desenvolveram e melhoraram métodos baseados em programação inteira. Os autores apresentaram um novo limitante inferior para a largura de banda que pode ser encontrado em tempo $\mathcal{O}(|V||E|)$ em um grafo $G(V, E)$. O limitante é

$\gamma(G) = \min_{v \in V} \left[\max_{C \subseteq V, v \in C} \left[\left\lceil \frac{|C|-1}{d(v,C)} \right\rceil \right] \right]$, em que $d(u, C)$ é a distância entre u e o subconjunto C. Esse limitante foi utilizado para construir um algoritmo que encontra *layouts* parciais de largura de banda mínima e estima a menor largura de banda que pode ser encontrada no grafo. Os autores mostraram que o método proposto obteve sucesso em quase todas as instâncias utilizadas de até 200 vértices. Por outro lado, a diferença entre os limitantes inferior e superior foi maior do que 20% em instâncias com menos de 150 vértices.

Algoritmos híbridos utilizam métodos exatos com algoritmos metaheurísticos. Martí e colaboradores [90] propuseram uma solução híbrida que utiliza a metaheurística GRASP [100] para obter um limitante superior inicial β_{up}. Após a utilização do método construtivo, aplica-se um método *branch and boud* baseado na proposta de Caprara e Salazar-González [8] com intuito de encontrar uma solução $\beta \leq \beta_{up} - 1$. Caso essa rotulação não possa ser encontrada, o algoritmo encerra e retorna β_{up}. Se uma solução $\beta \leq \beta_{up} - 1$ é encontrada, atualiza-se $\beta = \beta_{up} = \beta_{up} - 1$ e o processo se repete. Os autores reportaram resultados favoráveis em comparação aos resultados obtidos pelos métodos MB-ID e MB-PS propostos por Caprara e Salazar-González [8].

Um algoritmo de complexidade $\mathcal{O}^*(5^n)$ para o tempo de execução e complexidade $\mathcal{O}^*(2^n)$ para o custo de ocupação de memória foi apresentado por Cygan e Pilipczuk [15]. Inicialmente, o algoritmo divide os vértices do grafo em $\left\lceil \frac{|V|}{b+1} \right\rceil$ segmentos de tamanho $b+1$, com exceção do último segmento que terá $|V| \cdot mod(b+1)$ vértices. Na sequência, uma busca em profundidade é realizada para tentar encontrar uma ordenação com largura de banda $\beta \leq b$. Após a verificação de cada segmento, a solução final é encontrada. Uma melhoria deste método foi apresentada por Cygan e Pilipczuk [18] e apresenta complexidade $\mathcal{O}(4,83^n)$ para o tempo de execução e complexidade $\mathcal{O}^*(4^n)$ para o custo de ocupação de memória. Cygan e Pilipczuk [16] propuseram um algoritmo com complexidade de tempo e espaço $\mathcal{O}(4,383^n)$. Cygan e Pilipczuk [17] propuseram um algoritmo com complexidade $\mathcal{O}(9,363^n)$ com ocupação de espaço polinomial.

Basicamente, autores de algoritmos aproximativos analisam a distância que a solução aproximada do algoritmo está da solução ótima. Em seguida, mostram a análise de complexidade do algoritmo. Para introdução a esse assunto, veja, por exemplo, a publicação de Fürer e colaboradores [33].

1.3 Conceitos básicos

É mostrada uma introdução a conceitos elementares de matrizes na subseção 1.3.1. É mostrada uma introdução a conceitos básicos de teoria dos grafos na subseção 1.3.2.

1.3.1 Conceitos elementares sobre matrizes

Neste texto, os conceitos sobre matrizes são utilizados conforme as definições dadas a seguir.

Definição 1.1 (matriz)**.** *Uma matriz $A = [a_{ij}]$ é um conjunto, de forma que seus elementos são números ou coeficientes. As entradas a_{ij} da matriz são organizadas em linhas e colunas, em que $1 \leq i, j \leq n$ e n é um inteiro positivo.*

Uma matriz representa um objeto matemático. Por exemplo, a matriz mostrada na figura 1.1 representa o grafo mostrado na mesma figura.

Definição 1.2 (matriz quadrada)**.** *Uma matriz quadrada tem o mesmo número de linhas e colunas.*

Neste texto, são utilizadas somente matrizes quadradas $n \times n$, em que n é um inteiro positivo maior que 1.

Definição 1.3 (matriz transposta)**.** *Sejam duas matrizes $A = [a_{ij}]$ e $A^{\mathrm{T}} = [a_{ji}^{\mathrm{T}}]$, em que $1 \leq i, j \leq n$ e n é um inteiro positivo. A matriz A^{T} é a matriz transposta da matriz A se $a_{ij} = a_{ji}^{\mathrm{T}}$.*

Definição 1.4 (matriz simétrica)**.** *Uma matriz A simétrica é um matriz quadrada em que $A = A^{\mathrm{T}}$.*

Por exemplo, a matriz mostrada na figura 1.1 é quadrada e simétrica.

Definição 1.5 (matriz positiva definida)**.** *Uma matriz simétrica A é positiva definida se, para qualquer vetor $x \in \mathbb{R}^n$ não nulo, tem-se, $x^{\mathrm{T}}Ax > 0$. A matriz A é positiva definida se e, somente se, todos os seus autovalores são positivos.*

Para a definição de autovalores, seria necessária a definição de outros conceitos de Álgebra Linear e está fora do escopo deste texto. Detalhes sobre essas definições e outras, como autovalores, podem ser consultados em livros de Álgebra Linear; a obra de Boldrini et al. [4] é uma ótima opção.

Definição 1.6 (diagonal principal)**.** *Em uma matriz $A = [a_{ij}]$ quadrada, em que $1 \leq i, j \leq n$, a diagonal principal é composta pelos termos a_{ii}.*

Definição 1.7 (matriz não singular)**.** *Uma matrix $A \in \mathbb{R}^{n\times n}$ é não singular (invertível ou não degenerada) se existe uma matriz $B \in \mathbb{R}^{n,n}$, tal que $AB = BA = I_n$. A matrix B é denotada por A^{-1} e é chamada de matriz inversa de A.*

Os sistemas de equações lineares considerados neste texto são na forma $\sum_{j=1}^{n} a_{ij}x_j = b_i$, para $1 \leq i \leq n$, ou em notação matricial, $Ax = b$, para $A \in \mathbb{R}^{n\times n}$, em que $A = [a_{ij}]$ é uma matriz esparsa, simétrica, não singular e de grande porte, $x \in \mathbb{R}^n$ é o vetor de incógnitas, $b \in \mathbb{R}^n$ é um vetor de termos independentes, que pode ser esparso ou denso, e n é a ordem ou dimensão da matriz A. Em uma matriz esparsa, a quantidade de entradas nulas nessa estrutura matricial é significativa a ponto de ser conveniente utilizar uma estrutura de dados que não armazene valores nulos [43]. No grafo não direcionado $G = (V, E)$ subjacente à matriz simétrica A, considere que os vértices $u, v \in V$ correspondem aos índices i, j, respectivamente. Se $\{u, v\} \notin E \implies a_{ij} = \emptyset$, ou seja, se os vértices $u, v \in E$ não são adjacentes, então, a entrada a_{ij} é zero (ou nula).

Considere uma estrutura de dados matricial A bidimensional convencional com $n \times n$ entradas a_{ij}, para $1 \leq i, j \leq n$, em que i e j são os índices das linhas e das colunas e correspondem aos vértices $v, u \in V$ do grafo $G = (V, E)$ subjacente, respectivamente. Nessa matriz de adjacências (veja a definição 1.24), as entradas vizinhas da entrada a_{ij} são acessadas simplesmente ao se incrementar ou decrementar os índices i e j. A principal vantagem dessa estrutura é que o acesso para verificar se $\{v, u\} \in E$ é realizado em $\mathcal{O}(1)$. Sua principal desvantagem é ter $\Theta(n^2)$ como ocupação de memória. Essa ocupação de memória pode ser impeditiva em simulações com instâncias de grande porte.

1.3.2 Conceitos de teoria dos grafos e matrizes

Os 13 métodos apresentados neste texto foram projetados sobre grafos. Neste texto, os conceitos sobre teoria dos grafos são utilizados conforme as definições dadas a seguir. Outros conceitos sobre matrizes também são apresentados no decorrer do texto.

Definição 1.8 (vértice)**.** *Um vértice é um objeto simples que possui rótulo (ou identificador) e pode possuir nome e outros atributos.*

Definição 1.9 (aresta)**.** *Uma aresta (ou arco) é uma conexão direta entre dois vértices.*

Arestas são formadas por um par de vértices em suas extremidades. Dessa forma, vértices são também chamados de extremos das arestas. Graficamente, as arestas são representadas por segmentos de retas ou

curvas e podem ou não ter direção. Se as arestas tiverem peso, tem-se um grafo ponderado.

Um grafo $G = (V, E)$ é um par de conjuntos V e E, em que V, com $|V| = n$, é um conjunto de objetos denominados *vértices* e $E = \{(v, u) : v, u \in V\}$ ($E = \{\{v, u\} : v, u \in V\}$) é um conjunto de pares (não) ordenados de vértices de V, se o grafo é (não) direcionado. Assim, $E \subseteq V \times V$. Neste texto, V e E são considerados conjuntos finitos e o conjunto V é considerado não vazio. Para evitar ambiguidades notacionais, considera-se neste texto que $V \cap E = \varnothing$.

Se as arestas não têm direção, tem-se um grafo não direcionado. Neste caso, as arestas são representadas por pares não ordenados de vértices como $\{v, u\} \in E$, indicando que os vértices $v, u \in V$ são adjacentes. Adjacência é explicada na definição 1.10. O grafo mostrado na figura 1.1, com 10 vértices e 12 arestas, não é ponderado e não é direcionado.

Se as arestas têm direção, são representadas graficamente com setas e tem-se um *grafo direcionado* (ou *digrafo*). Neste caso, as arestas são representadas por pares ordenados de vértices como $(v, u) \in E$, indicando que o vértice u é adjacente ao vértice v, mas o inverso não é verdadeiro.

No contexto de métodos numéricos, vértices são chamados de pontos de malhas computacionais em que os métodos de discretização são aplicados. Nesse sentido, basicamente, uma malha é um grafo em que é relevante a geometria entre os vértices e arestas [45].

Adiante, considere um grafo $G = (V, E)$ não direcionado e não ponderado, constituído por um conjunto finito de vértices $V = \{v_1, v_2, v_3, \ldots, v_n\}$ e um conjunto finito de arestas $E = \{e_1, e_2, e_3, \ldots, e_m\}$ representadas por pares não ordenados de vértices $\{v_i, v_j\}$. Têm-se as definições mostradas a seguir [46].

Definição 1.10 (adjacência)**.** *Sejam um grafo não direcionado $G = (V, E)$ e $v, u \in V$. O vértice u é adjacente (ou vizinho) a v se e, somente se, $\{v, u\} \in E$. Assim, $Adj(G, v) = \{u \in V : \{v, u\} \in E\}$ é o conjunto de vértices adjacentes ao vértice v.*

Definição 1.11 (incidência)**.** *Sejam um grafo não direcionado $G = (V, E)$ e $v, u \in V$. A aresta $\{v, u\} \in E$ é incidente aos vértices $v, u \in V$.*

Definição 1.12 (grau)**.** *O grau de um vértice v é o número de vértices adjacentes ao vértice v, ou seja, $Grau(G, v) = |Adj(G, v)| = |\{u \in V : \{v, u\} \in E\}|$.*

Por exemplo, os vértices com rótulos 6 e 9 são adjacentes ao vértice com rótulo 2 no grafo mostrado na figura 1.1. Portanto, o grau do vértice com rótulo 2 é 2. Dependendo do contexto, como em geometria computacional, grau é chamado de valência.

Definição 1.13 (caminho)**.** *Caminho é um grafo não vazio na forma de uma a sequência de vértices tal que, a partir de um vértice inicial, de cada um dos vértices parte uma aresta para o vértice seguinte, até o último vértice da sequência.*

Por exemplo, $V = \{x_0, x_1, \ldots, x_k\}$ e $E = \{\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}\}$ formam um caminho que conecta os vértices x_0 e x_k. Em particular, k pode ser zero. Um caminho é *simples* (ou elementar) se nenhum vértice é repetido.

Definição 1.14 (tamanho do caminho)**.** *O tamanho (ou comprimento) de um caminho é o seu número de arestas.*

Definição 1.15 (ciclo)**.** *Um ciclo é um caminho fechado na forma* $\{x_0, x_1, \ldots, x_{k-1}, x_0\}$ *e* $E = \{\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_0\}\}$*, em que cada um dos* x_i *são distintos para* $0 \leq i \leq k-1$ *e o primeiro vértice do caminho também é o último.*

Definição 1.16 (tamanho do ciclo)**.** *O tamanho (ou comprimento) de um ciclo é o seu número de arestas (ou vértices).*

Definição 1.17 (grafo conexo)**.** *Um grafo não vazio é conexo se existe um caminho entre quaisquer pares de vértices.*

Por exemplo, o grafo mostrado na figura 1.1 é simples e conexo.

Definição 1.18 (grafo acíclico)**.** *Um grafo sem ciclos é acíclico.*

Definição 1.19 (árvore)**.** *Árvore é um grafo conexo acíclico.*

Portanto, árvore é um caso especial de grafos. No contexto de árvores, vértices são comumente chamados de nodos ou nós. Nesse sentido, arestas de árvores são chamadas comumente de ramos.

Basicamente, árvore é uma estrutura hierárquica. Em uma árvore, há um nodo singular chamado de *raiz*. Uma árvore computacional tem uma só *raiz* e é, geralmente, representada no topo da estrutura, como um organograma de uma empresa. Os nodos *filhos* f_i da raiz são representados embaixo da raiz. Os nodos filhos f_{i+1} dos nodos f_i são representados embaixo dos nodos f_i e assim por diante. Se há um ramo entre o nodo f_i e o nodo f_{i+1}, então, o nodo f_i é chamado de *pai* do nodo f_{i+1}. Os nodos que não têm filhos são chamados de *folhas* ou *nodos terminais*. Apesar de ser comum essa representação hierárquica, com a raiz no topo, isso não é necessário, pois uma árvore é um grafo acíclico, como mostrado na figura 1.3.

Definição 1.20 (nodo folha de árvore)**.** *Um nodo de grau 1 de uma árvore é uma* folha*, exceto o nodo raiz.*

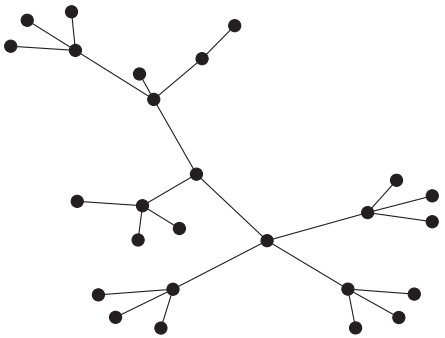

Figura 1.3: Uma árvore.

Nesse sentido, em uma árvore, nodos não folhas são nodos internos e folhas são nodos externos.

Definição 1.21 (subgrafo)**.** *Seja um grafo $G = (V, E)$. Um grafo $G' = (V', E')$ é um subgrafo do grafo $G = (V, E)$ se $V' \subseteq V \wedge E' \subseteq E$.*

Sejam $G = (V, E)$ e $G' = (V', E')$. Se $V' \subseteq V \wedge E' \subseteq E$, então, G' é um subgrafo de G, que é um supergrafo de G', escrito $G' \subseteq G$.

Definição 1.22 (componente de um grafo)**.** *Componente de um grafo $G = (V, E)$ é um subgrafo conexo maximal de $G = (V, E)$, ou seja, é o subgrafo conexo que não está estritamente contido em outro subgrafo conexo.*

De forma geral, ao se referir a um grafo maximal (ou minimal) em relação a alguma propriedade, mas não se especifica nenhuma ordenação particular, a referência é em relação ao subgrafo.

Definição 1.23 (numeração bijetora dos vértices do grafo)**.** *Uma numeração $S : V \rightarrow \{1, 2, \ldots, |V|\}$ dos vértices do grafo não direcionado $G = (V, E)$ retorna, para cada vértice, um número do conjunto $\{1, 2, \ldots, |V|\}$. Assim, $s(v)$ é o rótulo do vértice $v \in V$ na numeração $S = \{s(v_1), s(v_2), \ldots, s(v_{|V|})\}$. Com isso, identificadores no intervalo $[1, |V|]$ são utilizados para rotular os vértices na numeração S dos vértices $V = \{v_1, v_2, \ldots, v_{|V|}\}$. Portanto, S é uma bijeção de V para o conjunto $\{1, 2, \ldots, |V|\}$.*

Como mostrado nas figuras 1.1 e 1.2, neste texto, os rótulos dos vértices do grafo são utilizados para identificá-los. Para manter a consistência no uso de funções que têm como argumento um vértice, será

utilizada a função inversa $s^{-1}(i)$, para $i \in [1, |V|]$, da função $s(v)$, para $v \in V$, em tais casos, isto é, $S^{-1} : \{1, 2, \ldots, |V|\} \to V$. Assim, $s^{-1}(i)$ retorna o vértice $v \in V$ com rótulo i na numeração S dos vértices em V.

Definição 1.24 (matriz de adjacências)**.** *Sejam um grafo conexo e não direcionado $G_S = (V, E)$ com numeração S e $n = |V|$. Uma matriz de adjacências $A_S = [a_{ij}]$, com $1 \leq i, j \leq n$, é uma matriz quadrada $n \times n$ utilizada para representar o grafo $G_S = (V, E)$. A entrada a_{ij} da matriz de adjacências é utilizada para indicar se pares de vértices em V com rótulos i e j são adjacentes. Assim, a matriz A_S é a matriz obtida pela numeração S do grafo $G_S = (V, E)$ subjacente.*

A matriz de adjacências A_S do grafo $G_S = (V, E)$ subjacente é simétrica se e, somente se, o grafo G_S não é direcionado. Por exemplo, na figura 1.1, é mostrada a matriz de adjacências do grafo mostrado na mesma figura.

Definição 1.25 (largura de banda da linha na matriz simétrica)**.** *Seja $A_S \in \mathbb{R}^{n \times n}$ uma matriz simétrica, em que $A = [a_{ij}]$, i.e., com entradas a_{ij}, para $1 \leq i, j \leq n$. A largura de banda da i-ésima linha é $\beta_i(A_S) = i - \min_{1 \leq j < i}[j \mid a_{ij} \neq 0] = \max_{1 \leq i \leq n, 1 \leq j < i}[i - j \mid a_{ij} \neq 0]$.*

A definição 1.25 significa que a largura de banda $\beta_i(A_S)$, da linha i da matriz A_S, é o número de entradas, a partir de coeficiente não nulo mais à esquerda na linha i, até a diagonal principal. Portanto, a definição 1.25 também significa que $a_{ij} = \emptyset$ (ou seja, nulo) se $|i - j| > \beta_i(A_S)$. Por exemplo, na matriz A com numeração S mostrada na figura 1.1, $\beta_1(A_S) = \beta_2(A_S) = 0$, $\beta_3(A_S) = 2$, $\beta_4(A_S) = 1$, $\beta_5(A_S) = 4$, $\beta_6(A_S) = \beta_8(A_S) = \beta_{10}(A_S) = 5$, $\beta_7(A_S) = 6$, $\beta_9(A_S) = 7$.

O valor 0 (zero) pode ser válido em alguns problemas, como em problemas em meios porosos em dinânica de fluidos computacional. Assim, neste texto, a entrada a_{ij} da matriz A também é descrita teoricamente como nula. Um valor nulo é representado por $\emptyset$ neste texto. Dessa forma, descreve-se $a_{ij} = \emptyset$ se $\{v, u\} \notin E$, no grafo subjacente $G_S = (V, E)$, em que i e j são as linhas da matriz A_S correspondentes aos vértices $v, u \in V$, respectivamente. Isso também pode estar consistente com uma fase de implementação. A razão é que dificilmente é utilizada matriz de adjacências para armazenar uma matriz esparsa de grande porte justamente por causa da ineficiência em armazenar grande quantidade de valores que são nulos no problema original. Uma matriz de adjacências ocupa $\Theta(|V|^2)$ posições de memória e pode ser impraticável em problemas de grande porte. Há formatos que demandam ocupação linear no número de vértices e arestas do grafo subjacente à matriz esparsa. Isso é comentado com brevidade na seção 2.2.2.

Definição 1.26 (largura de banda na matriz simétrica)**.** *Seja* $A_S \in \mathbb{R}^{n\times n}$ *uma matriz simétrica, em que* $A = [a_{ij}]$*, i.e., com entradas* a_{ij}*, para* $1 \leq i, j \leq n$*. A largura de banda da matriz* A_S *é* $\beta(A_S) = \max_{1\leq i\leq n}[\beta_i(A_S)] = (1 \leq i \leq n)\ i - \min_{1\leq j<i}[j \mid a_{ij} \neq 0] = \max_{1\leq i\leq n, 1\leq j<i}[i - j \mid a_{ij} \neq 0]$.

A definição 1.26 significa que a largura de banda $\beta(A_S)$ é o maior número de entradas, a partir de coeficiente não nulo mais à esquerda da diagonal principal da matriz, até a diagonal principal, considerando-se todas as n linhas da matriz. Portanto, a definição 1.26 também significa que a largura de banda de uma matriz simétrica A_S é o número de subdiagonais, com coeficientes não nulos (representados por zeros), à esquerda da diagonal principal da matriz. Por exemplo, $\beta(A_S) = 7$ na matriz A_S com numeração S mostrada na figura 1.1. A matriz A e a numeração S podem ser omitidas quando estão claramente compreendidas no contexto. Com isso, escreve-se simplesmente $\beta = 7$.

Definição 1.27 (largura de banda do vértice no grafo não direcionado)**.** *Equivalentemente à definição 1.25, a largura de banda do vértice* v *do grafo não direcionado* $G_S = (V, E)$*, para uma numeração* S *dos vértices, é definida como* $\beta_v(G_S) = \max_{u\in Adj(G_S,v)\wedge s(u)<s(v)}[s(v) - s(u)]$*, em que* $s(v)$ *e* $s(u)$ *são os rótulos dos vértices* v *e* u*, respectivamente.*

Definição 1.28 (largura de banda no grafo não direcionado)**.** *Equivalentemente à definição 1.26, a largura de banda do grafo não direcionado* $G_S = (V, E)$*, para uma numeração* S *dos vértices, é definida como* $\beta(G_S) = \max_{v\in V}[\beta_v(G_S)] = \max_{\{v,u\}\in E\wedge s(u)<s(v)}[s(v) - s(u)]$*, em que* $s(v)$ *e* $s(u)$ *são os rótulos dos vértices* v *e* u*, respectivamente.*

Por exemplo, no grafo mostrado na figura 1.1, $\beta(G_S) = 7$. O grafo $G = (V, E)$ e a numeração S podem ser omitidos quando estão claramente compreendidos no contexto. Com isso, escreve-se simplesmente $\beta = 7$.

Para uma definição para matrizes assimétricas, pode-se utilizar $\beta(A_S) = \max\left[\max_{1\leq i\leq n, 1\leq j<i}[i - j \mid a_{ij} \neq 0], \max_{1\leq i\leq n, i<j\leq n}[j - i \mid a_{ij} \neq 0]\right]$. Equivalentemente para um grafo direcionado, $\beta(G_S) = \max\left[\max_{(v,u)\in E\wedge s(u)<s(v)}[s(v) - s(u)], \max_{(v,u)\in E\wedge s(v)<s(u)}[s(u) - s(v)]\right]$, em que $s(v)$ e $s(u)$ são os rótulos dos vértices v e u, respectivamente.

Foca-se neste texto em matrizes simétricas. Há algumas formas de se manipular uma matriz assimétrica A_a. Uma das maneiras mais simples e com menores custos computacionais (tempo e espaço) é aplicar o método heurístico para redução de largura de banda na matriz simétrica

resultante de $A_a + A_a^{\mathrm{T}}$. Assim, a renumeração obtida pelo método é utilizado para renumerar as linhas da matriz assimétrica original. Essa estratégia apresentou os piores resultados em redução de largura de banda ao utilizar o método RCM [36] em comparação com as outras duas estratégias mostradas por Reid e Scott [101]. Entretanto, essa é a estratégia utilizada no *software* matemático Matlab [111], por exemplo. O método RCM é descrito na subseção 2.4.2, na página 32.

Uma matriz é em banda quando todos os seus coeficientes diferentes de nulo (ou zero) estão dispostos em torno da diagonal principal. Com a definição 1.26, a largura da banda da matriz simétrica é $2\beta + 1$, consistindo das porções inferior e superior da banda, mais a diagonal principal. Com isso, semibanda superior (inferior) consiste de todos os β elementos na porção superior (inferior) da banda, em que $0 < j - i \leq \beta$ ($0 < i - j \leq \beta$). Dessa forma, há β elementos em cada linha (ou subdiagonais) na semibanda da matriz, i.e., a partir da diagonal principal nas porções superior e inferior da matriz. Assim, β é a largura da semibanda, em vez da largura da banda da matriz. Assim, alguns autores definem semilargura de banda (*semibandwidth* [106] ou *half-bandwidth* [2]) com o mesmo significado de largura de banda mostrada na definição 1.26. Da mesma forma, a largura de banda é $2\beta + 1$. Por outro lado, a nomenclatura adotada neste presente texto está de acordo com diversos artigos [89, 14, 37, 10, 40, 88, 98, 28] (para citar alguns até 1979), relatórios técnicos [110, 39] (para citar alguns até 1980) e livros [36, 19].

Definição 1.29 (envelope)**.** *O envelope da matriz $A_S \in \mathbb{R}^{n\times n}$, com numeração S, é definido como $Env(A_S) = \{(i,j) : (1 \leq i \leq n, 1 \leq j < i)\ i - j \leq \beta_i(A_S)\}$.*

Assim, o envelope é o conjunto de pares de índices, isto é, as entradas da matriz, entre a primeira entrada de cada linha diferente de nulo (ou zero) e a diagonal principal. A definição 1.29 significa que, considerando-se as subdiagonais à esquerda da diagonal principal, os coeficientes nulos mais à esquerda da i-ésima linha, em $\beta_i(A_S)$, não compõem o envelope. O envelope de uma matriz simétrica inclui somente entradas da porção inferior da matriz.

Definição 1.30 (perfil (*profile*) da matriz)**.** *O perfil (profile) da matriz $A_S \in \mathbb{R}^{n\times n}$, para $A_S = [a_{ij}]$, com numeração S, é definido como $|Env(A_S)| = \sum_{i=1}^{n} \beta_i(A_S) = \sum_{i=1}^{n} \left[i - \min_{1\leq j<i} [j \mid a_{ij} \neq 0] \right] = \sum_{i=1}^{n} \max_{1\leq j<i} [i - j \mid a_{ij} \neq 0]$.*

A definição 1.30 significa que o perfil (*profile*), também chamado de tamanho do envelope da matriz A, é a soma do número de entradas, em cada linha, do primeiro coeficiente não nulo até a diagonal principal.

Por exemplo, como $\beta_1(A_S) = \beta_2(A_S) = 0$, $\beta_3(A_S) = 2$, $\beta_4(A_S) = 1$, $\beta_5(A_S) = 4$, $\beta_6(A_S) = \beta_8(A_S) = \beta_{10}(A_S) = 5$, $\beta_7(A_S) = 6$, $\beta_9(A_S) = 7$ na matriz mostrada na figura 1.1, então, o *profile* é $2 \cdot 0 + 2 + 1 + 4 + 3 \cdot 5 + 6 + 7 = 35$.

Definição 1.31 (distância)**.** *A distância $d(v, u)$ entre dois vértices $v, u \in V$ é o tamanho do menor caminho entre eles.*

Definição 1.32 (excentricidade)**.** *A excentricidade $\ell : V \to \mathbb{N}$ de um vértice $v \in V$ é dada por $\ell(v) = \max_{u \in V}[d(v, u)]$.*

Definição 1.33 (diâmetro do grafo)**.** *O diâmetro $\Phi(G) = \max_{v \in V}[\ell(v)] = \max_{v,u \in V}[d(v, u)]$ é a maior excentricidade encontrada em $G = (V, E)$.*

Definição 1.34 (vértice periférico)**.** *Um vértice v é considerado periférico se a sua excentricidade é igual ao diâmetro do grafo, ou seja, $\ell(v) = \Phi(G)$.*

Definição 1.35 (vértice pseudoperiférico)**.** *Um vértice v é considerado pseudoperiférico se a sua excentricidade for próxima ao diâmetro do grafo, ou seja, $\ell(v) \approx \Phi(G)$.*

O conceito de vértice pseudoperiférico foi introduzido por Gibbs, Poole e Stockmeyer [40].

Definição 1.36 (estrutura de nível enraizada)**.** *Seja o grafo $G = (V, E)$ conexo e simples. Dado um vértice $v \in V$, a estrutura de nível enraizada do vértice v, com profundidade $\ell(v)$, é o particionamento $\mathscr{L}(v) = \{L_0(v), L_1(v), \ \ldots, L_{\ell(v)}(v)\}$, em que $L_0(v) = \{v\}$ e $L_i(v) = Adjc(L_{i-1}(v)) - \bigcup_{j=0}^{i-1} L_j(v)$, para $i = 1, 2, \ 3, \ \ldots, \ell(v)$ e $Adjc(.)$ retorna os vértices adjacentes aos vértices do argumento.*

Isso significa que uma estrutura de nível enraizada de um vértice de um grafo conexo e simples é um particionamento do conjunto de vértices em camadas (ou níveis) de vértices com as mesmas distâncias para um vértice determinado no nível 0 da estrutura. O número de partições (ou níveis) de $\mathscr{L}(v)$ é $\ell(v) + 1$ e $\bigcup_{i=0}^{\ell(v)} L_i(v) = V$.

Mostra-se um exemplo de montagem de uma estrutura de nível enraizada na figura 1.4. Para isso, considere o grafo da figura 1.1.

Para a estrutura de nível enraizada mostrada na figura 1.4, o vértice inicial, com rótulo 1, foi escolhido arbitrariamente. Neste exemplo, $\ell(s^{-1}(1)) = 3$. O particionamento dos vértices em níveis, a partir do vértice com rótulo 1, é:

- vértice com rótulo 1 no nível $L_0(s^{-1}(1))$,

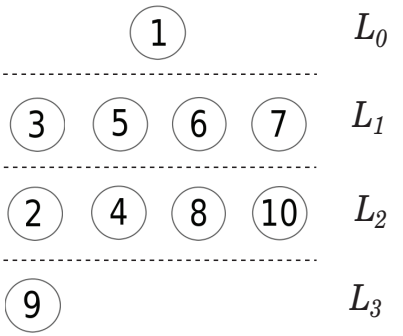

Figura 1.4: Rótulos dos vértices na estrutura de nível enraizada $\mathscr{L}(s^{-1}(1))$ do grafo mostrado na figura 1.1.

- vértices com rótulos 3, 5, 6 e 7 no nível $L_1(s^{-1}(1))$,
- vértices coms rótulos 2, 4, 8 e 10 no nível $L_2(s^{-1}(1))$,
- vértice com rótulo 9 no nível $L_3(s^{-1}(1))$.

Definição 1.37 (largura de nível)**.** *A largura de nível $b(\mathscr{L}(u))$ é o número de vértices do nível com mais vértices, ou seja, $b(\mathscr{L}(u)) = \max_{0\leq i\leq \ell(u)}[|L_i(u)|]$.*

Por exemplo, na figura 1.4, $b(\mathscr{L}(s^{-1}(1))) = 4$.

Definição 1.38 (estrutura de nível)**.** *Seja o grafo $G = (V, E)$ conexo e simples. A estrutura de nível é o particionamento $\mathscr{K}(v,\dots) = \{K_0(v,\dots), K_1(v,\dots), \dots, K_{\ell(v)}(v,\dots)\}$, em que $v \in K_0(v,\dots)$ e $K_i(v,\dots) = Adjc(K_{i-1}(v,\dots)) - \bigcup_{j=0}^{i-1} K_j(v,\dots)$, para $i = 1, 2, 3, \dots, \ell(v)$ e $Adjc(.)$ retorna os vértices adjacentes aos vértices do argumento.*

Essa é uma estrutura de nível *não* enraizada em um só vértice, ou seja, pode ter mais de um vértice em seu nível 0. Consta, pelo menos, um vértice $v \in V$ no primeiro nível da estrutura, ou seja, no nível $K_0(v,\dots)$.

Segundo Cuthill e Mckee [14], se a renumeração S dos vértices de $G_S = (V, E)$ é realizada nível a nível de $\mathscr{K}(v,\dots)$, então, $\beta(G_S) \leq 2b(\mathscr{K}(v,\dots)) - 1$. Ainda, se a estrutura de nível é enraizada, a largura de banda de G_S não pode ser menor que a largura de nível da estrutura de nível enraizada, ou seja, $b(\mathscr{L}(v)) \leq \beta(G_S) \leq 2b(\mathscr{L}(v)) - 1$.

Definição 1.39 (aresta crítica)**.** *Uma aresta* $\{v, u\} \in E$ *no grafo* $G_S = (V, E)$ *com numeração* S *dos vértices é crítica se* $(\{v, u\} \in E)\ |s(v) - s(u)| = \beta(G_S)$*, ou seja,* $(\exists u)\ \beta_v(G_S) = \beta(G_S) \mid \{v, u\} \in E$*.*

Definição 1.40 (vértice crítico)**.** *Vértices* $v, u \in V$ *são vértices críticos do grafo* $G_S = (V, E)$ *se* $(\exists u \in V)$ *tal que* $\{v, u\}$ *é uma* aresta crítica *de* $G = (V, E)$*, ou seja,* $\beta_v(G_S) = \beta(G_S)$*.*

1.4 Evolução dos métodos heurísticos

As evoluções dos métodos heurísticos baseados em conceitos de teoria dos grafos e dos algoritmo meta-heurísticos são abordadas nas subseções 1.4.1 e 1.4.2, respectivamente.

1.4.1 Métodos baseados em teoria dos grafos

Em revisões da bibliografia [57, 9, 53], foram analisados diversos métodos heurísticos para redução de largura de banda. É possível que os principais métodos heurísticos tenham sido avaliados. Nessas revisões, diversos métodos foram considerados superados por outros ao serem analisados os resultados e os tempos de execuções dos métodos. Os métodos RCM [36], GPS [40] e KP-band [80] foram mencionados como os mais promissores algoritmos com baixo custo de execução. Esses métodos são baseados em conceitos da teoria dos grafos. O método RCM é apresentado nas subseção 2.4.2, na página 32. A heurística KP-band é apresentada na subseção 2.4.7, na página 41. O algoritmo GPS é apresentado na seção 2.5, na página 42. Posteriormente, foi verificado que a heurística KP-band retorna resultados similares ao método RCM, com tempos de execução também similares [63, 62].

Em revisão da bibliografia [53], foi constatado que o método RCM [36] era o algoritmo no estado da arte para redução de largura de banda de matrizes no contexto de redução do custo de execução de resolutores de sistemas de equações lineares. Esse método está disponível em diversos *softwares* matemáticos e bibliotecas, como descrito na subseção 2.4.2, na página 32. Todavia, com a grande diversidade de matrizes, oriundas de tantas áreas de aplicação, como mencionadas no apêndice A, na página 97, foi verificado que haveria métodos mais promissores por áreas de aplicação [53]. Com isso, em nova publicação [54], foi constatado que não seria necessário redução tão grande de largura de banda para fornecer resultados satisfatórios na redução do custo de execução do resolutor do sistema de equações lineares, em diferentes áreas de aplicação. Assim, foi introduzida a heurística RBFS-GL [54], que é apresentada na subseção 2.4.6, na página 40. Esse método heurístico

fornece reduções quase tão satisfatórias quanto o método RCM, mas, em geral, na metade do tempo de execução.

Em sequência a esses estudos, foi utilizada uma hiper-heurística baseada em colônica de formigas para gerar métodos heurísticos especializados por área de aplicação. Foram evoluídos os métodos heurísticos RCM [36], KP-band [80] e um novo método heurístico chamado de RLK [63, 62], apresentado na subseção 2.4.8, na página 42. Como mencionado, esses métodos foram evoluídos por área de aplicação. Assim, novos métodos heurísticos baseados em conceitos da teoria dos grafos, com baixo custo de execução, foram gerados, ou o melhor método heurístico foi selecionado por área de aplicação, incluindo a heurística RBFS-GL [54]. Assim, esses métodos heurísticos superaram os demais métodos considerados no estado da arte para redução do custo de execução de sistemas de equações lineares (RCM, KP-band e RBFS-GL). A hiper-heurística por colônia de formigas é apresentada na seção 3.3, na página 57.

Em resumo, para situações práticas para redução do custo de execução de resolutores de sistemas de equações lineares, os algoritmos baseados em conceitos da teoria dos grafos resultantes da hiper-heurística ACHH [63, 62] estão no estado da arte. Para situações gerais, o método RCM [36] é o método mais empregado. Por outro lado, a heurística RBFS-GL [54] apresenta redução de largura de banda quase tão satisfatória quanto o método RCM, mas com a metade do custo de execução. Alguns eventos e publicações importantes em redução de largura de banda no contexto de resolução de sistemas de equações lineares foram:

- 1950 - Engenheiros de estruturas perceberam que redução de largura de banda acelera a resolução de sistemas de equações lineares [89];
- 1965 - Em nosso conhecimento, Alway e Martin [2] apresentaram o primeiro algoritmo para redução de largura de banda de matrizes; é uma busca sistemática de permutações para redução da largura de banda, até o mínimo teórico ser alcançado; o algoritmo procura por uma quantidade inaceitável de permutações;
- 1968 - Algoritmo de Rosen [103] - troca pares numeração de pares de vértices a cada iteração; pode encontrar um mínimo local;
- 1969 - Método Cuthill-McKee [14];
- 1971 - Método RCM [37];
- 1976 - Algoritmo GPS [40];
- 1979 - Algoritmo George-Liu (GL) [35];

- 1981 - Método RCM-GL [36];
- 2011 - Método KP-band [80] - Koohestani e Poli evoluíram o método RCM por programação genética;
- 2018 - RBFS-GL [54] (cerca de metade do custo do método RCM, com quase a mesma redução de largura de banda);
- 2020 - *Ant Colony Hiperheuristic* (ACHH) - gera métodos heurísticos especialistas, por área de aplicação - evolução dos métodos RCM, KP-band, RLK e também pode selecionar o algoritmo RBFS-GL [62, 63].

1.4.2 Algoritmos meta-heurísticos

Na década de 1960, pesquisadores perceberam que não conseguiam projetar algoritmos rápidos para diferentes problemas de otimização. A teoria sobre $\mathcal{NP}$-Completude foi desenvolvida na década seguinte, no sentido que seria mais prático utilizar métodos heurísticos rápidos que entregassem soluções aproximadas às soluções ótimas dos problemas de otimização. Importantes meta-heurísticas foram desenvolvidas principalmente a partir das décadas de 1970 e 1980. Algumas das principais meta-heurísticas são Algoritmos Genéticos [32, 6, 72], Simulated Annealing [78], Busca Tabu [42], GRASP [31], Otimização por Colônia de Formigas [23, 12], VNS [93] entre muitas outras. Basicamente, meta-heurística é uma técnica de construção de métodos heurísticos. Na década de 1990, as meta-heurísticas passaram a ser aplicadas em grande escala nos problemas de otimização. Assim, os primeiros algoritmos meta-heurísticos para redução de largura de banda de matrizes foram projetados nessa década [9].

Em geral, algoritmos meta-heurísticos entregam resultados melhores que algoritmos baseados em conceitos da teoria dos grafos, a custo de execução bem maior. Assim, algoritmos meta-heurísticos, como VNS-band [94] e DRSA [113], são geralmente aplicados, com tempo razoável, em matrizes pequenas.

Nas revisões da bibliografia [57, 9, 53], os algoritmos meta-heurísticos FNCHC [86] e VNS-band [94] foram considerados os mais promissores para a redução de largura de banda. A heurística FNCHC é apresentada na seção 4.2, na página 70. A heurística VNS-band é apresentada na seção 4.3, na página 77. Posteriormente, foi publicada a heurística DRSA [113], que superou a heurística VNS-band em redução de largura de banda de matrizes muito pequenas, isto é, com dimensões menores que, aproximadamente, 1.000. A heurística DRSA é apresentada na seção 4.4, na página 90.

Como mencionado, os algoritmos meta-heurísticos tradicionais para redução de largura de banda de matrizes, tais como como VNS-band [94] e DRSA [102], apresentam custo de execução bastante superior aos algoritmos baseados em conceitos de teoria dos grafos [52, 50, 59, 60, 53, 54, 58, 62]. Com isso, algoritmos meta-heurísticos, como VNS-band e DRSA, são aplicáveis, com tempos razoáveis, a matrizes com dimensão até, aproximadamente, 1.000. Por outro lado, a heurística FNCHC [86] apresentou resultados satisfatórios em matrizes com dimensão média, isto é, com dimensões de 100 mil a 1 milhão [53], mas não apresentou resultados satisfatórios para matrizes de grande porte, ou seja, com dimensão acima de um milhão [55]. Por sua vez, o algoritmo GPS apresentou os melhores resultados em matrizes de grande porte e era o estado da arte nesse contexto [55]. Com isso, foi proposta a heurística FNCHC+, que superou os resultados dos algoritmos GPS e FNCHC em matrizes de grande porte [55]. Assim, a heurística FNCHC+ é considerada o algoritmo meta-heurístico no estado da arte na redução de largura de banda de matrizes de grande porte. Esse método heurístico apresenta custo de execução superior aos métodos baseados em conceitos de teoria dos grafos no estado da arte (resultante do sistema ACHH [62]), mas apresenta resultados melhores em redução de largura de banda [55]. Inclusive, a heurística FNCHC+ apresentou melhores tempos de execução que o algoritmo GPS [47], considerado o método no estado da arte anterior para redução de largura de banda de matrizes de grande porte. A heurística FNCHC+ é apresentada na subseção 4.2.6, na página 76.

Em resumo, a heurística DRSA [113] é considerada o algoritmo meta-heurístico no estado da arte na redução de largura de banda de matrizes com dimensões muito pequenas, ou seja, com dimensão de aproximadamente 1.000. Para matrizes de grande porte, ou seja, com dimensões maiores que um milhão, o algoritmo meta-heurístico no estado da arte é a heurística FNCHC+ [55]. A seguir, são citados alguns algoritmos e algumas meta-heurísticas aplicadas na construção de métodos heurísticos para redução de largura de banda:

- 1995 - Simulated Annealing (SA-DJ) - Dueck e Jeffs [24] compararam os resultados em 18 grafos com o algoritmo GPS [40];
- 1996 - Busca Tabu (TS-ET) - Esposito e Tarricone [27];
  - 1998 - WBRA (variação do RCM) - Esposito e colaboradores [26] compararam com TS-ET [27] e RCM (Matlab) [36];
- 2001 - Busca Tabu (TS) - Martí e colaboradores [91] compararam os resultados em 37 grafos com até 199 vértices e 89 grafos com até 1.000 vértices com GPS [40] e SA-DJ [24];

- 2004 - GRASP-PR - Piñana et al. [100] compararam os resultados em 31 grafos com até 199 vértices e 82 grafos com até aproximadamente 1.000 vértices (padrão de 113 grafos) com GPS [40], TS [91] e GRASP;
- 2006 - Algoritmo Genético (GA) - Lim et al. [85];
- 2006 - Node Shift (NS) com *Hill Climbing* - Lim et al. [85] compararam os resultados nos 113 grafos e outros 14 grafos com GA [85], GRASP-PR [100], GPS [40], WBRA [26] e RCM (Matlab) [36] - o algoritmo NS apresentou resultados melhores;
- 2007 - Otimização por enxame de partículas [*Particle Swarm Optimization* (PSO)] - Lim et al. [84];
- 2007 - NCHC e FNCHC (autores não mencionam, mas usa conceitos da meta-heurística ILS) - Lim et al. [86] compararam os resultados nos 113 grafos com GPS [40], TS [91], GRASP-PR [100], GA [85], NS [85], NCHC - NS apresentou resultados melhores e FNCHC apresentou custo computacional quase tão baixo quanto o algoritmo GPS;
- 2008 - Simulated Annealing (SA) - Rodriguez-Tello e colaboradores [102] compararam os resultados nos 113 grafos com TS [91], GRASP-PR [100], GPS [40] e SA-DJ [24];
- 2010 - VNS - Mladenović e colaboradores [94] compararam os resultados nos 113 grafos com TS [91], GRASP-PR [100], SS-TS [7] (resultados de uma versão preliminar do artigo), GA [85], NS [85], PSO [84], SA [102]; mostraram que SA [102] era o algoritmo meta-heurístico no estado da arte em redução de largura de banda - a heurística VNS-band tornou-se o algoritmo meta-heurístico no estado da arte em grafos com dimensão até aproximadamente 1.000;
- 2011 - *Scatter Search* e Busca Tabu (SS-TS)- Campos et al. [7];
- 2015 - Simulated Annealing (DRSA) - Torres-Jimenez e colaboradores [113] compararam os resultados nos 113 grafos com GRASP-PR [100], SA [102] e VNS-band [94] - tornou-se o algoritmo meta-heurístico no estado da arte em grafos com dimensão até 1.000;
- 2022 - ILS e FNCHC+ (também com conceitos da meta-heurística ILS, variação da heurística FNCHC [86]) - Gonzaga de Oliveira e Carvalho [55] compararam os resultados em centenas de matrizes, incluindo 76 matrizes de grande porte (de um milhão de vértices a 19 milhões de vértices) com GPS [40], ILS, FNCHC [86],

VNS-band [94] e DRSA (implicitamente) - tornou-se o algoritmo meta-heurístico no estado da arte para grafos de grande porte - o algoritmo GPS [40] retornou custo de execução menor que a heurística FNCHC+;

- 2023 - FNCHC+ - Gonzaga de Oliveira [47] comparou os resultados em 32 matrizes de grande porte (de um milhão de vértices a 24 milhões de vértices) com o algoritmo GPS [40] - a heurística FNCHC+ superou o algoritmo GPS em redução de largura de banda a custos de execução menores que o algoritmo GPS.

1.5 Armazenamento

Em relação a armazenamento, muitas vezes não é prático armazenar todas as subdiagonais da banda de uma matriz porque, geralmente, pode ocorrer que poucas linhas da banda tenham largura de banda grande. Nesses casos, pode ser melhor utilizar o esquema de armazenamento por banda variável ou por envelope [74]. Envelope é explicado na definição 1.29, na página 13 da seção 1.3.

Uma variação do esquema de Jennings [74] é obtida ao se armazenar a transposta do envelope das subdiagonais abaixo da diagonal principal. Esse esquema é chamado de *skyline* [30].

Ocorrem, ainda, casos em que mesmo o envelope tem um número grande de coeficientes nulos. Nesses casos, um esquema de armazenamento que pode ser utilizado é, para cada linha, os coeficientes não nulos serem armazenados em uma lista encadeada e também haver outra lista para ligar as listas de cada linha [56]. Entretanto, o custo é alto para se percorrer listas de adjacências, em comparação com outras estruturas de dados para armazenamento de matrizes. A estrutura *Compressed Sparse Row* (CSR) é geralmente utilizada para o armazenamento de matrizes [45]. Também é utilizada uma versão híbrida dos formatos CSR e *skyline* [41].

Capítulo 2

Métodos baseados em conceitos de teoria dos grafos

2.1 Introdução

Mostra-se, neste capítulo, uma introdução a métodos heurísticos para redução de largura de banda de matrizes baseados em conceitos de teoria dos grafos. Os métodos heurísticos apresentados neste capítulo formam uma boa amostra da grande quantidade de métodos heurísticos aplicados no problema de otimização, utilizados na aceleração de resolutores de sistemas de equações lineares.

Neste capítulo, são apresentados métodos heurísticos que utilizam implicitamente a estrutura de nível (veja as definições 1.36 e 1.38 na página 14 da seção 1.3) ou variações dessa estrutura para redução de largura de banda de matrizes.

Este capítulo está dividido como descrito a seguir. Comenta-se sobre resolução de sistemas de equações lineares na seção 2.2. Como mencionado, os métodos heurísticos para redução de largura de banda baseados em conceitos de teoria dos grafos são variações da busca em largura. Por isso, a busca em largura é apresentada na seção 2.3. Os métodos Cuthill-McKee (CM) [14], Cuthill-McKee reverso (RCM) [37] e GPS [40] são métodos heurísticos clássicos para a redução de largura de banda de matrizes. Os métodos Cuthill-McKee [14] e RCM [37] são apresentados na seção 2.4. As heurísticas RBFS-GL, KP-band e RLK são descritas respectivamente nas seções 2.4.6, 2.4.7 e 2.4.8. O algoritmo GPS [40] é abordado na seção 2.5. Finalmente, exemplos de tempos de execução de métodos heurísticos para redução de largura de banda de matrizes são apresentados na seção 2.6.

2.2 Sistemas de equações lineares

Métodos heurísticos para redução de largura de banda de matrizes são particularmente importantes na redução do custo de execução na resolução de sistemas de equações lineares. A resolução de sistemas de equações lineares do tipo $Ax = b$, em que A é uma matriz esparsa, é central em diversas simulações em Ciência e Engenharias e é, geralmente, a fase que requer o maior custo de execução na simulação. A resolução de sistemas de equações lineares de forma confiável e eficiente é de grande importância para a viabilidade e desempenho de um número grande de softwares. Esse tema é relevante a cientistas e engenheiros, e também a uma grande e variada comunidade de pessoas que desconhece que no núcleo do software utilizado consta um resolutor de sistemas de equações lineares [106].

A principal fonte de problemas com matrizes de grande porte é oriunda da discretização (e linearização) de equações diferenciais parciais elípticas ou parabólicas [3]. Por exemplo, a discretização de um modelo matemático pode levar a um sistema de equações não lineares de grande porte. Métodos de linearização reduzem-no a uma série de sistemas de equações lineares de grande porte. Alguns dos métodos numéricos mais populares para a resolução de problemas relacionados a fenômenos físicos modelados por equações diferenciais parciais são os métodos dos elementos finitos, das diferenças finitas e dos volumes finitos. Sistemas de equações lineares de grande porte são gerados nas aplicações desses métodos. Por outro lado, sistemas algébricos de grande porte também são oriundos de aplicações não modeladas por equações diferenciais parciais, como o projeto e a análise de circuitos integrados, redes de sistemas de potência, processos em engenharia química e modelos econômicos [56]. Diversos exemplos de áreas de aplicação do problema de redução de largura de banda são citados no apêndice A. Por exemplo, Kaveh [75, p. 221] explica que, na mecânica das estruturas, 30% a 50% do custo computacional pode ser relacionado à resolução de sistemas de equações lineares, para problemas encontrados na prática; e isso pode chegar a 80% em problemas não lineares de otimização de estruturas. Por serem de grande porte, são necessárias muita memória e alto custo de processamento para armazenar e resolver esses sistemas de equações lineares.

A seguir, métodos para resolução de sistemas de equações lineares são abordados na subseção 2.2.1. Comenta-se sobre reordenações de linhas e de colunas de matrizes na subseção 2.2.2.

2.2.1 Métodos para resolução de sistemas de equações lineares

Por causa da importância da aceleração de resolutores de sistemas de equações lineares do tipo $Ax = b$, foi proposta uma grande quantidade de métodos heurísticos para as reduções de largura de banda e de *profile*. Esses problemas são relacionados, mas independentes.

Métodos heurísticos para renumerar os vértices do grafo subjacente à matriz A têm sido desenvolvidos especificamente para a redução de largura de banda ou de *profile*. O estudo foi orientado pelo interesse dos pesquisadores no método de resolução do sistema de equações lineares. Apesar de estudos específicos em apenas um desses casos, muitos dos métodos heurísticos propostos são efetivos, muitas vezes, na redução dessas características em conjunto. Entretanto, pode ser que, com a redução da largura de banda, não haja redução do *profile*. O inverso é verdadeiro [56].

Resolutores de sistemas de equações podem ser divididos em métodos diretos e métodos iterativos. Na ausência de erro de arredondamento, métodos diretos para resolver o sistema de equações lineares $Ax = b$ produzem a solução x em aritmética exata. Por sua vez, métodos iterativos são úteis para problemas envolvendo matrizes de grande porte ou especiais.

Métodos heurísticos para as reduções de largura de banda de matrizes podem acelerar tanto métodos iterativos quando métodos diretos para resolução de sistemas de equações lineares. Uma numeração de vértices adequada é desejada para se obter resoluções de sistemas de equações lineares com baixo custo de execução. Com uma numeração S dos vértices apropriada, a matriz (de coeficientes) A_S associada tem largura de banda pequena. Dessa forma, métodos heurísticos para redução de largura de banda são empregados como um passo de pré-processamento de matrizes na resolução de sistemas de equações lineares.

No método dos elementos finitos, ao se utilizar funções bases com suporte compacto, pode-se fazer com que a largura de banda da matriz seja correspondente à ordem do polinômio sendo empregado. Por sua vez, no método dos volumes finitos, a largura de banda da matriz correspondente, em geral, depende de como a malha é percorrida para se montar o sistema de equações lineares. Há formas de se percorrer os volumes de controle em tempo linear ao se utilizar uma curva de preenchimento de espaço, por exemplo [44, 61]. Entretanto, essa forma de percorrer os vértices da malha pode gerar um sistema de equações lineares com matriz com largura de banda grande. É necessário encontrar meios adequados para se montar o sistema de equações lineares para que seja resolvido com custo computacional baixo. A forma de mon-

tagem do sistema de equações lineares pode deixar de ser uma fonte de preocupação com a utilização de um método heurístico para reordenações das linhas e colunas da matriz de coeficientes [56]. Na maioria dos casos, essas reordenações de vértices afetam a taxa de convergência de métodos baseados no subespaço de Krylov [81], como é o caso do método dos gradientes conjugados [82, 71]. Ainda, o reordenamento das linhas e das colunas de matrizes é um dos itens mais importantes em implementações paralelas de resoluções diretas ou iterativas [104, p. 72]. Métodos iterativos e diretos para resolução de sistemas de equações lineares são abordados em mais detalhes a seguir.

Métodos iterativos

Políticas de paginação e a arquitetura hierárquica de memória dos projetos de computadores atuais favorecem programas que consideram localidade de referência em memória. Assim, a utilização de métodos heurísticos para redução de largura de banda é uma alternativa para se obter uma numeração do conjunto de vértices, do grafo subjacente à matriz A, de forma que forneça localidade espacial (em memória). Vértices com diferença pequena entre seus rótulos são mais prováveis de se localizarem no mesmo nível da hierarquia de memória dos computadores modernos. Por isso, métodos iterativos para resolução de sistemas de equações lineares são beneficiados pela redução de largura de banda, ou seja, vértices vizinhos têm mais chance de serem acessados em seguida. Com isso, já podem estar carregados no mesmo nível na hierarquia de memória.

O custo total de execução do método heurístico para redução de largura de banda, em conjunto com um pré-condicionador, no caso de métodos iterativos, mais o custo do resolutor do sistema de equações, deve ser menor que o custo do resolutor sozinho (em conjunto com um pré-condicionador, no caso de métodos iterativos), pelo menos quando se utiliza apenas um vetor de termos independentes. Portanto, no contexto de aceleração de resolutores de sistemas de equações lineares, o método heurístico para redução de largura de banda deve apresentar baixo custo de execução.

Na resolução de sistemas de equações lineares, métodos iterativos exigem menos memória e, geralmente, exigem menos operações que métodos diretos, mas não são considerados tão confiáveis quanto os métodos diretos. Há aplicações em que métodos iterativos falham e um pré-condicionamento da matriz A é necessário, apesar de nem sempre necessário, para que haja convergência em um tempo razoável [3]. Entretanto, métodos diretos têm escalabilidade insatisfatória em relação ao tamanho do problema, em termos de contagem de operações e exigências de memória, especialmente em problemas tridimensionais

oriundos da discretização de equações diferenciais parciais. Com isso, sistemas de equações lineares com matrizes de grande porte são, muitas vezes, resolvidos por métodos iterativos [56]. Veja, por exemplo, a obra de Saad [104], para detalhes sobre métodos iterativos para a resolução de sistemas de equações lineares. Scott e Tůma [106] abordaram métodos diretos e iterativos para resolução de sistemas de equações lineares.

Benzi [3] ainda explica que, após diversas ideias e técnicas da área de métodos diretos serem transferidas, na forma de pré-condicionadores, para métodos iterativos, os métodos iterativos passaram a ser cada vez mais confiáveis. Para a resolução de sistemas de equações lineares com matrizes esparsas de grande porte, um método iterativo eficiente é o método dos gradientes conjugados [71, 82], que pode ser utilizado se a matriz do sistema de equações lineares é simétrica e positiva definida.

Métodos diretos

Benzi [3] explica que métodos diretos têm sido tradicionalmente preferidos em análises de estruturas e modelagens de periféricos semicondutores, em grande parte da área de dinâmica de fluidos computacional e em diversas aplicações em que o problema não é modelado por equações diferenciais parciais, como circuitos e redes de sistemas de potência.

Exemplos de métodos diretos são as decomposições LU e de Choleski. Importantes exemplos de métodos diretos para resolução de sistemas de equações lineares são métodos multifrontais baseados na eliminação gaussina. Por exemplo, a rotina MA41, incluída na biblioteca de rotinas HSL [109], é um resolutor de sistemas de equações lineares assimétricos. A rotina é um método direto baseado em uma variante multifrontal esparsa da eliminação gaussiana. De acordo com a documentação, essa rotina é recomendada para matriz cujo padrão seja simétrico ou quase simétrico. A rotina MA57 é recomendada para matrizes esparsas simétricas. As rotinas MA38 e MA48 são recomendadas para matrizes muito assimétricas ou especiais.

A eliminação gaussiana necessita de $\mathcal{O}(n \cdot \beta^2)$ operações para a resolução de um sistema de equações lineares com n elementos e largura de banda β [110]. Claramente, se $\beta \cong n$, então, são realizadas $\mathcal{O}(n^3)$ operações, que apresenta custo de execução impeditivo. Ainda, a eliminação gaussiana requer $\mathcal{O}(n \cdot \beta)$ espaços na memória ao se utilizar armazenamento baseado em vetores[1] [110]. Veja a definição 1.26 de largura de banda, na página 12, na seção 1.3. Veja, por exemplo, a obra de Davis [19], para detalhes sobre métodos diretos para a resolução de sistemas de equações lineares. Como mencionado, Scott e Tůma [106] abordaram métodos diretos e iterativos para resolução de sistemas de

[1]No restante do texto, vetor é utilizado com o significado de *array* ou arranjo.

equações lineares.

2.2.2 Renumerações de linhas e de colunas de matrizes

Os métodos heurísticos para redução de largura de banda realizam permutações de linhas e colunas das matrizes, deixando-as com uma estrutura compacta e com coeficientes não nulos próximos à diagonal principal. Ao serem trocadas as linhas da matriz de um sistema de equações lineares, altera-se a ordem em que as equações são escritas. Ao se trocar as colunas, as incógnitas são renumeradas ou reordenadas.

Para uma sistema de equações lineares $Ax = b$, esses métodos heurísticos produzem uma matriz de permutação P, tal que PAP^{T} tem uma largura de banda pequena. O sistema de equações lineares permutado $PAP^{\mathrm{T}}(P\overline{x}) = P\overline{b}$ continua esparso, simétrico e positivo definido. As possíveis vantagens de resolver o sistema de equações lineares permutado é que a solução pode exigir um número menor de operações aritméticas, menor exigência de armazenamento do que o sistema original ou ambas as vantagens [87, 56].

2.3 Busca em largura

A busca em largura é um dos algoritmos mais importantes e estudados na teoria dos grafos. O algoritmo determina a distância de cada vértice alcançável a partir de um vértice inicial. Distância entre vértices é explicada na definição 1.31, na página 14 da seção 1.3. Se existir um caminho c do vértice u para o vértice v, então, v é *alcançável* (ou *atingível*) partindo-se do vértice u pelo caminho c [46].

A busca em largura foi proposta por Moore [96] no contexto de encontrar caminhos em labirintos. Este algoritmo é base de vários outros métodos com aplicações em diversas áreas na ciência e indústria. Por exemplo, percorrer os vértices do grafo é um componente fundamental em aplicações práticas como em suítes de *benchmark* de supercomputadores ou com foco em aplicações em grafos. A suíte de *benchmark* Graph500 e várias outras plataformas de *benchmark* de computação de alto desempenho utiliza a busca em largura como um componente central. A busca em largura também é um algoritmo base na detecção e descoberta de estruturas de comunidades, em biologia computacional, identificação de componentes conexas, computação de alto desempenho, automação de projetos eletrônicos, classificação, processamento de imagens, robótica, redes sociais [73, 11, 112]. Também é base para algoritmos no estado da arte no problema de redução de largura de banda [63, 62, 65, 55].

O algoritmo 2 recebe o grafo conexo $G = (V, E)$ e o vértice $v \in V$

inicial. Nesta versão da busca em largura, o algoritmo também rotula os vértices do grafo conexo $G = (V, E)$ com identificadores no intervalo $[1, |V|]$, na sequência em que são percorridos. Isso é realizado nas linhas 5 e 14.

Algoritmo 2: Busca em largura.

Entrada: grafo $G = (V, E)$ não direcionado e conexo; vértice $v \in V$ inicial;
Saída: determina a distância de todos os vértices a partir do vértice $v \in V$;

```
1  início
       // considera que, inicialmente, o atributo percorrido
          de todos os vértices esteja estabelecido como falso
2      v.percorrido ← verdadeiro;
3      v.distância ← 0;
4      rótulo ← 1;
5      v.rótulo ← rótulo;
6      Insere(v,F);
7      enquanto ( F ≠ ∅ ) faça
8          w ← Remove(F);
9          para cada ( vértice u adjacente a w ) faça
10             se ( u.percorrido = falso ) então
11                 Insere(u,F);
12                 u.distância ← w.distância+1;
13                 rótulo ← rótulo + 1;
14                 u.rótulo ←rótulo;
15                 u.percorrido ← verdadeiro;
16             fim-se;
17         fim-para-cada;
18     fim-enquanto;
19 fim.
```

O algoritmo 2 considera que o atributo percorrido de todos os vértices está inicialmente estabelecido com falso. O algoritmo inicia ao estabelecer como percorrido o vértice $v \in V$ na linha 2. O algoritmo estabelece como 0 a distância do vértice v na linha 3. Esse vértice é inserido na fila F na linha 6. Comumente, dados compostos são organizados em objetos, que são compostos por atributos. Um atributo é acessado utilizando a sintaxe de muitas linguagens orientadas a objetos: o nome do objeto, seguido de um ponto, seguido pelo nome do atributo. Por exemplo, a distância de um objeto vértice em relação ao vértice inicial é acessado por v.distância (veja a linha 3 do algoritmo 2).

Na estrutura de repetição mostrada nas linhas 7 a 18, enquanto a fila

F não estiver vazia, o algoritmo remove um vértice da fila F na linha 8. O vértice removido da fila é atribuído ao vértice w. Em seguida, o algoritmo percorre a lista de adjacências do vértice w na linha 9. O algoritmo atribui ao vértice u cada vértice adjacente ao vértice w. Se o vértice u ainda não foi percorrido (veja a linha 10), então, o algoritmo insere o vértice u na fila na linha 11, atribui a distância do vértice u como a distância do vértice w mais 1 na linha 12, rotula o vértice u na linha 14 e atribui como verdadeiro o atributo percorrido do vértice u na linha 15.

É removido um vértice da fila a cada iteração do laço **enquanto** nas linha 7 a 18. Assim, esse laço de repetição tem custo $\Theta(|V|)$. A fila F é simples. Assim, o custo da linha 8 também é $\Theta(|V|)$. A lista de adjacências de cada vértice é percorrida na linha 9. Assim, o algoritmo percorre todos os vértices e arestas do grafo $G = (V, E)$. Com isso, o custo do laço de repetição nas linhas 9 a 17 é $\Theta(|V| + |E|)$. Dessa forma, o custo de execução do algoritmo é $\Theta(|V| + |E|)$.

2.4 Métodos Cuthill-McKee, RCM e variações

Com o método Cuthill-McKee [14], reduz-se a largura de banda de uma matriz simétrica pela renumeração dos vértices do grafo subjacente à matriz. Dessa forma, são realizadas permutações simétricas das linhas e colunas da matriz.

O método Cuthill-McKee é similar à busca em largura, em que são percorridos todos os vértices alcançáveis do grafo, a partir do vértice inicial. A ordem em que os vértices adjacentes ao vértice corrente são percorridos é o que difere o método Cuthill-McKee da busca em largura. Uma abordagem gulosa é realizada no método Cuthill-McKee, em que os vértices são rotulados na ordem crescente de grau dos vértices adjacentes ao vértice corrente. Dado um vértice inicial, seus vértices adjacentes são rotulados em ordem crescente de grau e isso ocorrerá para os vértices adjacentes desses adjacentes e assim, sucessivamente, até que todos os vértices sejam rotulados. Rotular os vértices adjacentes em ordem crescente de grau proporciona uma boa configuração da matriz correspondente em relação a sua largura de banda, pois os vértices de maior grau ficam posicionados, no possível, nas linhas centrais da matriz.

O restante desta seção está dividida como descrito a seguir. O método Cuthill-McKee é apresentado na subseção 2.4.1. Apresenta-se o método Cuthill-McKee reverso (RCM) na subseção 2.4.2. A análise de complexidade desses métodos é apresentada na subseção 2.4.3. Um exemplo é mostrado na subseção 2.4.4. O algoritmo George-Liu [35] para encontrar um vértice pseudoperiférico é apresentado na subseção

2.4.5. As heurísticas RBFS-GL, KP-band e RLK são descritas nas subseções 2.4.6, 2.4.7 e 2.4.8, respectivamente.

2.4.1 Método Cuthill-McKee

O método Cuthill-McKee é mostrado no algoritmo 3. O algoritmo recebe um grafo $G = (V, E)$ conexo, em que V é o conjunto de vértices e E é o conjunto de arestas. Outro parâmetro do algoritmo é o vértice inicial $v \in V$.

Algoritmo 3: Cuthill-McKee.

Entrada: grafo $G = (V, E)$ conexo; vértice inicial $v \in V$;
Saída: renumeração S com $|V|$ entradas;

```
1  início
2      s(v) ← 1;
3      i ← 1; // vértices já renumerados
       // próximo da "fila" de vértices cujos vértices
4      j ← 1; // adjacentes serão renumerados
5      enquanto ( i < |V| ) faça
6          para cada ( vértice w ∈
           Adj(G, s^-1(j)) − {s^-1(1), s^-1(2), ..., s^-1(i)}, em ordem
           crescente de Grau(G, w) ) faça
7              i ← i + 1;
8              s(w) ← i;
9          fim-para-cada;
10         j ← j + 1;
11     fim-enquanto;
12     retorna S;
13 fim.
```

Na linha 2, 1 é atribuído a $s(v)$, em que S é a numeração dos vértices a ser retornada. Veja a definição 1.23, mostrada na página 10 na seção 1.3.

Nas linhas 3 e 4, as variáveis i e j são inicializadas, respectivamente. A variável i indica a última entrada ocupada de S e a variável j é utilizada para percorrer os vértices do grafo.

No algoritmo 3, considera-se a função inversa da definição 1.23, mostrada na página 10 na seção 1.3: $S^{-1} : \{1, 2, \ldots, |V|\} \to V$. Assim, o valor de $s^{-1}(j)$ é o vértice com rótulo j, para $j \in [1, |V|]$.

A estrutura de repetição nas linhas 5 a 11 itera enquanto todos os vértices não tiverem sido rotulados. Os vértices adjacentes ao vértice com rótulo j, que ainda não foram renumerados, são rotulados na estrutura de repetição *para cada* nas linhas 6 a 9. Em mais detalhes, os vértices w, adjacentes ao vértice com rótulo j, são renumerados na

linha 8. Especificamente, na estrutura de repetição das linhas 5 a 11, enquanto a variável i for menor que o número de vértices do grafo, a cada iteração, são renumerados os vértices w adjacentes ao vértice com rótulo j, ou seja, $s^{-1}(j)$, em ordem crescente de $Grau(G, w)$ (veja a definição 1.12, na página 8), exceto os vértices que já foram renumerados (veja a linha 6). A saída do algoritmo é a sequência de vértices em S, na linha 12.

2.4.2 Método RCM

George [37] descobriu que renumerar os vértices na ordem reversa em que foram rotulados, em vez da renumeração habitual do método Cuthill-McKee, reduz, frequentemente, o *profile* do grafo. Entretanto, a largura de banda é mantida. Assim, o método RCM é o método Cuthill-McKee, mas com numeração reversa. O método RCM [37] produz matrizes com as mesmas larguras de banda do método Cuthill-McKee. Liu e Sherman [88] e Liu [87] provaram que o método RCM retorna matriz renumerada com *profile*, pelo menos, tão satisfatórios quanto o método Cuthill-McKee. Depois da publicação do método RCM, os pesquisadores passaram a reverter a numeração do método.

Na método RCM, $s(v)$, i e j recebem $|V|$ em vez de 1 nas linhas 2, 3 e 4 do algoritmo 3, respectivamente. Assim, a estrutura de repetição, mostrada nas linhas 5 a 11, opera enquanto $i > 1$. Com isso, as variáveis i e j são decrementadas, em vez de incrementadas, nas linhas 7 e 10 do algoritmo 3, respectivamente.

2.4.3 Análise de complexidade

Conforme demonstrado por Liu [87], a complexidade no pior caso das heurísticas Cuthill-McKee e RCM é dada pelo produto entre o grau m do vértice de maior grau do grafo e o número de arestas $|E|$. Liu [87] obteve a complexidade, no pior caso, considerando que fosse utilizado o algoritmo por inserção no passo de ordenação dos vértices na linha 6 do algoritmo 3. Com isso, considere que todas as ordenações na linha 6 exijam, no máximo, $\frac{n^2}{c}$ operações, para n vértices e uma constante c. Esse número de operações é menor que $\frac{1}{c}\sum_{v\in V} Grau(G,v)^2 \leq \frac{m}{c}\sum_{v\in V} Grau(G,v) \leq \frac{2m|E|}{c}$ porque, para um grafo não direcionado, ocorre $\sum_{v\in V} Grau(G,v) = 2|E|$. Com isso, $\frac{2m|E|}{c}$ é o limite superior do número de operações na linha 6.

O número de operações da estrutura de repetição na linha 5 é $\Theta(|V|)$. Com isso, a complexidade no pior caso desse algoritmo é $\mathcal{O}(m \cdot |E|)$. O limite inferior desse algoritmo, considerando-se um grafo em que o vértice inicial tem ligações com todos os demais vértices e

utilizando-se, por exemplo, o algoritmo *merge sort* no passo da ordenação, é $\Omega(|V| \lg |V|)$.

2.4.4 Exemplo

A seguir, é mostrado um exemplo de execução do método Cuthill-McKee. Apresenta-se apenas o método Cuthill-McKee, pois o métodos Cuthill-McKee e RCM diferem apenas na reversão da ordem da renumeração final. Considere o grafo da figura 2.1 e, ao seu lado, sua representação matricial M.

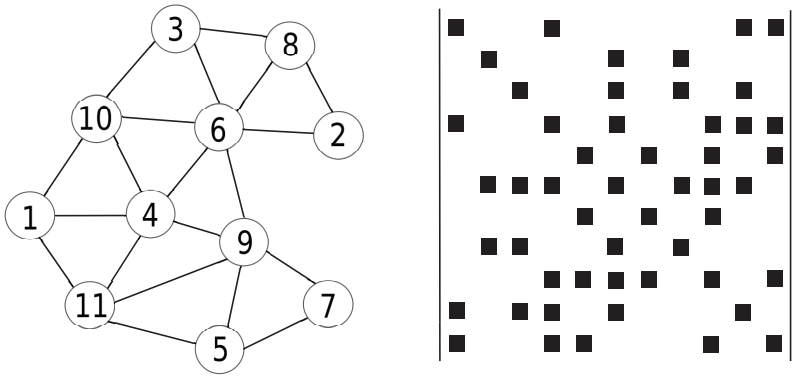

Figura 2.1: Grafo e sua representação matricial para um exemplo do método Cuthill-McKee [56].

Escolheu-se, arbitrariamente, o vértice com rótulo original 9 como o vértice inicial. O vértice com rótulo original 9 recebe 1 como o novo rótulo. Em seguida, os vértices adjacentes a esse vértice são rotulados, em ordem crescente de grau. Na tabela 2.1, são mostrados os vértices adjacentes ao vértice com rótulo original 9, ordenados por seus graus. Esses vértices são rotulados nessa ordem.

Rótulo original do vértice	7	5	11	4	6
Grau	2	3	4	5	6
Novo rótulo do vértice	2	3	4	5	6

Tabela 2.1: Rótulos originais, graus e novos rótulos dos vértices adjacentes ao vértice com rótulo 9 no grafo mostrado na figura 2.1.

O próximo vértice a ser processado é o vértice com rótulo original 7. O novo rótulo desse vértice é 2, como mostrado na tabela 2.1. Nenhum outro vértice é rotulado a partir desse vértice porque os vértices adja-

centes a esse vértice já foram rotulados (vértices com rótulos originais 9 e 5 e com novos rótulos 1 e 3, respectivamente).

O próximo vértice a ser processado é o vértice com rótulo original 5. O novo rótulo desse vértice é 3, como mostrado na tabela 2.1. Nenhum outro vértice é rotulado a partir desse vértice porque os vértices adjacentes a esse vértice também já foram rotulados (vértices com rótulos originais 9, 7 e 11 e novos rótulos 1, 2 e 4, respectivamente).

O próximo vértice a ser processado é o vértice com rótulo original 11. O novo rótulo desse vértice é 4, como mostrado na tabela 2.1. Neste passo, o vértice com rótulo original 1 recebe o rótulo 7.

Esse processo é realizado até que todos os vértices sejam rotulados. A nova numeração é mostrada na figura 2.2.

O resultado da renumeração é mostrado na figura 2.3, juntamente com a matriz M_1 correspondente. O grafo com a renumeração pelo método RCM e sua representação matricial M_2 são mostrados na figura 2.4. Neste exemplo, $\beta(M) = 10$, $\beta(M_1) = \beta(M_2) = 5$, $|Env(M)| = 39$, $|Env(M_1)| = 33$ e $|Env(M_2)| = 24$. Esses dados estão sumarizados na tabela 2.2.

Matriz	Largura de banda	*Profile*
M (original)	10	39
M_1 (Cuthill-McKee)	5	33
M_2 (RCM)	5	24

Tabela 2.2: Largura de banda e perfil (*profile*) de matrizes do exemplo para os métodos Cuthill-McKee e RCM [56].

Com a numeração revertida pelo método RCM, as colunas da matriz M_1 transformam-se nas linhas da matriz M_2. Conforme destacado nas linhas tracejadas nas representações de matrizes das figuras 2.3 e 2.4, as larguras de banda das linhas 4, 5 e 6, isto é, $\beta_4(M_1)$, $\beta_5(M_1)$ e $\beta_6(M_1)$, não são contadas no *profile* de M_2 no método RCM. Os primeiros coeficientes dessas três linhas de M_1 pertencem à última linha de M_2. Com isso, há redução no *profile* de M_2 em relação ao *profile* de M_1.

2.4.5 Algoritmo George-Liu

Na subseção 2.4.5.1, há comentários sobre a escolha do vértice inicial do processamento do método RCM. A descrição do algoritmo George-Liu consta na subseção 2.4.5.2. Um exemplo é mostrado na subseção 2.4.5.3.

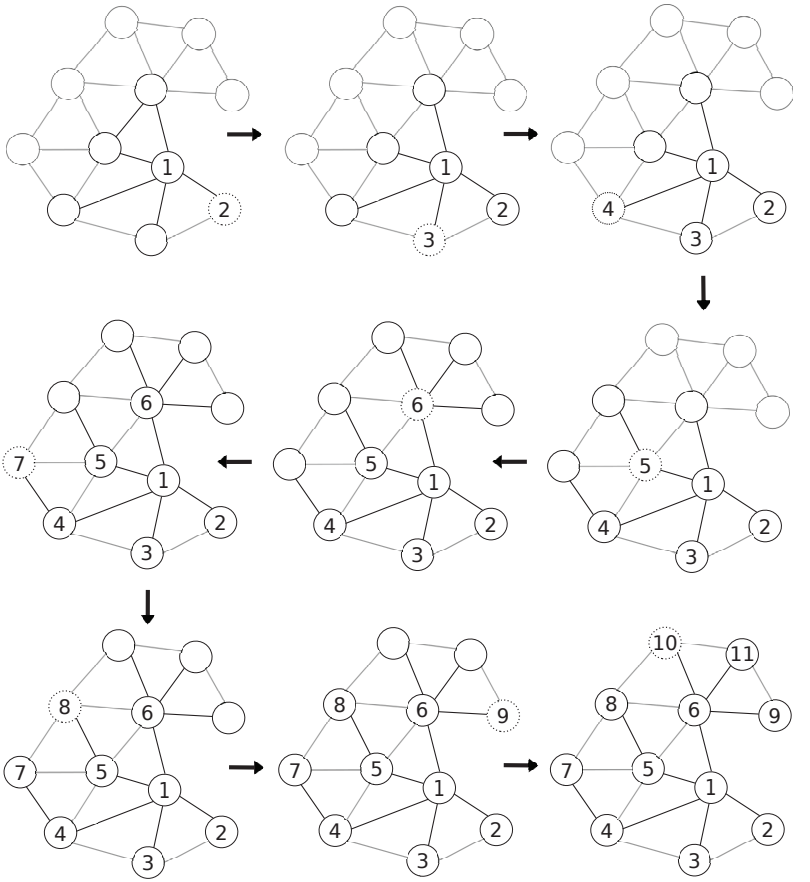

Figura 2.2: Passo a passo, de renumeração de vértices de um grafo pelo método Cuthill-McKee [56].

2.4.5.1 Vértice inicial do método RCM

Cuthill e McKee [14] constataram que a escolha do vértice inicial interfere na qualidade da solução: na largura de banda e no perfil (*profile*) da matriz resultante. Então, propuseram que a heurística começasse pelo vértice de grau mínimo do grafo. Nos métodos Cuthill-McKee [14] e RCM [37] originais, eram geradas as estruturas de nível de todos os vértices de grau mínimo. Em seguida, eram selecionadas as estruturas de nível que apresentassem a menor largura de nível (explicada na definição 1.37, na página 15 da seção 1.3). A renumeração era realizada com base em todas essas estruturas de nível e era escolhida a

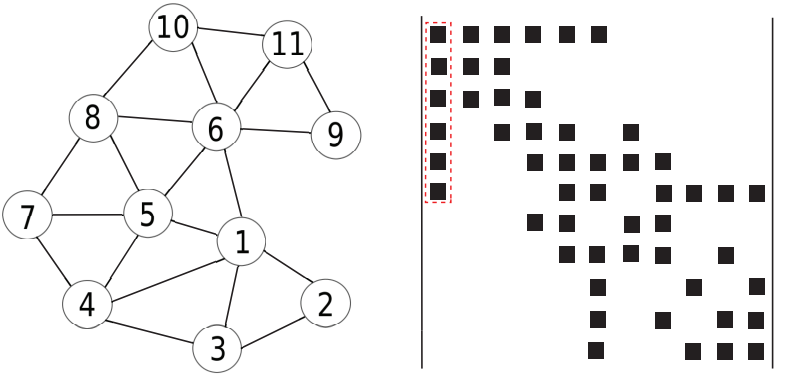

Figura 2.3: Grafo, com sua representação matricial, reordenado pelo método Cuthill-McKee [56].

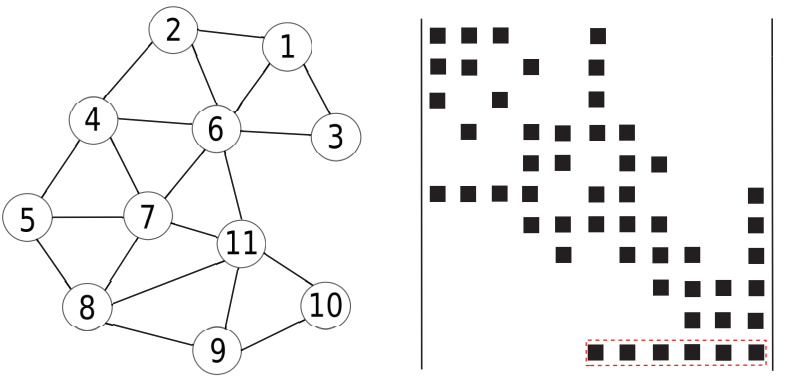

Figura 2.4: Grafo, com sua representação matricial, reordenado pelo método RCM [56].

renumeração que gerasse a menor largura de banda.

Como descrito, Cuthill e McKee [14] iniciaram a ordenação com o vértice de grau mínimo do grafo e que apresentasse estrutura de nível enraizada com a menor largura de nível. Todavia, nem sempre essa era uma escolha satisfatória. Após a proposta de Cuthill e McKee [14], houve publicações com propostas de algoritmos que encontram o vértice inicial, como o trabalho de Cheng [10].

Embora iniciar a renumeração por um vértice de grau mínimo apresente resultados satisfatórios, isso não garante que a largura de banda seja menor que uma solução em que o vértice inicial seja o vértice de maior excentricidade, ou seja, um vértice periférico. Um vértice pe-

riférico é ótimo vértice inicial para alguns métodos heurísticos para redução de largura de banda. Para encontrar o vértice periférico, é necessário, por exemplo, aplicar a busca em largura em todos os vértices do grafo. O custo dessa tarefa, no pior caso, é $\mathcal{O}(|V| \cdot (|V| + |E|))$, em que $|V|$ é o número de vértices e $|E|$ é o número de arestas do grafo $G = (V, E)$. Como esse custo computacional é impraticável para um algoritmo rápido para redução de largura de banda, a partir da publicação do algoritmo GPS [40], os pesquisadores começaram a utilizar um vértice pseudoperiférico (veja a definição 1.35, na página 14 da seção 1.3) em vez de serem testadas as renumerações com vértices iniciais com grau mínimo.

Gibbs et al. [40] constataram que o melhor vértice inicial seria um vértice cuja distância para outro vértice qualquer fosse igual ao diâmetro do grafo, caracterizando-se, assim, vértices periféricos. Isso porque, quanto maior a excentricidade de um vértice v, menos vértices haverá em um mesmo nível de uma estrutura de nível de v [14]. Diâmetro, vértice periférico, excentricidade e estrutura de nível estão definidos na seção 1.3, na página 6.

Na versão original do método RCM [37], também eram selecionados vértices iniciais com grau mínimo do grafo e eram geradas as estruturas de nível desses vértices. Em seguida, os vértices eram renumerados pelas estruturas de nível enraizadas que possuíssem as menores larguras de nível e era escolhida a renumeração que gerasse a menor largura de banda. (Largura de nível é explicada na definição 1.37, na página 15 da seção 1.3.) Com isso, o algoritmo original apresentava alto custo computacional. Isso motivou o desenvolvimento do algoritmo GPS [40]. Os autores propuseram iniciar a numeração por um vértice pseudoperiférico. Gibbs et al. [40] propuseram, então, como descrito na subseção 2.5.1, na página 43, um algoritmo que encontra vértice cuja excentricidade é próxima ao diâmetro do grafo, caracterizando-se, assim, um vértice pseudoperiférico.

George e Liu [35] publicaram um algoritmo para encontrar um vértice pseudoperiférico para o método RCM. Assim, o método RCM [36] inicia pelo vértice pseudoperiférico retornado pelo algoritmo George-Liu.

O método RCM [36], com vértice inicial fornecido pelo algoritmo George-Liu [35], está implementado em diversos *softwares* matemáticos, tais como MATLAB [111] e GNU Octave [25], com a função *symrcm*[2], Wolfram Mathematica [115] e ViennaCL [114], e nas bibliotecas Boost C++ [5][3], SciPy [105] e NetworkX [97].

[2]https://www.mathworks.com/help/matlab/ref/symrcm.html?requestedDomain=www.mathworks.com, https://octave.sourceforge.io/octave/function/sym rcm.html.

[3]http://www.boost.org/doc/libs/1_38_0/libs/graph/doc/cuthill_mckee_ordering.html.

2.4.5.2 Descrição do algoritmo George-Liu

George e Liu [35] avaliaram quatro características no algoritmo, descritas a seguir. A primeira característica consistiu em terminar a montagem das estruturas de nível enraizadas dos vértices no último nível de $\mathscr{L}(v)$ assim que encontrasse um vértice com estrutura de nível com largura de nível maior do que a já encontrada. A segunda característica foi a de iniciar o algoritmo com um vértice $v \in V$ arbitrário em vez de iniciar o algoritmo com um vértice de grau mínimo. A terceira foi a de montar a estrutura de nível enraizada apenas para o vértice com grau mínimo em $L_{\ell(v)}(v)$. A última característica foi escolher um vértice arbitrário em $L_{\ell(v)}(v)$ para a montagem da estrutura de nível. Após experimentos, o algoritmo final proposto por George e Liu [35] é a combinação da segunda e da terceira características. Esse algoritmo é descrito nesta subseção.

Mostra-se o pseudocódigo no algoritmo 4. Na fase da inicialização, um vértice arbitrário $v \in V$ é escolhido na linha 2 para ser o vértice inicial. Em seguida, a estrutura de nível enraizada $\mathscr{L}(v)$ é gerada na linha 3. Isso pode ser implementado por algoritmo similar à busca em largura, mostrada no algoritmo 2, na página 29 da seção 1.3.

Algoritmo 4: George-Liu.

```
   Entrada: grafo G = (V, E);
   Saída: vértice pseudoperiférico v ∈ V;
1  início
2  |  v ← VérticeArbitrário(V);
   |  // constrói estrutura de nível enraizada
3  |  L(v) ← EstruturaDeNívelEnraizada(v);
4  |  repita
5  |  |  u ← VérticeGrauMínimo(G, L_ℓ(v)(v)); // algoritmo 5
   |  |  // estrutura de nível enraizada no
   |  |  // vértice de grau mínimo de L_ℓ(v)(v)
6  |  |  L(u) ← EstruturaDeNívelEnraizada(u);
7  |  |  se ( ℓ(u) > ℓ(v) ) então
8  |  |  |  v ← u;
9  |  |  |  L(v) ← L(u);
10 |  |  fim-se;
11 |  até que (u ≠ v) ;
12 |  retorna v;
13 fim.
```

Na estrutura de repetição *repita* das linhas 4 a 11, o algoritmo constrói a estrutura de nível (na linha 6) do vértice com grau mínimo u em $\mathscr{L}_{\ell(v)}(v)$ (o algorito 5 é invocado na linha 5). O algoritmo 4 verifica se

a excentricidade de u é maior que a excentricidade de v [$\ell(u) > \ell(v)$] na linha 7. Se essa condição for satisfeita, então, o vértice u passa a ser o vértice v (linhas 8 e 9) e o processo é repetido. Caso contrário, o vértice pseudoperiférico será v, retornado na linha 12. Portanto, o algoritmo retorna o vértice v quando $\ell(v)$ for maior ou igual à excentricidade do vértice de grau mínimo entre os vértices em $L_{\ell(v)}(v)$. A função $\ell(v)$ pode ser implementada de forma similar à busca em largura, mostrada no algoritmo 2, na página 29 da seção 2.3.

Algoritmo 5: *VérticeGrauMínimo.*

Entrada: grafo $G = (V, E)$; conjunto de vértices $V' \subseteq V$;
Saída: vértice de grau mínimo $v \in V'$;

```
1  início
2  |  grau ← +∞;
3  |  para cada ( u ∈ V' ) faça
4  |  |  se ( Grau(G, u) < grau ) então
5  |  |  |  v ← u;
6  |  |  |  grau ← Grau(G, u); // veja a definição 1.12
7  |  |  fim-se;
8  |  fim-para-cada;
9  |  retorna v;
10 fim.
```

Após a publicação do algoritmo George-Liu [35] para encontrar um vértice pseudoperiférico, foram propostos diversos outros algoritmos com o mesmo objetivo. Foram realizados diversas comparações [49, 48, 51, 1, 64] e verificou-se que o algoritmo George-Liu [35] apresentou os melhores resultados entre os algoritmos com baixo custo de execução.

2.4.5.3 Exemplo

Como exemplo de uma execução do algoritmo de George e Liu [35], considere o grafo representado na figura 2.5. O vértice v, com rótulo 1, foi escolhido arbitrariamente para a inicialização do procedimento. Na figura 2.6, mostra-se a estrutura de nível enraizada do vértice v. O número de níveis de $\mathscr{L}(v)$ é 4 e $\ell(v) = 3$.

Atribui-se o vértice com rótulo 2 ao vértice u, pois é um vértice com grau mínimo de $L_{\ell(v)}(v) = \{s^{-1}(2), s^{-1}(7), s^{-1}(8)\}$, em que $Grau(G, s^{-1}(2)) = Grau(G, s^{-1}(7)) = 2$ e $Grau(G, s^{-1}(8)) = 3$. Como os vértices com rótulos 2 e 7 têm o mesmo grau, não há diferença na escolha de um ou outro. Neste exemplo, optou-se arbitrariamente pelo vértice com rótulo 2. Assim, $\mathscr{L}(u)$ é mostrada na figura 2.7. Como $\ell(v) = \ell(u)$, o algoritmo para, determinando o vértice com rótulo 1

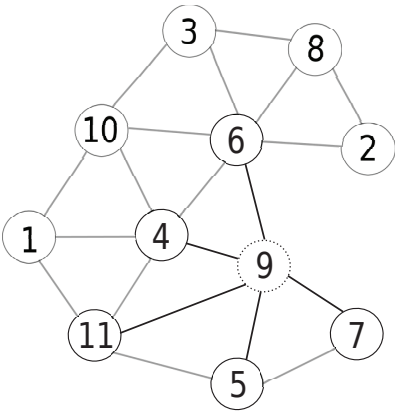

Figura 2.5: Um grafo para a escolha do vértice pseudoperiférico [56].

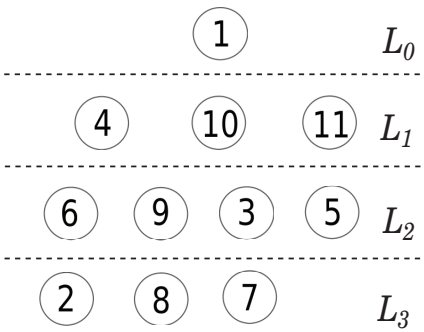

Figura 2.6: Rótulos dos vértices na estrutura de nível enraizada do vértice com rótulo 1 do grafo mostrado na figura 2.5 [56].

como o vértice pseudoperiférico. Neste exemplo, o algoritmo de fato encontrou um vértice periférico do grafo.

2.4.6 Heurística RBFS-GL

A heurística RBFS-GL foi utilizada no contexto de redução do custo de execução de sistemas de equações lineares [54]. Foi percebido que a redução em largura de banda da matriz de coeficientes não é proporcional à redução do custo de execução na resolução de sistemas de equações lineares. Assim, não haveria necessidade de grande redução de largura de banda. Por outro lado, a redução da largura de banda em baixo custo de execução é relevante na aceleração da resolução de sistemas de equações lineares.

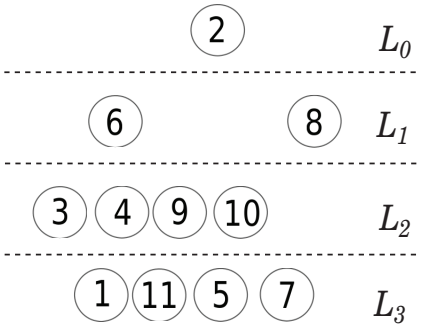

Figura 2.7: Rótulos dos vértices na estrutura de nível enraizada para o vértice com rótulo 2 do grafo mostrado na figura 2.5 [56].

Apenas a busca em largura (BFS), mostrada no algoritmo 2, na página 29 da seção 2.3, não fornece a redução na largura de banda necessária. Como visto na subseção 2.4.5, na página 35, o vértice inicial é relevante para a qualidade da numeração neste contexto. Por isso, o algoritmo George-Liu (mostrado no algoritmo 4, na página 38, na subseção 2.4.5) foi escolhido para fornecer o vértice inicial [48].

Como abordado na subseção 2.4.2 na página 32, George [37] mostrou que reverter a numeração apresenta resultados melhores. A heurística RBFS-GL é a junção desses conceitos. A heurística RBFS-GL retorna soluções em matrizes de grande porte com qualidade quase tão satisfatórias em relação ao método RCM, mas a heurística RBFS-GL apresenta cerca da metade do custo de execução do método RCM [62].

Na heurística RBFS, não é estabelecida a distância dos vértices do grafo em relação ao vértice inicial, tal como realizado nas linhas 4 e 12 do algoritmo 2, mostrado na página 29 da seção 2.3. Na heurística RBFS, $s(v)$ recebe $|V|$ em vez de 1 na linha 5 do algoritmo 2. Assim, a estrutura de repetição, mostrada nas linhas 9 a 20, opera enquanto $s(l) > 1$. Com isso, $s(u) \leftarrow s(l) - 1$ na linha 13 do algoritmo 2.

2.4.7 Heurística KP-band

Pode ocorrer que vértices w em um mesmo nível da estrutura de nível enraizada em vértice pseudoperiférico (fornecido pelo algoritmo George-Liu) tenham o mesmo grau. Vértice pseudoperiférico é explicado na definição 1.35, na página 14 da seção 1.3.

O método RCM não faz distinção na ordem dos vértices $w_1, w_2, \ldots, w_k$, com $Grau(G, w_1) = Grau(G, w_2) = \cdots = Grau(G, w_k)$ para vértices $w_j \in Adj(G, v)$, para $j \in [1, k]$. Koohestani e Poli [80] procuraram reduzir esses empates na ordenação dos vértices w adja-

centes a determinado vértice v. Os autores evoluíram por programação genética o método RCM, gerando a heurística KP-band, que ordena os vértices $w \in Adj(G, v)$ pela fórmula $0,179492928171 \cdot \varsigma(w)^3 + 0,292849834929 \cdot \varsigma(w)^2 - 0,208926175433 \cdot\ n - 0,736485142138 \cdot\ n \cdot \varsigma(w) - 1,77524579882 \cdot \varsigma(w) - 1,75681383404$, em que $n = |V|$ e $\varsigma(w) = \sum_{u \in Adj(G,w)} Grau(G, u)$. Assim, $\varsigma(w)$ é a soma dos graus dos vértices adjacentes ao vértice w. Claramente, $\varsigma(w)$ é o produto do grau do vértice w pela média dos graus dos vértices adjacentes ao vértice w: $\varsigma(w) = Grau(G, w) \cdot \frac{\sum_{u \in Adj(G,w)} Grau(G,u)}{Grau(G,w)}$. Com isso, $\varsigma(w)$ contém informação dos vértices a dois níveis do vértice v: $(\forall w \in Adj(G, v))\ u \in Adj(G, w)$.

2.4.8 Heurística RLK

A heurística *Reverse Level King* (RLK) foi projetada com conceitos dos métodos RCM, para redução de largura de banda, e King [77], para redução de perfil (*profile*) de matrizes. A heurística RLK ordena vértices $w \in Adj(G, v)$ em ordem crescente do número de adjacências de forma similar ao método RCM. Entretanto, diferentemente do método RCM, vértices já rotulados não são considerados na contagem do número de adjacências do vértice w.

Diferentemente da heurística de King, a heurística RLK rotula os vértices nível a nível da estrutura de nível enraizada do vértice pseudoperiférico fornecido pelo algoritmo George-Liu. Como mencionado, vértice pseudoperiférico é explicado na definição 1.35, na página 14 da seção 1.3.

2.5 Algoritmo GPS

O algoritmo GPS [40] foi desenvolvido após estudo detalhado do método RCM original [37]. Gibbs et al. [40] verificaram que o método RCM original [37] não era adequada em situações em que há um custo computacional excessivo para se encontrar o vértice inicial. Como explicado na subseção 2.4.5, nos métodos Cuthill-McKee e RCM originais, são encontrados vértices com grau mínimo do grafo, são geradas as estruturas de nível de todos esses vértices e são escolhidas as estruturas de nível que apresentem a menor largura de nível. O método RCM original [37] realizaria esforço desnecessário, por exemplo, quando muitos vértices do grafo possuíssem o grau mínimo. Isso implicaria na montagem da estrutura de nível enraizada por meio da busca em largura para cada um desses vértices e seriam realizadas as renumerações por todas essas estruturas de nível.

O algoritmo GPS [40] pode ser dividido em três passos. No primeiro passo, é encontrado um par de vértices iniciais. Esse passo é apresentado na subseção 2.5.1. No segundo passo, reduz-se a largura de nível da estrutura montada no passo anterior. Esse passo é descrito na subseção 2.5.2. Por último, realiza-se a renumeração dos vértices do grafo de uma forma similar ao método RCM, iniciando a numeração pelo vértice encontrado no primeiro passo. Esse passo é descrito na subseção 2.5.3.

2.5.1 Escolha dos vértices iniciais

Cuthill e Mckee [14] constataram que, quanto mais níveis uma estrutura de nível tem, menor será a largura de banda, se a renumeração for realizada por meio dessa estrutura de nível. Gibbs et al. [40] utilizaram esse conhecimento para a escolha por qual vértice inicializar o processamento da renumeração dos vértices.

Considere um grafo $G = (V, E)$ e um vértice $v \in V$ com grau mínimo do grafo. O algoritmo gera a estrutura de nível enraizada $\mathscr{L}(v)$. Em seguida, em ordem crescente de grau, o algoritmo gera estruturas de nível enraizadas em $w \in L_{\ell(v)}(v)$. Se $\ell(w) > \ell(v)$, então, atribui-se w a v e repete-se o processo até que não tenha vértice em $L_{\ell(v)}(v)$ com excentricidade maior que a excentricidade de v. O vértice v será um dos dois vértices iniciais do algoritmo. Esse algoritmo é mostrado na subseção 2.5.1.1.

Para obter o segundo vértice inicial u, entre os vértices de $L_{\ell(v)}(v)$, escolhe-se o vértice que apresentar $\mathscr{L}(u)$ com a menor largura de nível. Esse algoritmo é mostrado na subseção 2.5.1.2. Um exemplo é mostrado na subseção 2.5.1.3.

2.5.1.1 Primeiro vértice inicial

Mostra-se, no algoritmo 6, um pseudocódigo para a escolha do primeiro vértice inicial. O vértice com grau mínimo do grafo é atribuído a v na linha 2. Em seguida, é montada a estrutura de nível enraizada $\mathscr{L}(v)$, na linha 3. A rotina na linha 3 coloca o vértice v no nível $L_0(v)$ da estrutura, os vértices adjacentes ao vértice v são colocados no nível $L_1(v)$ e assim por diante, até montar o último nível $L_{\ell(v)}(v)$. Essa rotina pode ser implementada de forma similar à busca em largura, mostrada no algoritmo 2, na página 29 da seção 2.3.

Na estrutura de repetição mostrada nas linhas 4 a 14, o algoritmo verifica os vértices $w \in L_{\ell(v)}(v)$ em ordem crescente de grau (veja a linha 5 e a linha 7 na estrutura de repetição mostrada nas linhas 6 a 13). As estruturas de nível enraizadas em vértices $w \in L_{\ell(v)}(v)$ são construídas, na linha 8, até que algum vértice w tenha excentricidade

Algoritmo 6: VerticePseudoPeriférico_GPS (escolha do primeiro vértice inicial para o algoritmo GPS).

```
   Entrada: grafo G = (V, E);
   Saída: vértice pseudoperiférico v ∈ V;
1  início
2  |  v ← VérticeGrauMínimo(G, V); // algoritmo 5
   |  // constrói-se estrutura de nível enraizada
3  |  ℒ(v) ← EstruturaDeNívelEnraizada(v);
4  |  repita
   |  |  // constrói fila de prioridades em ordem crescente
   |  |  // de grau dos vértices em L_ℓ(v)(v)
5  |  |  F ← FilaPrioridades(L_ℓ(v)(v));
6  |  |  repita
7  |  |  |  w ← F.Remove();
   |  |  |  // constrói-se estrutura de nível enraizada
8  |  |  |  ℒ(w) ← EstruturaDeNívelEnraizada(w);
   |  |  |  // ℓ(w) e ℓ(v) são as excentricidades dos vértices
9  |  |  |  se ( ℓ(w) > ℓ(v) ) então
10 |  |  |  |  v ← w;
11 |  |  |  |  ℒ(v) ← ℒ(w);
12 |  |  |  fim-se;
13 |  |  até que ((v = w) ∨ (F = ∅)) ;
14 |  até que (v = w) ;
15 |  retorna v;
16 fim.
```

maior que a excentricidade de v. Então, atribui-se w a v na linha 10 e repete-se o processo. Itera-se na estrutura de repetição nas linhas 4 a 14 até que algum vértice v tenha excentricidade igual ou maior que todos os vértices em $L_{\ell(v)}(v)$. Ao sair da estrutura de repetição *repita* externa, o vértice v é um dos vértices iniciais do algoritmo GPS, retornado na linha 15.

2.5.1.2 Segundo vértice inicial

Para determinar o segundo vértice periférico, utiliza-se a sub-rotina *VerticeMenorLarguraDeNivel*, mostrada no algoritmo 7. A sub-rotina *VerticeMenorLarguraDeNivel*, mostrada no algoritmo 7, recebe o conjunto de vértices $w \in L_{\ell(v)}(v)$, que são percorridos na estrutura de repetição *para cada*, mostrada nas linhas 3 a 9. A sub-rotina que determina a largura de nível de $\mathscr{L}(w)$, na estrutura condicional nas linhas 5 a 8, é mostrada no algoritmo 8.

Algoritmo 7: *VérticeMenorLarguraDeNível* (escolhe vértice com menor largura de nível).

Entrada: conjunto composto pelos vértices $w \in L_{\ell(v)}(v)$;
Saída: segundo vértice pseudoperiférico $u \in V$ para o algoritmo GPS;

```
1  início
2  |  largura ← +∞;
3  |  para cada ( vértice w ∈ L_ℓ(v)(v) ) faça
   |  |  // constrói-se estrutura de nível enraizada
4  |  |  ℒ(w) ← EstruturaDeNívelEnraizada(w);
   |  |  // a sub-rotina b é mostrada no algoritmo 8
5  |  |  se ( b(ℒ(w)) < largura ) então
6  |  |  |  largura ← b(ℒ(w));
7  |  |  |  u ← w;
8  |  |  fim-se;
9  |  fim-para-cada;
10 |  retorna u;
11 fim.
```

Algoritmo 8: *b*.

Entrada: estrutura de nível $\mathscr{L}(v)$;
Saída: largura de nível de $\mathscr{L}(v)$;

```
1  início
   |  // a largura de nível de ℒ(v) é, pelo menos, 1,
2  |  l ← 1; // porque o número de vértices no nível 0 é 1
   |  // são percorridos os níveis de ℒ(v)
3  |  para ( i ← 1; i ≤ ℓ(v); i ← i + 1 ) faça  se ( |L_i(v)| > l ) então
   |  l ← |L_i(v)|;
4  |  retorna l;
5  fim.
```

No algoritmo 7, ao sair da estrutura de repetição *para cada*, estará armazenado em u o vértice de $L_{\ell(v)}(v)$ com a menor largura de nível, que é retornado na linha 10.

2.5.1.3 Exemplo

Na figura 2.8, mostra-se um grafo e a sua representação matricial com os vértices iniciais determinados pelo algoritmo GPS. O vértice v com rótulo 1 foi determinado como um vértice inicial porque possui grau três, que é um vértice com grau mínimo do grafo (além de outros vértices com grau três).

Pela montagem da estrutura de nível enraizada do vértice v

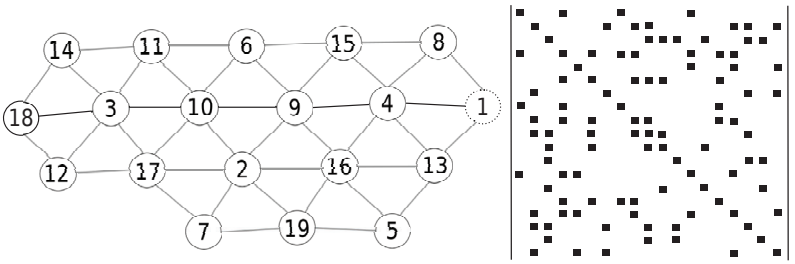

Figura 2.8: Um grafo de exemplo para a renumeração pelo algoritmo GPS com os vértices iniciais com rótulos 1 e 18 destacados e a representação matricial M_G do grafo [56].

com rótulo 1, obtém-se a excentricidade $\ell(v) = 5$ e os vértices em $L_{\ell(v)}(v) = \{s^{-1}(12), s^{-1}(14), s^{-1}(18)\}$. Como $\ell(s^{-1}(12)) = \ell(s^{-1}(14)) = \ell(s^{-1}(18)) = 5$, então, um dos vértices iniciais é o vértice com rótulo 1. Calculando-se a largura de nível dos vértices em $L_{\ell(v)}(v)$, encontra-se $b(\mathscr{L}(s^{-1}(14))) = b(\mathscr{L}(s^{-1}(18))) = 4$ e $b(\mathscr{L}(s^{-1}(12))) = 5$. Com isso, escolheu-se o vértice com rótulo 18 como o segundo vértice inicial do algoritmo GPS.

2.5.2 Redução da largura de nível

É explicado o segundo passo do algoritmo GPS na subseção 2.5.2.1. Um exemplo do passo é mostrado na subseção 2.5.2.2.

2.5.2.1 Segundo passo do algoritmo GPS

No segundo passo do algoritmo GPS, cria-se uma nova estrutura de nível a partir das estruturas de nível enraizadas dos vértices iniciais v e u. A largura de nível da nova estrutura de nível será, no máximo, a menor largura de nível entre as estruturas de nível enraizadas em v e u.

Seja $k = \ell(v) = \ell(u)$. A excentricidade de v é igual às excentricidades dos vértices em $L_{\ell(v)}(v)$, ou seja, $(\forall u \in L_{\ell(v)}(v))\ \ell(v) = \ell(u)$, pois $v \in L_{\ell(u)}(u)$. Associa-se, a cada vértice w do grafo $G = (V, E)$, um par ordenado (i, j), chamado de par associado do nível, em que i é o índice associado ao nível de $\mathscr{L}(v)$ que contém w, e j é o índice associado, em ordem decrescente, de $\mathscr{L}(u)$ que contém w. O índice do nível em $\mathscr{L}(u)$ que contém w é $k - j$. O par (i, j) é associado ao vértice w se e, somente se, $w \in L_i(v) \cap L_{k-j}(u)$. Dessa forma, os pares associados aos vértices v e u são $(0, 0)$ e (k, k), respectivamente. Os passos do processo que resultarão em uma nova estrutura de ní-

vel $\mathscr{K}(\nu,\dots) = \{K_0(\nu,\dots), K_1(\nu,\dots),\dots,K_k(\nu,\dots)\}$ são descritos a seguir.

1. A cada par na forma (i,i), associado a algum vértice w, w é colocado em $K_i(\nu,\dots)$. O par $(0,0)$ é associado a v, então, $v \in K_0(\nu,\dots)$; e o par (k,k) é associado a u, então, $u \in K_{\ell(\nu)}(\nu,\dots)$. Cada vértice w e as arestas incidentes a w são removidas de $G = (V,E)$. Se $V = \varnothing$, então, pare.

2. Após o primeiro passo, tem-se $V = \bigcup_{\iota=1}^{t} \mathscr{C}_\iota$, em que os conjuntos $\mathscr{C}_\iota$ são disjuntos e t é o número de componentes. As componentes são dispostas em ordem decrescente de número de vértices, isto é, $|\mathscr{C}_1| \geq |\mathscr{C}_2| \geq \cdots \geq |\mathscr{C}_t|$.

3. Para cada $\mathscr{C}_\iota$, $\iota = 1,2,\dots,t$ e considerando-se $i = 0,1,\dots,k$; e $j = k,k-1,\dots,0$, faça:

   - calcule $(n_0,n_1,\dots,n_k)$, em que n_r é o número de vértices em $K_r(\nu,\dots)$, para $0 \leq r \leq k$;
   - calcule $(h_0,h_1,\dots,h_k)$, em que h_r é n_r mais o número de vértices que $K_r(\nu,\dots)$ teria se o índice i do par associado do nível fosse utilizado como a renumeração de nível;
   - calcule $(l_0,l_1,\dots,l_k)$, em que l_r é n_r mais o número de vértices que $K_r(\nu,\dots)$ teria se o índice j do par associado do nível fosse utilizado como a renumeração de nível;
   - encontre $h_{max} = \max_{0\leq r\leq k}[h_r : h_r - n_r > 0]$, $l_{max} = \max_{0\leq r\leq k}[l_r : l_r - n_r > 0]$ e
     - se $h_{max} < l_{max}$, então, numere o nível de cada vértice de $\mathscr{C}_\iota$ de acordo com o índice i do par associado do nível;
     - se $h_{max} > l_{max}$, então, numere o nível de cada vértice de $\mathscr{C}_\iota$ de acordo com o índice j do par associado do nível;
     - se $h_{max} = l_{max}$, então:
       * se $b(\mathscr{L}(u)) \geq b(\mathscr{L}(v))$, então, numere o nível de cada vértice de $\mathscr{C}_\iota$ de acordo com o índice i do par associado do nível;
       * se $b(\mathscr{L}(u)) < b(\mathscr{L}(v))$, então, numere o nível de cada vértice de $\mathscr{C}_\iota$ de acordo com o índice j do par associado do nível.

São utilizados os níveis resultantes da combinação das estruturas de nível enraizadas dos vértices v e u. Para isso, verifica-se qual quantidade é menor, h_{max} ou l_{max}.

2.5.2.2 Exemplo

Um exemplo da execução da segunda parte do algoritmo GPS é mostrado nas figuras 2.9 a 2.12. Neste exemplo, são considerados os vértices iniciais v, com rótulo 1, e u, com rótulo 18, do grafo mostrado na figura 2.8. Na figura 2.9, associados aos vértices, são mostrados os índices (i, j) referentes aos níveis das estruturas de nível dos vértices com rótulos 1 e 18.

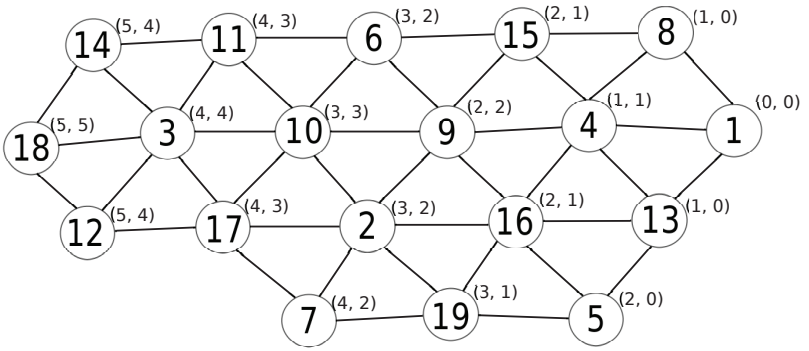

Figura 2.9: Grafo com os vértices rotulados e associados aos índices das estruturas de nível enraizadas nos vértices v, com rótulo 1, e u, com rótulo 18. O primeiro índice do par (i, j) indica o nível crescente do vértice na estrutura de nível enraizada no vértice v e o segundo índice indica o nível decrescente do vértice na estrutura de nível enraizada no vértice u [56].

Na figura 2.10, são mostrados os índices dos vértices com rótulos 1, 3, 4, 9, 10, e 18 na nova estrutura de nível $\mathscr{K}(\nu, \dots)$. Esses vértices são dispostos na estrutura de nível $\mathscr{K}(s^{-1}(1), \dots)$ porque $i = j$ em (i, j). Também, na figura 2.10, são mostrados os componentes $\mathscr{C}_1$, composto pelos vértices com rótulos 2, 5, 7, 12, 13, 16, 17 e 19, e $\mathscr{C}_2$, composto pelos vértices com rótulos 6, 8, 11, 14 e 15. Na figura 2.10, os vértices em $\mathscr{C}_1$ e $\mathscr{C}_2$ não possuem índices no estágio corrente do algoritmo.

Em seguida, gera-se $(n_0, n_1, n_2, n_3, n_4, n_5) = (1, 1, 1, 1, 1, 1)$ porque tem um vértice em cada nível de $\mathscr{K}(s^{-1}(1), \dots)$, ou seja, cada um dos vértices com rótulos 1, 4, 9, 10, 3 e 18 está em um nível de $\mathscr{K}(s^{-1}(1), \dots)$ (veja os índices associados a esses vértices na figura 2.10). Com isso, de $\mathscr{C}_1$, são gerados:

- $(h_0, h_1, h_2, h_3, h_4, h_5) = (1, 1, 1, 1, 1, 1) + (0, 1, 2, 2, 2, 1) = (1, 2, 3, 3, 3, 2)$, pois, para $0 \leq r \leq k$, h_r é n_r adicionado do número

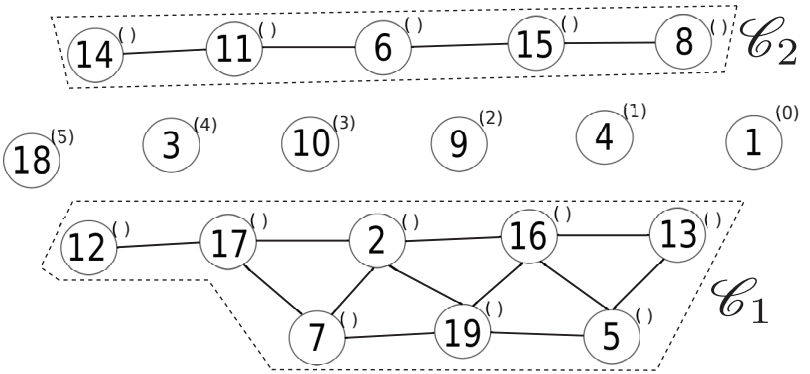

Figura 2.10: Os vértices que não estão em $\mathscr{C}_1$ ou $\mathscr{C}_2$ estão nos níveis $K_0(\nu,\dots), K_1(\nu,\dots),\dots,K_k(\nu,\dots)$, em que $k = \ell(s^{-1}(1)) = \ell(s^{-1}(18))$. Os índices estão mostrados juntos desses vértices rotulados. O grafo corrente é $G = \mathscr{C}_1 \cup \mathscr{C}_2$ [56].

de vértices de $\mathscr{C}_1$, no nível r, considerando-se o índice i (veja os índices i associados aos vértices em $\mathscr{C}_1$ na figura 2.9);

- $(l_0, l_1, l_2, l_3, l_4, l_5) = (1,\ 1,\ 1,\ 1,\ 1,\ 1) + (2,\ 2,\ 2,\ 1,\ 1,\ 0) = (3,\ 3,\ 3,\ 2,\ 2,\ 1)$, pois, para $0 \leq r \leq k$, l_r é n_r adicionado do número de vértices de $\mathscr{C}_1$, no nível r, considerando-se o índice j (veja os índices j associados aos vértices em $\mathscr{C}_1$ na figura 2.9).

Com isso, $h_{max} = l_{max} = 3$ e $b(\mathscr{L}(u)) = b(\mathscr{L}(v))$. Então, os vértices de $\mathscr{C}_1$ têm os seus índices da estrutura de nível organizados e incorporados ao grafo de acordo com o índice i, como pode ser visto na figura 2.11.

Após associar os vértices de $\mathscr{C}_1$ na nova estrutura de nível, os passos são repetidos para $\mathscr{C}_2$. Nesta etapa do algoritmo, tem-se $(n_0, n_1, n_2, n_3, n_4, n_5) = (1,\ 2,\ 3,\ 3,\ 3,\ 2)$, pois $\mathscr{C}_1$ foi incorporado à estrutura de nível $\mathscr{K}(\nu,\dots)$, de acordo com o índice i do par associado do nível. Por isso, de $\mathscr{C}_2$, são gerados:

- $(h_0, h_1, h_2, h_3, h_4, h_5) = (1,\ 2,\ 3,\ 3,\ 3,\ 2) + (0,\ 1,\ 1,\ 1,\ 1,\ 1) = (1,\ 3,\ 4,\ 4,\ 4,\ 3)$, pois, para $0 \leq r \leq k$, h_r é n_r adicionado do número de vértices de $\mathscr{C}_2$, no nível r, considerando-se o índice i (veja os índices i associados aos vértices em $\mathscr{C}_2$ na figura 2.9);

- $(l_0, l_1, l_2, l_3, l_4, l_5) = (1,\ 2,\ 3,\ 3,\ 3,\ 2) + (1,\ 1,\ 1,\ 1,\ 1,\ 0) = (2,\ 3,\ 4,\ 4,\ 4,\ 2)$, pois, para $0 \leq r \leq k$, h_r é n_r adicionado do número de vértices de $\mathscr{C}_2$, no nível r, considerando-se o índice j (veja os índices j associados aos vértices em $\mathscr{C}_2$ na figura 2.9.

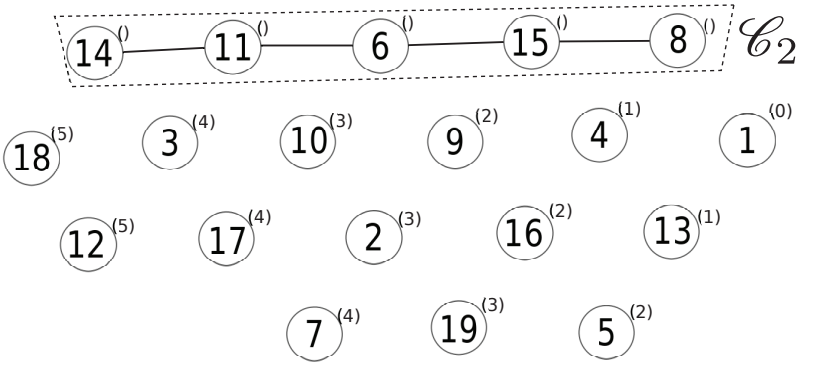

Figura 2.11: Os vértices de $\mathscr{C}_1$, mostrados na figura 2.10, foram incorporados na nova estrutura de nível de acordo com o índice i de (i, j), mostrados na figura 2.9 [56].

Com isso, $h_{max} = l_{max} = 4$ e $b(\mathscr{L}(u)) = b(\mathscr{L}(v))$. Então, os vértices de $\mathscr{C}_2$ têm os seus índices da estrutura de nível organizados de acordo com o índice i, como pode ser visto na figura 2.12.

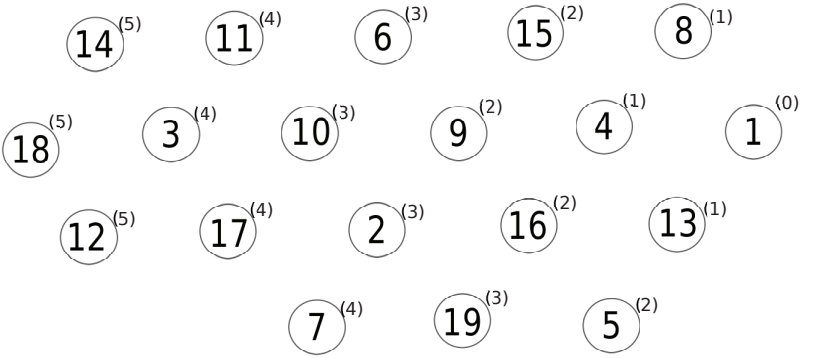

Figura 2.12: Os vértices de $\mathscr{C}_2$, mostrado na figura 2.10, foram incorporados na nova estrutura de nível de acordo com o índice i de (i, j), mostrados na figura 2.9 [56].

Com essa estrutura de nível, é obtida a seguinte ordenação dos vértices com rótulos:

- 1, porque está em $K_0(\nu, \dots)$;
- 8, 13 e 4, porque estão em $K_1(\nu, \dots)$ e possuem graus 3, 4 e 6, respectivamente;

- 5, 15, 9 e 16, porque estão em $K_2(\nu, \dots)$ e possuem graus 3, 4, 6 e 6, respectivamente;
- $\dots$;
- 12, 18 e 14, porque estão em $K_5(\nu, \dots)$ e possuem o mesmo grau e, com isso, a ordem entre esses vértices é arbitrária.

2.5.3 Renumeração

A ordem para percorrer os vértices é dada considerando-se a estrutura de nível obtida no passo mostrado na subseção 2.5.2. Realiza-se a renumeração dos vértices iniciando-se pelo nível $K_k(\nu, \dots)$ até o nível $K_0(\nu, \dots)$. Para vértices de mesmo nível, primeiramente, percorre-se o que tem grau maior.

No exemplo mostrado nas figuras 2.8 a 2.12, a renumeração é dada na ordem reversa ao que foi apresentado no final da subseção 2.5.2. A renumeração final do grafo da figura 2.8 e a sua representação matricial são mostradas na figura 2.13.

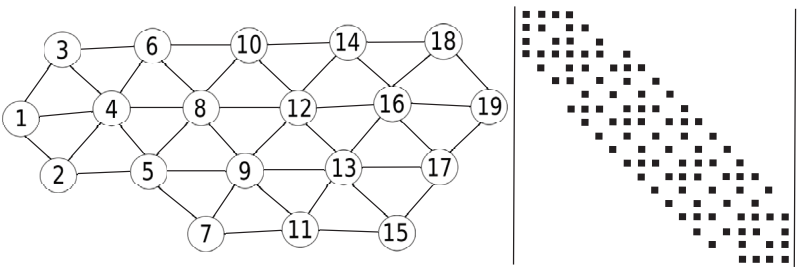

Figura 2.13: Grafo da figura 2.8 renumerado pelo algoritmo GPS e sua representação matricial M_{G_1} [56].

Para uma comparação, a renumeração do grafo da figura 2.8 pelo método RCM é mostrada na figura 2.14, considerando-se o vértice com rótulo 1 como o vértice inicial. Na tabela 2.3, observa-se que resultados satisfatórios (reduções de largura de banda e de *profile* em relação à matriz M_G mostrada na figura 2.8) são obtidos pelo método RCM ao se escolher um vértice inicial com excentricidade aproximada ao diâmetro do grafo. No caso da figura 2.8, $\ell(s^{-1}(1)) = 5$, que é o diâmetro do grafo.

Na tabela 2.3, também são mostradas as larguras de banda e os *profiles* de M_{G_1} e de M_{G_2} das figuras 2.13 e 2.14, obtidos pelas renumerações do grafo mostrado na figura 2.8 por meio dos algoritmos GPS e

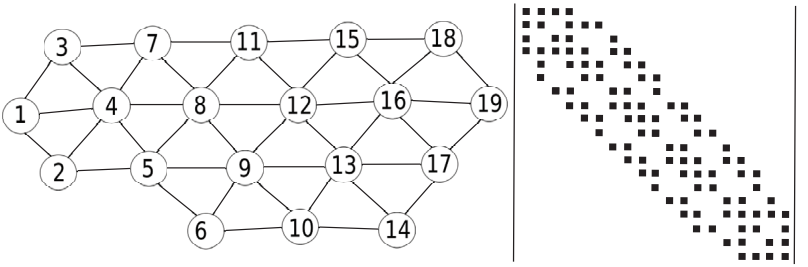

Figura 2.14: Grafo da figura 2.8 renumerado pelo método RCM e sua representação matricial M_{G_2} [56].

Matriz	Largura de banda	$Profile$
M_G (original) da figura 2.8	17	142
M_{G_1} (GPS) da figura 2.13	4	61
M_{G_2} (RCM) da figura 2.14	4	63

Tabela 2.3: Largura de banda e de perfil (*profile*) das matrizes mostradas nas figuras 2.8, 2.13 e 2.14 [56].

RCM, respectivamente. Em particular, o custo de execução do método RCM é menor que o custo de execução do algoritmo GPS [63, 62].

Na *Netlib* [22], são disponibilizadas implementações na linguagem Fortran do algoritmo GPS[4] [40] e da heurística Gibbs-King[5] [38]. Lewis [83] também implementou o algoritmo GPS na linguagem Fortran.

2.6 Exemplos de tempos de execução dos métodos heurísticos

Os métodos heurísticos baseados em conceitos da teoria dos grafos apresentados neste texto são de baixo custo de execução. A seguir, são mostrados quatro exemplos de testes executados em estações de trabalho comuns [63, 62].

1. Em um grafo composto de 3,5 milhões de vértices e 95 milhões de arestas (matriz *nlpkkt120*), os custos de execução seriais dos métodos foram [62]:

   - 12 segundos para o método RCM,
   - 11 segundos para o método KP-band,

[4] *http://www.netlib.org/toms/508.* [13]
[5] *http://www.netlib.org/toms/509* [38], *http://www.netlib.org/toms/582* [83].

- 7 segundos para o método RBFS-GL,
- 12 segundos para o método resultante da hiper-heurística ACHH (apresentado no capítulo 3).

2. Em um grafo composto de 1,6 milhão de vértices e 114 milhões de arestas (matriz *Flan_1565*), os custos de execução seriais dos métodos foram de nove segundos para o método KP-band e para o método resultante da hiper-heurística ACHH [63]. O resolutor do sistema de equações lineares, sem e com pré-processamento da matriz pelo método resultante da hiper-heurística ACHH, executou em 89.293 e 88.421 segundos, respectivamente, resultando em aceleração de 1%. O resultado do resolutor foi um pouco pior com a matriz pré-processada pela heurística KP-band e pior pelo método RCM.

3. Em um grafo composto de 1,5 milhão de vértices e 21 milhões de arestas (matriz *StocF_1465*), os custos de execução seriais dos métodos foram de quatro segundos para o método KP-band e para o método resultante da hiper-heurística ACHH, e de dois segundos para a heurística RBFS-GL [63]. O resolutor do sistema de equações lineares, sem e com pré-processamento da matriz pelo método resultante da hiper-heurística ACHH, executou em 45.570 e 44.430 segundos, respectivamente, resultando em aceleração de 3%. O resultado do resolutor foi pior com a matriz pré-processada pelos demais métodos heurísticos para redução de largura de banda.

4. Em um grafo composto de 1,4 milhão de vértices e 60 milhões de arestas (matriz *Geo_1438*), os custos de execução seriais dos métodos foram [63]: seis segundos para o método RCM, 30 segundos para o método KP-band, e sete segundos para o método resultante da hiper-heurística ACHH. O resolutor do sistema de equações lineares, sem e com pré-processamento da matriz pelo método resultante da hiper-heurística ACHH, executou em 76.052 e 57.104 segundos, respectivamente, resultando em aceleração de 33%. Não houve aceleração do resolutor com o método RCM e houve desaceleração do resolutor com os demais métodos heurísticos para redução de largura de banda avaliados.

Como comparação, 500 segundos foi o critério de parada estabelecido para a heurística VNS-band [94] (apresentada na seção 4.3), para fornecer soluções em grafos com aproximadamente 1.000 vértices e menos de 4.000 arestas (a maior matriz utilizada nos testes dos autores foi a matriz *sherman4*, com dimensão de 1.104 e composta de 3.786 coeficientes não nulos). A heurística DRSA (apresentada na seção 4.4) apresentou resultados melhores com tempos de execução cerca de cinco

vezes maiores que a heurística VNS-band, para encontrar as melhores soluções. Gonzaga de Oliveira e Silva [62] mostraram que os métodos RCM, KP-band e RBFS-GL superaram a heurística DRSA ao serem aplicados a grafos maiores que 2.000 vértices[6]. Pode-se considerar que as heurísticas VNS-band e DRSA são impraticáveis de serem aplicadas em matrizes de grande porte, isto é, com dimensão maior que um milhão.

[6] A implementação da heurística DRSA utilizada no estudo está disponível em https://github.com/Pigzaum/drsa_bmp.

Capítulo 3

Hiper-heurística baseada em colônia de formigas

3.1 Introdução

Basicamente, uma hiper-heurística é um método heurístico que tem um ou mais dos seguintes objetivos:

- no contexto de computação evolucionária, gera novos métodos heurísticos pela evolução de métodos heurísticos mais simples;
- seleciona o melhor entre vários métodos heurísticos para resolver eficientemente determinado problema computacional;
- combina métodos heurísticos para gerar um novo método heurístico;
- adapta métodos heurísticos para resolver eficientemente problemas computacionais.

Neste capítulo, é mostrada uma hiper-heurística baseada na metaheurística otimização por colônia de formigas. Essa hiper-heurística evolui três métodos heurísticos baseados em teoria dos grafos: RCM, KP-band e RLK. O método RCM é apresentado na subseção 2.4.2, na página 32. As heurísticas KP-band e RLK são apresentadas respectivamente nas seções 2.4.7 e 2.4.8, na página 41.

Além desses três métodos, a hiper-heurística também pode selecionar a heurística RBFS-GL, mostrada na subseção 2.4.6, na página 40. Assim, a hiper-heurística seleciona ou gera métodos heurísticos baseados em conceitos da teoria dos grafos. Os métodos heurísticos resultantes da hiper-heurística são tão rápidos quanto os métodos rápidos utilizados como base da evolução.

Os métodos RCM, KP-band, RLK e RBFS-GL são métodos de propósito geral, ou seja, podem ser aplicados a quaisquer matrizes. Os métodos heurísticos resultantes da hiper-heurística ACHH são métodos especialistas, ou seja, são evoluídos para serem aplicados a matrizes oriundas de determinada área de aplicação. Com isso, os métodos heurísticos resultantes da hiper-heurística forneceram resultados melhores que os métodos heurísticos anteriores, tanto em redução de largura de banda de matrizes [62], quanto no pré-processamento de matrizes para aceleração de resolutores de sistemas de equações lineares [63].

Como a hiper-heurística *Ant Colony Hyper-Heuristic* (ACHH) foi projetada pela meta-heurística otimização por colônia de formigas, essa meta-heurística é abordada na seção 3.2. A hiper-heurística ACHH é apresentada na seção 3.3.

3.2 Meta-heurística otimização por colônia de formigas

Essa meta-heurística é uma técnica probabilística para a solução de problemas computacionais, principalmente problemas de encontrar caminhos satisfatórios em grafos. Basicamente, formigas minimizam a distância entre o formigueiro e a fonte de alimento. Um conjunto de formigas artificiais percorre os caminhos de um grafo, que representa a trilha percorrida pela formiga até o formigueiro. Além de mesclar informações determinísticas e estocásticas, esta meta-heurística também apresenta uma forma de soluções trocar informações.

Uma função de probabilidade é utilizada para determinar o caminho a ser percorrido pela formiga. Durante o percurso, as formigas depositam feromônio no caminho percorrido. Nesta analogia, uma ou mais funções objetivos são utilizadas para determinar a quantidade de feromônio depositada no caminho percorrido pela formiga para levar "comida" para o formigueiro. Assim, a meta-heurística é baseada na comunicação via feromônio depositado pelas formigas no caminho percorrido. Uma taxa de evaporação de feromônio é utilizada para evitar convergência muito rápida.

O critério de parada do processo pode ser o tempo de execução ou um número pré-definido de iterações. Ao final do processo, o caminho com maior quantidade de feromônio é retornado, que fornece uma solução aproximada à solução ótima do problema.

3.3 Hiper-heurística ACHH

Como mencionado, a hiper-heurística ACHH evolui os métodos RCM, descrito na subseção 2.4.2, na página 32, e as heurísticas KP-band [80] e RLK [62], descritas respectivamente nas seções 2.4.7 e 2.4.8, na página 41. Assim, a hiper-heurística ACHH evolui esses três métodos heurísticos e, consequentemente, gera um novo método heurístico para redução de largura de banda de matrizes para a área de aplicação específica. A hiper-heurística ACHH também pode selecionar um entre quatro (RCM [37], KP-band [80], RBFS-GL [54] e RLK [63]) métodos heurísticos para a área de aplicação do problema.

O processo de localizar o vértice pseudoperiferico para iniciar o processamento da numeração é descrito na subseção 3.3.1. O grafo de componentes utilizado na hiper-heurística ACHH é apresentado na subseção 3.3.2. As fórmulas de prioridade dos métodos heurísticos são descritas na subseção 3.3.3. A estrutura da hiper-heurística ACHH é mostrada na subseção 3.3.4. Finalmente, o pseudocódigo da hiper-heurística ACHH é mostrado na subseção 3.3.5.

3.3.1 Vértice pseudoperiférico

Os métodos RCM, KP-band, RBFS e RLK são variações da busca em largura, mostrada no algoritmo 2, na página 29 da seção 2.3. As quatro heurísticas diferem na fórmula de prioridade utilizada para ordenar os vértices adjacentes a determinado vértice.

As quatro heurísticas pertencem a uma classe de algoritmos conhecidos como reordenações por nível a partir de um vértice inicial, em que o algoritmo rotula os vértices de um grafo ao particioná-los em conjuntos de vértices a partir da distância de um vértice inicial. Portanto, o vértice inicial do processamento da numeração contribui fortemente para a qualidade da ordenação resultante. Em geral, essas heurísticas fornecem resultados melhores quando a largura de nível (veja definições 1.36 e 1.37 na página 14, na seção 1.3) da estrutura de nível enraizada no vértice inicial é estreita e a excentricidade do vértice inicial é próxima ao diâmetro do grafo.

A hiper-heurística ACHH foi projetada para avaliar diferentes algoritmos para encontrar o vértice inicial do processamento em conjunto com algoritmos de reordenação de vértices do grafo. O sistema ACHH avaliou dois desses algoritmos [62] e o algoritmo George-Liu, mostrado na subseção 2.4.5, na página 34, apresentou os melhores resultados em todos os casos avaliados. Assim, neste texto, para todos os métodos heurísticos para redução de largura de banda, será considerado que o vértice inicial do processamento será fornecido pelo algoritmo George-Liu. Ainda, a ordem fornecida pela heurística resultante é revertida.

3.3.2 Grafo de componentes

A hiper-heurística inicializa um grafo $C = (V_C, E_C)$ que contém componentes dos métodos heurísticos para redução de largura de banda. Para evitar ambiguidade com vértices do grafo $\mathscr{G}$ de entrada, descreve-se, neste texto, como nodos os vértices do grafo de componentes $C = (V_C, E_C)$. O grafo $C = (V_C, E_C)$ é particionado em quatro partições de nodos:

- na partição 0, um nodo inicial i,
- na partição 1, um ou mais nodos, de forma que cada um representa um algoritmo que retorna um vértice pseudoperiférico,
- na partição 2, nodos f_j, de forma que cada um representa um algoritmo de reordenação utilizado para rotular os vértices do grafo de entrada $G = (V, E)$,
- na partição 3, nodos que representam valores limites.

Na analogia com colônia de formigas, o nodo i, na partição 0, é de onde a formiga parte com a "comida" (grafos, ou matrizes) e os nodos na partição 3 localizam-se no formigueiro. Assim, cada formiga transporta cada grafo, um de cada vez, pelo caminho no grafo $C = (V_C, E_C)$, do nodo i, na partição 0, até o nodo na partição 3.

A heurística RBFS é mais rápida que os métodos RCM e KP-band porque não ordena os vértices adjacentes a um vértice v. O método RCM é mais rápido que a heurística KP-band quando realiza poucas trocas para ordenar os vértices adjacentes a um vértice v. Assim, um algoritmo de reordenação de baixo custo de execução realiza poucas trocas ao ordenar as adjacências de um vértice. A hiper-heurística procura gerar heurísticas de baixo custo de execução com número pequeno de trocas nessa ordenação. Dessa forma, o grafo de componentes $C = (V_C, E_C)$ na hiper-heurística foi projetado com nodos de valores limites para informar um limiar quando o valor da fórmula de prioridade não é relevante. Entretanto, a hiper-heurística ACHH não gerou nenhum valor limite em todos os métodos heurísticos resultantes [63, 62]. Por isso, por simplicidade nesta apresentação, os nodos de valores limites na partição 3 não serão mais considerados neste texto e essa partição somente conterá o nodo terminal $\mathfrak{t}$.

Como mencionado na subseção 3.3.1, a hiper-heurística ACHH selecionou o algoritmo George-Liu para fornecer o vértice pseudoperiférico para o método heurístico resultante em todos os casos [63, 62]. Por isso, por simplicidade nesta apresentação, será considerado neste texto que há somente um nodo na partição 1 do grafo de componentes $C = (V_C, E_C)$.

Na fase de calibração dos parâmetros, Gonzaga de Oliveira e Silva [63, 62] determinaram que haveria seis nodos na partição 2 do grafo de componentes $C = (V_C, E_C)$. Dessa forma, o grafo de componentes $C = (V_C, E_C)$ da hiper-heurística ACHH é apresentado neste texto com $|V_C| = 9$ nodos:

- um nodo inicial i na partição 0,
- um nodo p na partição 1, que representa o algoritmo George-Liu para retornar um vértice pseudoperiférico,
- seis nodos f_j na partição 2, de forma que cada um representa um algoritmo de reordenação utilizado para rotular os vértices do grafo de entrada $G = (V, E)$,
- um nodo terminal $\mathfrak{t}$ na partição 3,

Dessa forma, $V_C = \{i, p, f_j, \mathfrak{t}\}$, para $j \in [1, 6]$ e $|E_C| = 13$, pois $E_C = \{\{i, p\}, \{p, f_j\}, \{f_j, \mathfrak{t}\}\}$. Como mencionado, na analogia com colônia de formigas, o nodo i, na partição 0, é de onde a formiga parte com a "comida" (grafos, ou matrizes) e o nodo $\mathfrak{t}$, na partição 3, representa o formigueiro. Assim, cada formiga percorre um caminho no grafo $C = (V_C, E_C)$ transportando cada grafo, do nodo i, na partição 0, até o nodo $\mathfrak{t}$, na partição 3.

3.3.3 Fórmulas de prioridades

Na primeira iteração, a hiper-heurística começa com as fórmulas de prioridade dos métodos RCM, KP-band e RLK em nodos f_j, na partição 2 do grafo de componentes $C = (V_C, E_C)$. A hiper-heurística pode evoluir ou selecionar as fórmulas de prioridade desses três métodos heurísticos.

Como descrito na subseção 2.4.2, na página 32, o método RCM ordena vértices $w \in Adj(G, v)$ em ordem crescente do grau do vértice w. As heurísticas KP-band e RLK, mostradas respectivamente nas seções 2.4.7 e 2.4.8, na página 41, são variações do método RCM.

O sistema ACHH utiliza uma árvore binária para manter as fórmulas de prioridade. Nodos folhas nessa árvore contêm uma função, enquanto nodos internos contêm uma operação: adição, subtração ou multiplicação. A hiper-heurística gera uma função em relação ao valor n em todas as fórmulas de prioridade. O sistema produz aleatoriamente três tipos de fórmulas de prioridade, associadas com n e:

- $Grau(G, w)$, relacionado ao método RCM, mostrado na subseção 2.4.2, na página 32,
- ς, relacionado à heurística KP-band, mostrada na subseção 2.4.7, na página 41,

- $GrauN(G_S, w) = |\{u \in V : \{u, v\} \in E \wedge s(u) = \emptyset\}|$, considerando somente adjacências para vértices que ainda não foram rotulados (aqui, $s(u) = \emptyset$ representa um vértice u ainda não rotulado e considera-se que os vértices foram inicializados ($v \in V$) $s(v) = \emptyset$ no início do processamento), relacionado à heurística RLK, mostrada na subseção 2.4.8, na página 42.

A hiper-heurística gera aleatoriamente as fórmulas. Consequentemente, a hiper-heurística pode produzir uma fórmula de prioridade sem operadores, por exemplo. Nesse caso, a fórmula de prioridade é associada à heurística RBFS. A hiper-heurística também pode selecionar os próprios métodos RCM, KP-band e RLK como o melhor algoritmo para a classe de matrizes utilizadas na fase de treinamento. A hiper-heurística seleciona a heurística RBFS-GL quando a fórmula de prioridade gerada é uma constante real ou gera uma fórmula de prioridade que envolve:

- n e o grau do vértice $w \in Adj(G, v)$ como variáveis, ao evoluir o método RCM,
- n e ς, ao evoluir a heurística KP-band,
- n e $GrauN(G_S, w)$ dos vértices $w \in Adj(G_S, v)$, em que se considera apenas adjacências para vértices que ainda não foram rotulados, ao evoluir a heurística RLK.

Mostra-se o método heurístico resultante da hiper-heurística ACHH no algoritmo 9. O algoritmo recebe o grafo conexo $G = (V, E)$ e o vértice inicial $v \in V$ fornecido pelo algoritmo George-Liu.

O primeiro vértice rotulado é v, na linha 3, com $n = |V|$ (linha 2). Os demais vértices são rotulados com identificadores em ordem decrescente até 1 (veja as linhas 4, 5, 8 e 11). O algoritmo rotula, na linha 9, os vértices do grafo de entrada $G = (V, E)$ com a mesma distância do vértice pseudoperiférico de entrada v, na ordem fornecida pela sub-rotina *FormulaPrioridade()*, na linha 7, interna à estrutura de repetição *enquanto*, mostrada nas linhas 6 a 12.

A sub-rotina *FormulaPrioridade()* é representada como uma função sobrecarregada na linha 7. Se a hiper-heurística ACHH gera uma fórmula de prioridade:

- que é uma constante real, então, a heurística RBFS-GL é selecionada e a sub-rotina *FormulaPrioridade()* não é utilizada;
- evoluída a partir do método RCM, então, a sub-rotina *FormulaPrioridade*(G, n, v) ordena os vértices $w \in Adj(G, v)$ na ordem dada por uma fórmula que pode envolver n e $Grau(G, w)$;

Algoritmo 9: Método heurístico resultante da hiper-heurística ACHH.

Entrada: grafo conexo $G = (V, E)$; vértice $v \in V$;
Saída: numeração S;

```
1  início
2      n ← |V|;
3      s(v) ← n;
4      i ← n;
5      j ← n;
6      enquanto ( i > 1 ) faça
7          para cada ( vértice w ∈
           Adj(G, s^{-1}(j)) − {s^{-1}(n), s^{-1}(n − 1), ..., s^{-1}(i)}, na ordem
           de FormulaPrioridade(...) ) faça
8              i ← i − 1;
9              s(w) ← i;
10         fim-para-cada;
11         j ← j − 1;
12     fim-enquanto;
13     retorna S;
14 fim.
```

- evoluída a partir da heurística KP-band, então, a sub-rotina *FormulaPrioridade*(G, n, v) ordena os vértices $w \in Adj(G, v)$ na ordem dada por uma fórmula que pode envolver n e $\varsigma(w)$;
- evoluída a partir da heurística RLK, então, a sub-rotina *FormulaPrioridade*(G_S, n, v) ordena os vértices $w \in Adj(G_S, v)$ na ordem dada por uma fórmula que pode envolver n e $GrauN(G_S, w)$, que considera somente adjacências para vértices que ainda não foram rotulados.

A nova numeração S é retornada na linha 13.

3.3.4 Estrutura da hiper-heurística ACHH

A função de probabilidade utilizada no sistema ACHH é explicada na subseção 3.3.4.1. A atualização de feromônio pelas formigas é descrita na subseção 3.3.4.2. A substituição de fórmulas de prioridade é mostrada na subseção 3.3.4.3.

3.3.4.1 Função de probabilidade

Na iteração k, uma formiga a determina um caminho a percorrer o grafo de componentes $C = (V_C, E_C)$, a partir do nodo i, na partição 0, até o

nodo $\mathfrak{t}$, na partição 3, que representa o formigueiro, utilizando a função de probabilidade $\mathfrak{p}^a_{wj}(k) = \frac{\tau_{wj}}{\sum_{c \in \mathcal{N}_w} \tau_{wc}}$, em que τ_{wj} é a trilha de feromônio do nodo (componente) w para o nodo j e $\mathcal{N}_w$ é a vizinhança plausível do nodo w. Assim, $\mathcal{N}_w$ é um conjunto composto de componentes na mesma partição do nodo j, incluindo j. A função $\mathfrak{p}^a_{wj}(k)$ é a razão entre a quantidade de feromônio depositada na aresta (ou "trilha da formiga") que conecta o nodo (componente) w ao nodo j e a soma da quantidade de feromônio depositado nas arestas entre nodos w e nodos contidos no conjunto $\mathcal{N}_w$.

Uma formiga a utiliza a função de probabilidade $\mathfrak{p}^a_{wj}(k)$, na iteração k, para decidir qual componente de uma heurística para redução de largura de banda adicionar em sua solução parcial. A função é, portanto, a probabilidade de a formiga a incluir, na iteração k, o nodo (componente) em sua trilha do nodo i, na partição 0, para o nodo $\mathfrak{t}$, na partição 3.

3.3.4.2 Atualização de feromônio

Uma formiga deposita feromônio em sua trilha de acordo com a largura de banda e o tempo de execução para o método heurístico definido nos nodos desse caminho no grafo de componentes $C = (V_C, E_C)$, ao ser aplicado a cada grafo da fase de treinamento. Como mencionado, na analogia com colônia de formigas, em cada iteação k, cada formiga carrega "comida" (matrizes ou grafos) do nodo i, na partição 0, para o nodo $\mathfrak{t}$, na partição 3, que representa o formigueiro. Nesse processo, os algoritmos de reordenação armazenados nos nodos f_j, na partição 2 do grafo de componentes $C = (V_C, E_C)$, são avaliados com o conjunto de grafos da fase de treinamento, a cada iteração k.

O sistema ACHH utiliza uma taxa de evaporação de feromônio para evitar uma convergência muito rápida do algoritmo em um caminho do nodo i, na partição 0, para o nodo $\mathfrak{t}$, na partição 3, que representa o formigueiro, no grafo de componentes. A atualização do feromônio é dada por $\tau_{ij} = (1-P)\cdot\tau_{ij} + g(\varrho)$, em que $P \in (0, 1]$ é o parâmetro da taxa de evaporação, $g()$ é a função de qualidade do depósito de feromônio nas arestas, e ϱ é o tempo de execução do algoritmo de reordenação ou a redução de largura de banda realizada em uma solução. Gonzaga de Oliveira e Silva [63, 62] utilizaram $P = 0.3$.

O sistema ACHH utiliza duas funções para determinar a qualidade de um método heurístico e, consequentemente, atualizar a quantidade de feromônio nas arestas do grafo de componentes.

1. A primeira função é $\frac{1}{t}$, em que t é o tempo de execução, em milissegundos, em um nodo. O sistema ACHH utiliza essa função

para atualizar a quantidade de feromônio da aresta imediatamente percorrida pela formiga no grafo de componentes.

2. A segunda função é a largura de banda de uma solução. O sistema ACHH utiliza essa função para atualizar a quantidade de feromônio de cada aresta percorrida por uma formiga no grafo de componentes.

Dessa forma, a hiper-heurística adiciona $\beta(G) + \frac{1}{t}$ na quantidade de feromônio no caminho percorrido pela formiga no contexto de pré-processamento de matrizes para aceleração de resolutor de sistema de equações lineares [63]. Por outro lado, $\beta(G)$ foi definida como cinco vezes mais importante que $\frac{1}{t}$ no contexto de gerar métodos heurísticos para redução de largura de banda de matrizes [62]. Assim, o sistema ACHH adiciona $5 \cdot \beta(G) + \frac{1}{t}$ na quantidade de feromônio no caminho percorrido pela formiga [62] nesse caso. O valor 5 foi escolhido arbitrariamente, mas outro valor poderia ser determinado. Uma valor maior daria ênfase à redução de largura de banda da fórmula de prioridade, em detrimento ao tempo de execução.

3.3.4.3 Substituição de fórmulas de prioridade

O sistema ACHH substitui metade dos nodos f_j, na partição 2, com novos nodos a cada 10 iterações. O sistema substitui nodos com arestas incidentes com a menor quantidade de feromônio. As formigas selecionam como componentes nodos incluídos recentemente porque os nodos já no sistema já têm valores de feromônio em suas arestas incidentes.

3.3.5 Pseudocódigo da hiper-heurística ACHH

Mostra-se o sistema ACHH no algoritmo 10. A hiper-heurística ACHH recebe o conjunto de grafos para o treinamento, o número de iterações e a quantidade de formigas.

A probabilidade de busca no espaço de soluções e o tempo de execução da hiper-heurística ACHH são proporcionais ao número de formigas, ao número de componentes no grafo $C = (V_C, E_C)$, ao número de iterações e à quantidade e tamanho dos grafos $\mathscr{G} \in Grafos$. Gonzaga de Oliveira e Silva [63, 62] calibraram esses parâmetros de forma que a hiper-heurística ACHH executasse em tempo razoável. Assim, os autores executaram a hiper-heurística com seis formigas e 100.000 iterações para selecionar ou gerar novos métodos heurísticos para redução de largura de banda para cada área de aplicação específica.

A hiper-heurística inicializa a variável k na linha 2. A estrutura de repetição *enquanto* nas linhas 3 a 19 executa o número de iterações da hiper-heurística. Na estrutura de repetição *para cada* nas linhas 4 a 16,

cada uma das seis formigas faz com que o algoritmo George-Liu (linha 8), representado no nodo p, na partição 1 do grafo de componentes $C = (V_C, E_C)$, e o algoritmo de reordenação com fórmula de prioridade representada no nodo f_j, na partição 2, sejam aplicados a cada grafo $\mathscr{G} \in Grafos$ da fase de treinamento, por meio da estrutura de repetição nas linhas 6 a 15.

Na linha 5, a hiper-heurística utiliza a função de probabilidade $\mathfrak{p}^a_{wj}(k)$ para determinar um nodo f_j, para $j \in [1, 6]$, na partição 2, para a formiga percorrer o grafo de componentes. Especificamente, com o nodo f_j, a hiper-heurística forma um caminho com vértices $i, p, f_i, \mathfrak{t} \in V_C$, para a formiga a percorrer, pelas arestas $\{i, p\}, \{p, f_j\}, \{f_j, t\} \in E_C$, o grafo de componentes $C = (V_C, E_C)$ na iteração k.

Na linha 7, a hiper-heurística calcula a largura de banda original do grafo $\mathscr{G} \in Grafos$. Em seguida, o vértice u recebe o vértice pseudo-periférico retornado pelo algoritmo George-Liu na linha 8. Na linha 9, a hiper-heurística atualiza a quantidade de feromônio na aresta $\{i, p\}$, que conecta o nodo inicial i, na partição 0, para o nodo p, que representa o nodo na partição 1 com o algoritmo George-Liu. Nessa atualização, a hiper-heurística utiliza $\frac{1}{t}$, em que t, em milissegundos, é o tempo de execução do algoritmo George-Liu.

A estrutura de nível enraizada no vértice u, $\mathscr{L}(u)$, é construída na linha 10. Também nessa linha, a hiper-heurística determina a prioridade de cada vértice $v \in \mathscr{G}$, utilizando a fórmula de prioridade armazenada no nodo f_i, na partição 2. Na linha 11, a hiper-heurística atualiza o feromônio da aresta $\{p, f_j\}$, que conecta os nodos nos níveis 1 e 2, utilizando $\frac{1}{t}$, em que t é o tempo de execução de calcular as prioridades de cada vértice do grafo $\mathscr{G} \in Grafos$, pela fórmula de prioridade armazenada no nodo f_j na partição 2.

Na linha 12, a hiper-heurística numera os vértices da grafo de treinamento $\mathscr{G} \in Grafos$, de acordo com o método heurístico com fórmula de prioridade contida no nodo f_j, na partição 2, determinado na linha 5. Na linha 13, a hiper-heurística atualiza o feromônio da aresta $\{f_j, \mathfrak{t}\}$, que conecta os nodos nos níveis 2 e 3, utilizando $\frac{1}{t}$, em que t é o tempo de execução do algoritmo de reordenação, com fórmula de prioridade armazenada no nodo f_j, na partição 2, aplicado ao grafo $\mathscr{G} \in Grafos$.

Na linha 14, a hiper-heurística atualiza o feromônio do caminho dado pelos vértices $i, p, f_i, \mathfrak{t} \in V_C$ pelas arestas $\{i, p\}, \{p, f_j\}, \{f_j, t\} \in E_C$, determinado na linha 5, considerando a largura de banda da nova numeração do grafo $\mathscr{G} \in Grafos$, realizada na linha 12. Especificamente, é considerada a diferença de largura de banda da numeração original, calculada na linha 7, com a largura de banda da nova numeração do grafo $\mathscr{G} \in Grafos$, realizada na linha 12. Como mencionado na subseção 3.3.4, na página 62, essa atualização da quantidade de feromô-

nio em todo o caminho percorrido pela formiga a, relacionada à redução da largura de banda do grafo $\mathscr{G} \in Grafos$, é multiplicada por cinco no contexto de evolução de métodos heurísticos para redução de largura de banda de matrizes [62]. Assim, essa atualização da quantidade de feromônio em todo o caminho percorrido pela formiga a, relacionada à redução da largura de banda do grafo $\mathscr{G} \in Grafos$, tem peso cinco vezes maior que as atualizações, realizadas nas linhas 9, 11 e 13, das quantidades de feromônio relacionadas com o tempo de execução dos passos executados nas linhas 8, 10 e 12, respectivamente [62].

A hiper-heurística incrementa a variável k na linha 17. Na linha 18, a cada 10 iterações, o sistema ACHH gera, aleatoriamente, três novas fórmulas de prioridade. Essas três novas fórmulas de prioridade substituem as fórmulas de prioridade dos três (metade) nodos f_j, na partição 2, com arestas incidentes (considerando nodos nos níveis 1 e 3) com menos feromônio que os demais nodos na partição 2 do grafo de componentes $C = (V_C, E_C)$. Finalmente, na linha 20, o sistema ACHH retorna a fórmula de prioridade armazenada no nodo f_j, na partição 2, com a maior quantidade de feromônio, considerando as arestas incidentes aos nodos nos níveis 1 e 3.

3.3.6 Considerações finais

Gonzaga de Oliveira e Silva [62] aplicaram o sistema ACHH em matrizes simétricas e assimétricas oriundas de vários domínios. Um grupo de matrizes de mesmo domínio foi utilizado como conjunto de treinamento para gerar um novo método heurístico especializado em redução de largura de banda de matrizes na área de aplicação específica do grupo de matrizes. Os autores utilizaram sete áreas de aplicação compostas de matrizes simétricas e três áreas de aplicação compostas de matrizes assimétricas oriundas da coleção de matrizes SuiteSparse [20]. Como exemplo, os autores utilizaram a hiperheurística ACHH em matrizes simétricas oriundas de problemas termodinâmicos no processo de aprendizagem. Nesse exemplo, foram utilizadas as matrizes ted_B ($n = 10.605$, $|E| = 144.579$), lshp3466 ($n = 3.466$, $|E| = 23896$) e lshp3025 ($n = 3025$, $|E| = 20.833$). O sistema ACHH retornou como o melhor programa para essas três matrizes um novo método heurístico para redução de largura de banda baseado na heurística KP-band em que a subrotina *FormulaPrioridade(...)* na linha 7 do algoritmo 9 é dada por $0.11364 \cdot \varsigma^7 + 0.060522 \cdot \varsigma^6 \cdot n - 0.022397 \cdot \varsigma^6 - 0.308595 \cdot \varsigma^4 - 1.29987 \cdot \varsigma^3 \cdot n - \varsigma^2 - 0.745801$, em que ς é a soma dos graus dos vértices adjacentes ao vértice $w \in V$. Cada vértice w é adjacente ao vértice em processamento $s^{-1}(j) \in V$. Assim, os vértices w são ordenados conforme essa fórmula de prioridade.

Os métodos heurísticos resultantes da hiper-heurística ACHH têm

baixo custo computacional, conforme os métodos RCM, KP-band, RBFS-GL e RLK. Em todos os testes seriais realizados com matrizes de grande porte (com dimensão maior que um milhão), os métodos heurísticos resultantes da hiper-heurística ACHH superaram os métodos heurísticos de baixo custo computacional no estado da arte em qualidade da solução [62], bem como na aceleração de resolutor de sistemas de equações lineares [63] (veja a seção 2.6). Portanto, os métodos heurísticos resultantes da hiper-heurística ACHH podem ser considerados os algoritmos de baixo custo computacional no estado da arte para redução de largura de banda de matrizes de grande porte.

Algoritmo 10: ACHH.

Entrada: conjunto de $Grafos$; número de iterações $NrIter$;
conjunto $ants$ de formigas;
Saída: fórmula de prioridade evoluída para algoritmo de reordenação;

```
1  início
2      k ← 0;
3      enquanto ( (k < NrIter) ) faça
4          para cada ( (formiga a ∈ ants) ) faça
5              utilizando a função de probabilidade 𝔭^a_wj(k), determine um
               nodo f_j, para j ∈ [1,6], na partição 2, para um caminho
               para a formiga a percorrer o grafo de componentes
               C = (V_C, E_C);
6              para cada ( (grafo 𝒢 ∈ Grafos) ) faça
7                  calcule a largura de banda original;
8                  u recebe o vértice pseudoperiférico retornado pelo
                   algoritmo George-Liu (no nodo na partição 1);
9                  atualize a quantidade de feromônio na aresta
                   {i,p} ∈ E_C, utilizando o custo do algoritmo
                   George-Liu;
10                 ao construir 𝒮(u), calcule a prioridade de cada vértice
                   v ∈ 𝒢, utilizando a fórmula armazenada no nodo f_i;
11                 atualize a quantidade de feromônio na aresta que
                   conecta o nodo na partição 1 ao nodo f_i, na partição 2,
                   utilizando o tempo de execução do passo anterior;
12                 numere cada vértice v ∈ 𝒢, de acordo com a heurística
                   com fórmula contida no nodo f_j, na partição 2;
13                 atualize a quantidade de feromônio da aresta que
                   conecta o nodo f_j, na partição 2, ao nodo na partição
                   3, considerando o tempo para numerar o grafo 𝒢;
14                 atualize a quantidade de feromônio do caminho
                   escolhido na linha 5, do nodo i ao nodo t, considerando
                   a largura de banda da numeração do grafo 𝒢;
15             fim-para-cada;
16         fim-para-cada;
17         k ← k + 1;
18         se ( (mod(k,10) = 0) ) então gere novas fórmulas de
           prioridade para três (metade) nodos f_j com arestas incidentes
           para o nodo i, na partição 1, e para o nodo p, na partição 3,
           com menos feromônio que os demais nodos na partição 2;
19     fim-enquanto;
20     retorna a fórmula de prioridade armazenada no nodo f_j, na
       partição 2, com a maior quantidade de feromônio, considerando as
       arestas incidentes aos nodos nos níveis 1 e 3.
21 fim.
```

Capítulo 4

Algoritmos meta-heurísticos

4.1 Introdução

Diferentemente das aplicações citadas no apêndice A, há aplicações que não impõem um limite restrito no tempo de execução do método heurístico para redução de largura de banda, tais como [55]: *browsing hypertext*, *small world networks*, análise visual de conjuntos de dados utilizando matrizes de similaridade visual, minimização da taxa de entropia de grafos, verificação de modelos simbólicos, otimização do *layout* de malhas, *the seriation problem*. Nesses casos, um algoritmo meta-heurístico pode ser aplicado porque pode fornecer resultados melhores que métodos baseados em conceitos da teoria dos grafos, como os métodos apresentados nos capítulos 2 e 3, apesar de apresentar custo de execução maior. Também, pode-se ter noção de quão distante um método rápido, baseado em conceitos da teoria dos grafos, retorna em relação ao melhor resultado já encontrados na matriz, fornecido por um algoritmo meta-heurístico.

Basicamente, meta-heurística é uma técnica de construção de métodos heurísticos. Como comentado na seção 1.4, na página 16, meta-heurísticas passaram a ser aplicadas em grande escala para a solução aproximada à solução ótima de problemas de otimização na década de 1990. Assim, os primeiros algoritmos meta-heurísticos para redução de largura de banda foram publicados nessa década [9].

O problema de redução de largura de banda é um dos principais problemas de otimização combinatória. Dessa forma, grande parte das principais meta-heurísticas já foi aplicada no problema, incluindo algoritmos genéticos [85, 79], *Iterated Local Search* [85, 86, 55], otimização por colônia de formigas [76, 63, 62], programação genética [80], *Variable Neighbourhood Search* [94], *simulated annealing* [113], BRKGA [107] e GRASP [107]. Foram publicadas diversas versões de métodos heurísticos projetados por essas meta-heurísticas e apenas foram citados alguns

dos principais exemplos.

Neste capítulo, são apresentados alguns dos principais algoritmos meta-heurísticos para redução de largura de banda de matrizes. As heurísticas *Node Centroid with Hill Climbing* (NCHC) [86] e *Fast Node Centroid with Hill Climbing* (FNCHC) [86] são descritas na seção 4.2. Esses algoritmos meta-heurísticos são as bases da heurística FNCHC+ [55], apresentada na subseção 4.2.6, que é o algoritmo meta-heurístico no estado da arte, em simulações seriais, para matrizes de grande porte, isto é, com dimensão maior que um milhão.

O método heurístico projetado pela meta-heurística *Variable Neighbourhood Search* (VNS-band) [94] é apresentado na seção 4.3. Até 2015, a heurística VNS-band era o algoritmo meta-heurístico no estado da arte para matrizes de porte muito pequeno, isto é, com dimensões até aproximadamente 1.000. A heurística VNS-band foi superada pela heurística DRSA [113], que é o atual algoritmo meta-heurístico no estado da arte para matrizes desse porte. A heurística DRSA [113] é descrita na seção 4.4.

4.2 Heurísticas NCHC, FNCHC e FNCHC+

A heurística *Node Centroid with Hill Climbing* (NCHC) baseia-se no ajuste de numeração para a exploração do espaço de soluções e no procedimento *Hill Climbing* para a busca local. Foi proposta com o intuito de ser uma heurística rápida, simples e que gerasse resultados satisfatórios. A heurística é dividida em quatro etapas:

- solução inicial,
- ajuste da numeração pelo procedimento *Node Centroid*,
- busca local e
- verificação se a nova solução é aceita.

A heurística NCHC inicia com uma solução gerada pela busca em largura iniciada em um vértice aleatório. A heurística altera essa numeração inicial pelo ajuste dos rótulos dos vértices em relação a seus vértices adjacentes. Os autores denominaram esse ajuste do rótulo do vértice como *Node Centroid* (NC) em relação a seus vértices adjacentes. Finalmente, a heurística aplica a busca local pelo procedimento *Hill Climbing* para obter um ótimo local. A nova solução é aceita se for melhor que a solução corrente.

O restante desta seção está dividida como descrito a seguir. A etapa de ajuste da numeração pelo procedimento *Node Centroid* é apresentada na subseção 4.2.1. A busca local *Hill Climbing* é descrita na

subseção 4.2.2. A condição se a nova solução é aceita é mostrada na subseção 4.2.3. A heurística NCHC completa também é descrita na subseção 4.2.3. A meta-heurística *Iterated Local Search* (ILS) é abordada na subseção 4.2.4. As heurísticas FNCHC e FNCHC+ são apresentadas na subseções 4.2.5 e 4.2.6, respectivamente.

4.2.1 Procedimento *Node Centroid*

Após obter a solução inicial, associa-se, na segunda etapa da heurística NCHC, um peso $\mathfrak{p}$ a cada vértice do grafo. Em seguida, os vértices são renumerados em ordem crescente de peso $\mathfrak{p}$ de cada vértice.

O procedimento *Node Centroid* utiliza a noção de *vértice crítico* de $G_S = (V, E)$. (Veja as definições 1.39 e 1.40, na página 16 da seção 1.3.) Na heurística NCHC, um vértice v é dito λ-crítico se $\beta_v(G_S) \geq \lambda \cdot \beta(G_S)$, com $\lambda \in [0, 1]$. Ainda, para o grafo $G_S = (V, E)$, são definidos os vértices $\lambda-$adjacentes de v como $V_\lambda(v) = Adj(G_S, v) \cap \{u : (\{v, u\} \in E) \ |s(u) - s(v)| \geq \lambda \cdot \beta(G_S)\}$. Com o conjunto $\lambda-$adjacentes $V_\lambda(v)$, têm-se o *feixe* do vértice v, definido por $f_\lambda(v) = V_\lambda(v) \cup \{v\}$. O vértice v é o vértice centroide do feixe $f_\lambda(v)$.

Na etapa de ajuste de numeração, busca-se reduzir a largura de banda dos vértices $\lambda-$críticos, com $\lambda \approx 1$, incluindo $\lambda = 1$, trocando-se os rótulos dos vértices centroides de seus respectivos feixes. Essa troca é realizada ao se associar um peso, definido como $\mathfrak{p}(v) = \frac{\sum_{u \in f_\lambda(v)} s(u)}{|f_\lambda(v)|}$, para cada vértice $v \in V$ para, depois, numerar os vértices em ordem crescente desse peso.

Mostra-se a sub-rotina *Node Centroid* no algoritmo 11. O algoritmo recebe um grafo $G_S = (V, E)$ com numeração S e o parâmetro λ.

Na estrutura de repetição *para cada* nas linhas 2 a 5, os vetores $\mathfrak{p}$ e c são inicializados, em que $c(v)$ é o número de vértices pertencentes a $f_\lambda(v)$. Os vetores $\mathfrak{p}$ e c de cada vértice $\lambda-$crítico (veja a condição na linha 7) são atualizados na estrutura de repetição *para cada* das linhas 6 a 13. Atualiza-se o valor de $\mathfrak{p}$ pela razão de $\mathfrak{p}$ por c para cada vértice $v \in V$ na estrutura de repetição *para* da linha 14. Em seguida, na linha 15, os vértices são renumerados em ordem crescente pelo peso $\mathfrak{p}$. O algoritmo retorna a renumeração S' na linha 16.

4.2.2 Busca local por *Hill Climbing*

Nesta etapa, um conjunto de vértices críticos do grafo é selecionado para que seus rótulos sejam trocados. O conjunto E_c de arestas críticas de $G_S = (V, E)$ é definido por $E_c = \{\{u, v\} \in E : |s(u) - s(v)| = \beta(G_S)\}$. O conjunto de vértices críticos V_c de $G = (V, E)$ é todo vértice de

Algoritmo 11: *NodeCentroid*.

Entrada: grafo $G_S = (V, E)$ com numeração S; criticidade λ;
Saída: renumeração S';

início
  para cada (*vértice* $v \in V$) **faça**
    $\mathfrak{p}(v) \leftarrow s(v)$; // $\mathfrak{p}(v)$: peso associado a cada vértice v
    // $c(v)$ é o número de vértices do feixe do vértice v
    $c(v) \leftarrow 1$;
  fim-para-cada;
  para cada (*aresta* $\{u, v\} \in E$) **faça**
    se ($|s(u) - s(v)| \geq \lambda \cdot \beta(G_S)$) **então**
      $\mathfrak{p}(u) \leftarrow \mathfrak{p}(u) + s(v)$;
      $c(u) \leftarrow c(u) + 1$;
      $\mathfrak{p}(v) \leftarrow \mathfrak{p}(v) + s(u)$;
      $c(v) \leftarrow c(v) + 1$;
    fim-se;
  fim-para-cada;
  para (*vértice* $v \in V$) **faça** $\mathfrak{p}(v) \leftarrow \frac{\mathfrak{p}(v)}{c(v)}$;
  // os vértices $v \in V$ são renumerados em ordem crescente
  $S' \leftarrow NumeraCrescente(G_S, \mathfrak{p})$; // do peso $\mathfrak{p}(v)$
  retorna S';
fim.

arestas de E_c. Veja também as definições 1.39 e 1.40, na página 16 da seção 1.3.

Para a aplicação da busca local, define-se a vizinhança reduzida $\mathrm{N}_S(v)$ do vértice crítico $v \in V_c$ para a troca de rótulo de v como $\mathrm{N}_S(v) = \{u : (\{v, u\} \in E_c) \ |med(v) - s(u)| < |med(v) - s(v)|\}$, em que $med(v) = \left\lfloor \frac{s_{\max}(v) + s_{\min}(v)}{2} \right\rfloor$, $s_{\max}(v) = \max_{\{v,u\} \in E} [s(u)]$ e $s_{\min}(v) = \min_{\{v,u\} \in E} [s(u)]$. Isso significa que, para os vértices críticos $v \in V_c$, $\mathrm{N}_S(v)$ é o conjunto de vértices $u \in Adj(G_S, v)$ com diferença de numeração média de v em relação ao rótulo de u menor que a diferença da numeração média de v pelo rótulo de v. O conjunto de vértices para a aplicação da busca local, ou o conjunto de vizinhanças reduzidas para a troca de numeração em relação à solução S, é definido por $N(S) = \bigcup_{v \in V_c} \mathrm{N}_S(v)$.

Como a etapa da busca local por *Hill Climbing* é depois da etapa de ajuste de numeração, a solução S' é a solução corrente. Com $N(S')$, forma-se um conjunto de vértices apropriados para a troca. A busca local é aplicada no conjunto $N(S')$. Troca-se o rótulo do vértice crítico v com cada rótulo dos vértices $u \in \mathrm{N}_{S'}(v) \subseteq N(S')$ até que uma melhora em relação à solução corrente S' seja constatada. Com isso, gera-se a

solução S''.

Verifica-se se uma melhora ocorre quando um vértice crítico deixa de ser crítico. Para isso, considera-se o *valor crítico*

$$\Gamma_{S'}(v) = \begin{cases} \beta_v(G_{S'}) - \beta(G_{S'}), & \text{se } \beta_v(G_{S'}) \geq \beta(G_{S'}) \\ \beta_v(G_{S''}) - \beta_v(G_{S'}), & \text{se } \beta_v(G_{S''}) > \beta_v(G_{S'}) \end{cases},$$

em que S'' é a solução gerada pela troca do rótulo do vértice v. Se $\Gamma_{S'}(v) = 0$, então, v é um vértice crítico. Se $\Gamma_{S'}(v) > 0$, então, a troca do rótulo do vértice v aumentou $\beta_v(G_{S''})$ em relação à largura de banda $\beta_v(G_{S'})$. Consequentemente, isso aumenta a largura de banda, o que não se deseja.

Seja $u \in \mathrm{N}_{S'}(v)$. Uma melhora em relação à solução corrente é constatada se $\Gamma_{S''}(u) \leq \Gamma_{S'}(u)$, $\Gamma_{S''}(v) \leq \Gamma_{S'}(v)$ e $\Gamma_{S''}(u) + \Gamma_{S''}(v) < \Gamma_{S'}(u) + \Gamma_{S'}(v)$.

Mostra-se a etapa de busca local por *Hill Climbing* no algoritmo 12. O algoritmo recebe um grafo $G_{S'} = (V, E)$ com numeração S'.

Na estrutura de repetição *para cada* das linhas 3 a 17, os vértices críticos do grafo são selecionados na linha 6. Em seguida, para cada vértice crítico v, são verificados os vértices $u \in \mathrm{N}_{S''}(v)$ na estrutura de repetição nas linhas 7 a 14. Com isso, os rótulos dos vértices u e v sãos trocados, gerando a solução candidata auxiliar S_{aux} na linha 8. Se a solução candidata auxiliar S_{aux} é melhor que a solução corrente (linha 9), então, a solução candidata S'' é atualizada na linha 10.

O algoritmo para quando não forem identificadas melhorias em relação à solução corrente S''. O algoritmo retorna a renumeração S'' na linha 18.

Se em nenhuma iteração do algoritmo 12 a condição da linha 9 for satisfeita, então, nenhuma melhoria em relação à solução candidata S' foi realizada. Com isso, o algoritmo retorna a numeração S', que foi atribuída à numeração S'' na linha 2.

4.2.3 Pseudocódigo da heurística NCHC

A heurística NCHC [86] é mostrada no algoritmo 13. O algoritmo recebe o grafo $G = (V, E)$, o número máximo de iterações i_{max} e o número máximo de iterações NC_{max} para a etapa do ajuste de numeração.

A estrutura de repetição *para*, mostrada nas linhas 3 a 10, é executada i_{max} vezes. Na linha 4, o algoritmo gera uma solução inicial S pela busca em largura, mostrada no algoritmo 2, na página 29 da seção 2.3, a partir de um vértice aleatório. Na estrutura de repetição *para* mostrada nas linhas 5 a 8, o algoritmo altera a numeração S pelo procedimento *Node Centroid* (algoritmo 11) NC_{max} vezes na linha 6.

Lim, Rodrigues e Xiao [86] constataram que aplicar a busca local na metade das iterações, e não em todas, torna a heurística mais rápida

Algoritmo 12: *HillClimbing*.

Entrada: grafo $G_{S'} = (V, E)$ com renumeração S';
Saída: renumeração S'';

```
1  início
2      S'' ← S';
3      repita
4          houveMelhora ← falso;
5          para cada ( vértice v ∈ V ) faça
6              se ( Γ_{S''}(v) = 0 ) então
7                  para cada ( vértice u ∈ N_{S''}(v) ) faça
                       // troca o rótulo de v pelo rótulo de u
                       // e gera a solução auxiliar S_aux
8                      S_aux ← TrocaRótulo(S'', v, u);
9                      se ( (Γ_{S_aux}(u) ≤ Γ_{S''}(u)) ∧ (Γ_{S_aux}(v) ≤
                       Γ_{S''}(v)) ∧ ((Γ_{S_aux}(u)+Γ_{S_aux}(v)) < (Γ_{S''}(u)+Γ_{S''}(v)))
                       ) então
                           // solução S'' com a troca dos rótulos
10                         S'' ← S_aux; // realizada na linha 8
                           // realiza a busca local enquanto houver
11                         houveMelhora ← verdadeiro; // melhorias
12                         break 2; // vai para o teste da linha 17
13                     fim-se;
14                 fim-para-cada;
15             fim-se;
16         fim-para-cada;
17     até que (¬houveMelhora) ;
       // se a condição da linha 9 não for satisfeita em
       // nenhuma iteração, então, retorna S'' = S', pois
18     retorna S''; // nenhuma melhoria foi realizada
19 fim.
```

e proporciona soluções satisfatórias. Com isso, os autores utilizam a estrutura condicional da linha 7 para aplicar a busca local na metade das iterações.

Se a solução candidata S' é melhor que a melhor solução corrente S^*, então, a solução S^* é atualizada na linha 9. O algoritmo retorna, na linha 11, a renumeração S^* com a menor largura de banda encontrada.

Gonzaga de Oliveira e Carvalho [55] constaram que o procedimento *Node Centroid* depende fortemente do vértice inicial da busca em largura. Veja a linha 4 do algoritmo 13. Os autores mostraram que o procedimento *Node Centroid* retorna um resultado diferente para cada vértice inicial da busca em largura. Essa característica é desejada para que se tenha um novo resultado a cada iteração do procedimento.

Algoritmo 13: NCHC.

Entrada: grafo $G_S = (V, E)$; número máximo de iterações i_{max} da heurística NCHC; número máximo de iterações NC_{max} para a sub-rotina $NodeCentroid$;
Saída: nova solução S^*;

```
1  início
2      S* ← S;
3      para ( i ← 1; i ≤ i_max; i ← i + 1 ) faça
           // solução inicial fornecida pela busca em largura
           // (algoritmo 2), a partir de um vértice aleatório
4          S ← NumeraBuscaEmLargura(G,VérticeAleatório(V));
5          para ( j ← 1; j ≤ NC_max; j ← j + 1 ) faça
               // altera a numeração por Node Centroid
6              S' ← NodeCentroid(S); // algoritmo 11
               // HillClimbing (algoritmo 12) retorna a
               // renumeração S'', que será a nova solução S'
7              se ( (j mod 2) = 1 ) então S' ← HillClimbing(S');
8          fim-para;
9          se ( β(G_S*) > β(G_S') ) então S* ← S';
10     fim-para;
11     retorna S*;
12 fim.
```

4.2.4 Meta-heurística *Iterated Local Search* (ILS)

A heurística NCHC segue os conceitos da meta-heurística *Iterated Local Search* (ILS). Essa meta-heurística pode ser vista como uma generalização e, consequentemente, fornecendo mais flexibilidade, que a meta-heurística VNS, apresentada na subseção 4.3.1, na página 77.

Basicamente, um método heurístico projetado pela meta-heurística ILS conduz uma busca local pelo espaço de soluções. Com uma busca local específica para o problema (procedimento *Hill Climbing* mostrado no algoritmo 12 e aplicado na linha 7 do algoritmo 13), o método heurístico contém quatro componentes principais:

1. gera uma solução inicial adequada para o problema (pela busca em largura a partir de um vértice aleatório mostrada na linha 4 do algoritmo 13),

2. altera essa solução (na linha 6 do algoritmo 13, pelo procedimento *Node Centroid* mostrado no algoritmo 11) - etapa também chamada de perturbação no contexto de meta-heurísticas,

3. aplica uma busca local (na linha 7 do algoritmo 13, pelo procedimento *Hill Climbing* mostrado no algoritmo 12) e

4. utiliza um critério para decidir se a nova solução encontrada é mantida (na linha 9 do algoritmo 13).

4.2.5 Heurística FNCHC

A heurística *Fast Node Centroid with Hill Climbing* (FNCHC) [86] é uma variação da heurística NCHC [86]. Na heurística FNCHC, os parâmetros foram definidos e ajustados de acordo com o tamanho da matriz de entrada [86]. Com isso, uma redução do custo computacional pode ser alcançada em relação à heurística NCHC.

Para a heurística FNCHC [86], os autores definiram, empiricamente, valores adequados para o parâmetro λ. O parâmetro λ é explicado na subseção 4.2.1, na página 71. De acordo com os autores [86], nas instâncias testadas, os melhores resultados foram obtidos pela heurística FNCHC com $\lambda = 0{,}95$, para a redução de largura de banda em relação aos resultados obtidos com outros valores de λ. Claramente, os melhores tempos de execução foram obtidos pela heurística FNCHC com $\lambda = 1$ em relação aos resultados obtidos com outros valores de λ. Na heurística FNCHC [86], os autores utilizaram $i_{max} = 25$ e $NC_{max} = 100$.

4.2.6 Heurística FNCHC+

A busca local *Hill Climbing*, mostrada no algoritmo 12, na página 74, tem custo de execução alto. Mesmo aplicando a busca local *Hill Climbing* na metade das vezes em que o procedimento *Node Centroid* é invocado (veja as linhas 5 a 8 no algoritmo 13), a heurística FNCHC apresenta tempo de execução alto para ser aplicada, em tempos de execução razoáveis, em matrizes de grande porte, isto é, em matrizes com dimensão maior que um milhão [55].

Gonzaga de Oliveira e Carvalho [55] constaram que invocar a busca local *Hill Climbing* a cada duas iterações, como ocorre na heurística FNCHC, não gera ganhos reais na solução S^*. A heurística FNCHC+ invoca a busca local *Hill Climbing* somente quando uma nova solução candidata S' tem largura de banda similar à melhor solução S^* corrente. Dessa forma, a heurística FNCHC+ utiliza a condição $\beta_{S'}(G) - \beta_{S^*}(G) < \kappa$, para $\kappa \in \mathbb{Z}$, na linha 9 do algoritmo 13. Além disso, em vez de utilizar valores predefinidos para i_{max} e NC_{max} como na heurística FNCHC, a heurística FNCHC+ permite atribuir valores diferentes para esses parâmetros para diferentes matrizes.

Com essas modificações, a heurística FNCHC+ superou a heurística FNCHC ao serem aplicadas a matrizes de grande porte. Consequentemente, a heurística FNCHC+ é o algoritmo meta-heurístico no estado da arte para redução de largura de banda de matrizes de grande porte.

Gonzaga de Oliveira e Carvalho [55] estabeleceram 1.200 segundos como o limite de tempo de execução para métodos heurísticos seriais fornecerem uma solução em matrizes com até 19 milhões de vértices (matriz *mawi_201512012345*) e 57 milhões de arestas (matriz wb-edu). Com isso, a heurística FNCHC+ superou as demais heurísticas na qualidade da solução.

4.3 Heurística com busca em vizinhança variável

Na heurística *Variable Neighbourhood Search for bandwidth reduction* (VNS-band), uma solução S é o conjunto dos rótulos dos vértices, conforme descrito na definição 1.23, na seção 1.3, na página 10. Na heurística VNS-band, uma solução inicial dos rótulos dos vértices é construída por uma busca em largura aleatória, iniciando-se por um vértice aleatório. Os demais passos da meta-heurística VNS, busca por nova solução candidata (etapa chamada de perturbação no contexto de meta-heurísticas), busca local e troca de vizinhança, são seguidos até que o critério de parada seja satisfeito.

A meta-heurística *Variable Neighbourhood Search* (VNS) é descrita na subseção 4.3.1. As etapas da heurística VNS-band são descritas nas subseções seguintes. Na subseção 4.3.2, descreve-se o passo de inicialização. A etapa de busca por nova solução candidata (perturbação) é descrita na subseção 4.3.3. A busca local é explicada na subseção 4.3.4. Descreve-se, na subseção 4.3.5, a etapa da permutação de vizinhança. Finalmente, na subseção 4.3.6, explica-se a heurística em detalhes.

Como mencionado na seção 4.1, a heurística VNS-band era o algoritmo meta-heurístico no estado da arte para matrizes com dimensões até aproximadamente 1.000 até ser superada pela heurística DRSA [113], que é o atual algoritmo meta-heurístico no estado da arte para matrizes desse porte. Apesar de ter sido superada na qualidade da solução pela heurística DRSA, a heurística VNS-band apresenta custo de execução menor que a heurística DRSA [62].

4.3.1 Meta-heurística VNS

A meta-heurística *Variable Neighbourhood Search* (VNS) [93] é dividida em quatro etapas: solução inicial, busca por nova solução candidata (perturbação), busca local e troca de vizinhança.

Após uma solução inicial S, o raio k da vizinhança da solução inicial S é inicializado com k_{min}. Alterações na solução candidata S são realizadas para tentar "escapar dos vales" do espaço de soluções a que os mínimos locais se localizam. Na etapa de busca por nova solução candidata (perturbação), uma solução S' é encontrada na vizinhança

$N_{k_{min}}(S)$.

Como ocorre com outras meta-heurísticas, há a tentativa de se alcançar um ótimo local por meio de busca local. Realiza-se uma exploração dentro do raio da vizinhança corrente. Uma busca local é aplicada na solução candidata S' para encontrar uma solução candidata S''.

Na quarta etapa, verifica-se se a solução candidata S'' é melhor que a solução S corrente. Caso seja, a solução candidata S'' passará a ser a solução corrente e k será atualizado novamente para k_{min}. Caso contrário, o raio da vizinhança k é incrementado com um k_{incr}, que é ajustado de acordo com o problema.

Esses passos são repetidos até que um critério de parada seja satisfeito. Um número máximo de iterações ou um tempo máximo de processamento são exemplos de critério de parada.

Seja C_S o conjunto de soluções possíveis para um determinado problema de otimização. A vizinhança $N_k(S)$, com $S \in C_S$, é o conjunto de soluções vizinhas à solução candidata S, dentro do raio k. Inicia-se por uma solução S e um método heurístico projetado pela meta-heurística VNS encontra, por meio de busca local, uma solução $S'' \in N_k(S)$. A solução S corrente e a vizinhança $N_k(S)$ são trocadas pela solução S'' e pela vizinhança $N_k(S'')$ se e, somente se, o custo da solução S'' é melhor que o custo da solução S. Caso a solução S'' não seja melhor que a solução S corrente, então, aumenta-se o raio de abrangência k de $N_k(S)$ e realiza-se uma nova busca.

Exemplifica-se, na figura 4.1, a busca por soluções candidatas pela meta-heurística VNS dentro das estruturas de vizinhança. Inicia-se pela solução candidata S e realiza-se a exploração da vizinhança até o raio k, encontrando-se a solução S''_1.

Neste exemplo, a solução S''_1 não apresentou melhoria em relação à solução S corrente. Dessa forma, a solução S continua como a solução corrente. Em seguida, aumenta-se o raio de $N_k(S)$ para $N_{k+1}(S)$. Com isso, encontra-se a solução S''_2, que representa melhoria em relação à solução S. Portanto, a solução candidata S''_2 passa a ser a solução corrente. A busca, então, passa a ser na vizinhança $N_k(S''_2)$. Esses passos são repetidos até que um critério de parada seja satisfeito.

4.3.2 Inicialização

A primeira etapa da heurística VNS-band consiste em construir uma solução inicial da numeração dos vértices de $G = (V, E)$. A solução inicial dos rótulos dos vértices é construída por uma busca em largura aleatória, iniciando-se por um vértice aleatório. Com a busca em largura aleatória, a heurística gera uma estrutura de nível enraizada $\mathscr{L}(v)$ em um vértice v aleatório. A renumeração em cada nível $L_i(v)$, para $1 \leq i \leq \ell(v)$, é iniciada também por um vértice aleatório $u \in L_i(v)$.

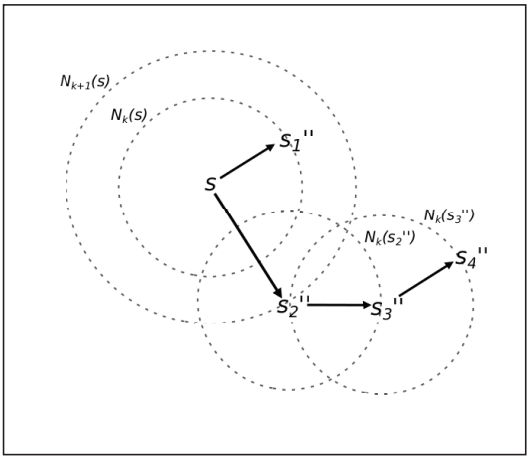

Figura 4.1: Exemplo de busca nas estruturas de vizinhanças pela metaheurística VNS. A busca é iniciada na solução inicial S. Em seguida, gera-se a solução candidata S_1'' dentro da vizinhança $N_k(S)$. É verificado se a solução S_1'' é melhor que a solução S corrente. Como não é, o raio de busca é incrementado e é gerada a solução S_2''. A solução S_2'' é melhor que a solução S, então, S_2'' passa a ser a solução corrente e o raio de busca volta a ser o raio inicial k. Posteriormente, na vizinhança $N_k(S_2'')$, encontra-se a solução S_3''. Como S_3'' é melhor que a solução corrente S_2'', a solução candidata S_3'' passa a ser a solução corrente. Realiza-se a busca na vizinhança da solução S_3'' e é encontrada a solução S_4'' [56].

Quando todos os vértices do nível $L_i(v)$ estiverem numerados, os vértices do nível $L_{i+1}(v)$ são renumerados. Esse processo é repetido para todos os níveis da estrutura de nível enraizada $\mathscr{L}(v)$, até renumerar os vértices do nível $L_{\ell(v)}(v)$.

A heurística inicia por um vértice v aleatório. O vértice v recebe o rótulo 1, ou seja, $s(v) = 1$, e constrói-se a estrutura de nível enraizada $\mathscr{L}(v)$. Os vértices são renumerados por nível da estrutura de nível enraizada $\mathscr{L}(v)$, iniciando-se por um vértice aleatório em cada nível $L_i(v)$, para $1 \leq i \leq \ell(v)$. Mladenović et al. [94] denominaram esse procedimento de busca em largura aleatória.

Para um nível $L_i(v)$ com vértices u_j, com $j = 1, \ldots, |L_i(v)|$, o procedimento escolhe aleatoriamente um índice $j^* \in [1, |L_i(v)|]$, para que u_{j^*} seja o primeiro vértice do nível corrente i a ser renumerado. Por exemplo, caso o nível da estrutura de nível enraizada seja $L_1(v)$ e o vértice u seja o primeiro vértice a ser renumerado, então, u deve receber o rótulo 2, ou seja, $s(u) \leftarrow 2$, porque o vértice $v \in L_0(v)$ recebeu o identificador 1. São renumerados os demais vértices de $L_i(v)$ na ordem

$$\begin{cases} u_{j^*}, u_{j^*+1}, u_{j^*+2}, \ldots, u_{|L_i(v)|}, u_1, \ldots, u_{j^*-1}, & \text{se } j^* \neq 1 \\ u_{j^*}, u_{j^*+1}, \ldots, u_{|L_i(v)|}, & \text{se } j^* = 1 \end{cases}.$$

Repete-se esse passo para cada nível $L_i(v)$ da estrutura de nível enraizada $\mathscr{L}(v)$. Como a construção da estrutura de nível enraizada é pela busca em largura, a complexidade desse procedimento é $\mathcal{O}(|V| + |E|)$.

Mostra-se a sub-rotina para a construção da solução inicial no algoritmo 14. O algoritmo recebe um grafo $G = (V, E)$ com a numeração original.

Algoritmo 14: SoluçãoInicial.

Entrada: grafo $G = (V, E)$;
Saída: solução inicial S;

```
1  início
2      para cada ( v ∈ V ) faça s(v) ← ∅;
3      v ← VerticeAleatório(V); // vértice aleatório de V
       // quantidade de vértices no nível corrente
4      j_c ← 1; // |L_0(v)| = 1
       // vértices na sequência em que são inseridos
5      C_c(j_c) ← v; // no nível corrente de ℒ(v)
6      rotulo ← 1; // identificador atribuído aos vértices
7      j_p ← 0; // quantidade de vértices no próximo nível
8      enquanto ( rotulo ≤ |V| ) faça
9          j* ← InteiroAleatório(1, j_c);
10         para ( i ← 1; i ≤ j_c; i ← i + 1 ) faça
11             para cada ( vértice w ∈ Adj(G, C_c(j*)) ) faça
12                 se ( s(w) = ∅ ) então
13                     j_p ← j_p + 1; // vértices na sequência
14                     C_p(j_p) ← w; // em que são inseridos
15                 fim-se;
16             fim-para-cada;
17             s(C_c(j*)) ← rotulo;
18             rotulo ← rotulo + 1;
19             j* ← j* + 1;
20             se ( j* > j_c ) então j* ← 1;
21         fim-para;
           // o próximo nível se torna o nível corrente
22         para ( i ← 1; i ≤ j_p; i ← i + 1 ) faça C_c(i) ← C_p(i);
23         j_c ← j_p;
24         j_p ← 0;
25     fim-enquanto;
26     retorna S;
27 fim.
```

O procedimento inicializa todos os vértices do grafo como não rotulados na linha 2. O vértice inicial v é inicializado com um vértice aleatório na linha 3.

A variável j_c é o número de vértices no nível corrente da estrutura de nível enraizada $\mathscr{L}(v)$. Essa variável recebe 1 na linha 4 porque $|L_0(v)| = 1$.

O vetor C_c contém os vértices no nível corrente da estrutura de nível $\mathscr{L}(v)$. Por isso, na linha 5, o vértice v é inserido na entrada $j_c = 1$ de C_c. Os vértices ficam armazenados na sequência em que são inseridos em C_c.

A variável *rotulo* é o identificador a ser atribuído aos vértices e é inicializada com 1 na linha 6. A variável j_p é a quantidade de vértices no próximo nível e é inicializada com 0 na linha 7.

A estrutura de repetição *enquanto* externa, nas linhas 8 a 25, executa enquanto todos os vértices não tiverem sido rotulados. Os vértices do nível corrente da estrutura de nível enraizada são rotulados na estrutura de repetição *para*, nas linhas 10 a 21, a partir de $j^* \in [1, j_c]$, escolhido aleatoriamente na linha 9. Na estrutura de repetição *para cada*, nas linhas 11 a 16, os vértices adjacentes ao vértice contido na entrada $C_c(j^*)$, que ainda não foram rotulados (linha 12), são inseridos no próximo nível a ser processado, na linha 14.

O vértice em $C_c(j^*)$ é rotulada na linha 17 e a variável *rotulo* é incrementada na linha 18. O procedimento numera os vértices do nível corrente na sequência $s(C_c(j^*)), s(C_c(j^* + 1)), \ldots, s(C_c(j_c)), s(C_c(1)), s(C_c(2)), \ldots, s(C_c(j^* - 1))$ se $j^* \neq 1$ ou $s(C_c(j^*)), s(C_c(j^* + 1)), \ldots, s(C_c(j_c))$ se $j^* = 1$.

O procedimento faz com que variável j^* indique o próximo vértice de C_c a ser rotulado nas linhas 19 e 20. O vetor C_c recebe o próximo nível na linha 22 e j_c recebe j_p na linha 23. Finalmente, a solução inicial S é retornada na linha 26.

4.3.3 Busca por nova solução candidata

A heurística VNS-band realiza uma busca por nova solução candidata (etapa também chamada de perturbação ou *shake* no contexto de metaheurísticas) ao trocar rótulos dos vértices, a partir de uma solução (ou numeração) S e gera uma solução $S' \in N_k(S)$. Para o entendimento da etapa de busca por nova solução candidata (perturbação), é necessário compreender o conceito de diferença (ou distância) entre soluções candidatas. Distância entre as soluções candidatas, utilizada na etapa de busca por nova solução candidata (perturbação), é mostrada na subseção 4.3.3.1.

A etapa de busca por nova solução candidata (perturbação) é dividida em três procedimentos. O primeiro procedimento de busca por

nova solução candidata, mostrado na subseção 4.3.3.2, é um método que seleciona previamente alguns vértices que possuem as maiores larguras de banda e seus rótulos são trocados. No segundo procedimento de busca por nova solução candidata, mostrado na subseção 4.3.3.3, os rótulos de alguns vértices são rotacionados. O terceiro procedimento, mostrado na subseção 4.3.3.4 seleciona qual dos dois procedimentos anteriores realizará a busca por nova solução candidata.

4.3.3.1 Distância entre as soluções candidatas

A distância k entre duas soluções S e S' é dada pelo número de vértices com rótulos diferentes entre S e S' e decrementa-se 1 dessa diferença. Uma solução S' pertence à vizinhança $N_k(S)$ se a solução S' difere de S em $k+1$ rótulos, ou seja, $\mathfrak{h}(S,S') = k \iff S' \in N_k(S)$, em que $\mathfrak{h}$ é a distância entre S e S'.

A distância $\mathfrak{h}$ pode ser definida como uma distância de Hamming [67]. Essa distância é o número de posições em que duas *strings* de mesmo comprimento diferem entre si. A distância $\mathfrak{h}$ entre as soluções S e S' é definida por $\mathfrak{h}(S,S') = \left(\sum_{v \in V} f(v)\right) - 1$, em que $f(v) = \begin{cases} 1, & \text{se } s(v) \neq s'(v) \\ 0, & \text{caso contrário} \end{cases}$.

Como exemplo, considere uma solução $s(v_1) = 1$, $s(v_2) = 2$, $s(v_3) = 3$, $s(v_4) = 4$ e outra solução $s'(v_1) = 4$, $s'(v_2) = 3$, $s'(v_3) = 2$, $s'(v_4) = 1$, para $V = \{v_1, v_2, v_3, v_4\}$. Portanto, $\mathfrak{h}(S,S') = 3$, porque as numerações S e S' diferem em todos os quatro vértices.

O número de trocas na numeração de S para resultar em S' deve ser menor ou igual a k. No exemplo, para se obter a numeração S', deve-se realizar duas trocas de rótulos, os rótulos de v_1 com v_4 e os rótulos de v_2 com v_3.

4.3.3.2 Primeiro procedimento de busca por nova solução candidata

O primeiro procedimento, inicialmente, seleciona um conjunto de vértices $Y \subseteq V$ com larguras de banda maiores que um valor β', em que $Y = \{v : \beta_v(G_S) \geq \beta'\}$ e $|Y| \geq k$, para que os rótulos dos vértices sejam trocados. Desse modo, são selecionados os vértices com as maiores larguras de banda.

Em seguida, são selecionados um vértice aleatório $u \in Y$ e seu vértice crítico v. Um vértice crítico v de u é um vértice adjacente a u com a maior diferença de numeração, ou seja, $|s(u) - s(v)| = \beta_u(G_S)$ (veja as definições 1.39 e 1.40, na página 16 da seção 1.3).

O próximo passo é encontrar um vértice w, tal que $s_{\min}(u) \leq s(w) \leq s_{\max}(u)$, em que $s_{\max}(u) = \max_{\{u,u'\} \in E}[s(u')]$ e $s_{\min}(u) = \min_{\{u,u'\} \in E}[s(u')]$. Isso significa que o rótulo do vértice w não pode aumentar $\beta_u(G_S)$. Além disso, é preciso analisar também as adjacências do vértice w de forma que trocar o rótulo do vértice v com o rótulo do vértice w não altere $\beta_v(G_S)$. Para isso, verifica-se $\max[|s(v) - s_{\min}(w)|, |s_{\max}(w) - s(v)|]$. Com isso, busca-se colocar v com um novo rótulo de forma que $\beta_v(G_S)$ seja o menor possível.

Mostra-se a primeira sub-rotina de busca por nova solução candidata no algoritmo 15. O algoritmo recebe o grafo $G_S = (V, E)$ com a numeração S corrente, o raio k da vizinhança da solução corrente e o limiar β'. O valor β', passado para o algoritmo 15, é de forma que $|Y| \geq k$.

Na estrutura de repetição *para* nas linhas 4 a 27, são selecionados o vértice $u \in Y$ aleatório na linha 5, o vértice crítico v (nas linhas 6 a 13) de u e o vértice w (nas linhas 14 a 23) para trocar com v (nas linhas 24 a 26). Repete-se esse processo k vezes. A sub-rotina retorna a renumeração $S' \in N_k(S)$ na linha 28.

4.3.3.3 Segundo processo da busca por nova solução candidata

Segundo Mladenović et al. [94], utilizar apenas o algoritmo 15 para escapar de um mínimo local não é suficiente em algumas situações. Portanto, os autores utilizaram uma segunda sub-rotina para busca por nova solução candidata, baseada na rotação de duas vizinhanças [102].

Na sub-rotina de rotação de vizinhanças, são trocados os rótulos de todos os vértices que pertencem a um intervalo preestabelecido. Inicialmente, para delimitar o intervalo de troca, são escolhidos dois índices b e f com um terceiro índice m entre eles, ou seja, $1 \leq b < m < f \leq |V|$. São trocados os rótulos dos vértices no intervalo $[b, f]$, deslocando-os m posições, ou seja, o rótulo de $s^{-1}(b)$ passará a ocupar a posição de $s^{-1}(b+m)$, se $b+m \leq f$. Para um índice i, em que $i+m > f$, o rótulo de $s^{-1}(i)$ será deslocado para a posição de $s^{-1}(b + |f - (i+m)|)$.

A sub-rotina de rotação é mostrada no algoritmo 16. O algoritmo recebe um grafo $G_S = (V, E)$ com numeração S e o o raio k da vizinhança da solução corrente.

Na estrutura de repetição *para* nas linhas 3 a 12, primeiramente, são escolhidos aleatoriamente os índices b e f do intervalo de troca, nas linhas 4 e 5. Os índices b e f são estabelecidos aleatoriamente, em que, na função *min*, são utilizados os parâmetros 20 e $\frac{\beta(G_S)}{2}$. Esses parâmetros foram sugeridos por Mladenović et al. [94], pois gerou resultados satisfatórios nos testes realizados.

Algoritmo 15: BuscaSolução1.

Entrada: grafo $G_S = (V, E)$ com numeração S; raio da vizinhança k; limiar β';
Saída: renumeração S';

```
1  início
2      S' ← S;
       // retorna um conjunto de vértices com β_v(G_S) ≥ β',
       // em que β' é um limiar para que |Y| ≥ k
3      Y ← SelecionaVerticesParaTroca(S, β');
4      para ( i ← 1; i ≤ k; i ← i + 1 ) faça
5          u ← VérticeAleatório(Y);
6          β_u ← −∞;
           // seleciona um vértice v, em que
           |s(u) − s(v)| = β_u(G_S)
7          para cada ( vértice v' ∈ Adj(G, u) ) faça
8              β_uv' ← |s(u) − s(v')|;
9              se ( β_uv' > β_u ) então
10                 β_u ← β_uv';
11                 v ← v';
12             fim-se;
13         fim-para-cada;
14         menor ← +∞;
15         para cada ( w' ∈ V ) faça
16             se ( s_min(u) ≤ s(w') ≤ s_max(u) ∧
               s_min(w) ≤ s(v) ≤ s_max(w) ) então
17                 maximo ← max(|s_max(w') − s(v)|, |s(v) − s_min(w')|);
18                 se ( maximo < menor ) então
19                     menor ← maximo;
20                     w ← w';
21                 fim-se;
22             fim-se;
23         fim-para-cada;
24         aux ← s(v); // troca os rótulos dos vértices v e w
25         s(v) ← s(w);
26         s(w) ← aux;
27     fim-para;
28     retorna S';
29 fim.
```

Na linha 7, estabelece-se m, também aleatório, no intervalo (b, f). Na estrutura de repetição *para* nas linhas 8 a 11, realiza-se a rotação no intervalo $[b, f]$. A sub-rotina retorna a numeração $S' \in N_k(S)$ na linha 13.

Algoritmo 16: BuscaSolução2.

Entrada: grafo $G_S = (V, E)$ com numeração S; raio da vizinhança k;
Saída: renumeração S';

```
1  início
2      S' ← S;
3      para ( i ← 1; i ≤ k; i ← i + 1 ) faça
4          b ← InteiroAleatório(1, |V|);
5          f ← b+ InteiroAleatório(2, min [20, β(G_S)/2]);
6          se ( f > |V| ) então f ← |V|;
7          m ← InteiroAleatório(b + 1, f − 1);
8          para ( j ← b; j ≤ f; j ← j + 1 ) faça
9              se ( j ≥ b + m ) então s'(s'^{-1}(j)) ← j − m;
10             senão s'(s'^{-1}(j)) ← j + f − b − m + 1;
11         fim-para;
12     fim-para;
13     retorna S';
14 fim.
```

4.3.3.4 Escolha do processo da busca por nova solução candidata

A sub-rotina de escolha utilizada para decidir qual sub-rotina de busca por nova solução candidata utilizar é mostrada no algoritmo 17. O algoritmo recebe um grafo $G_S = (V, E)$ com numeração S, o raio k da vizinhança da solução corrente, o raio inicial k_{min}, o fator de incremento k_{incr} para aumentar o raio de busca na vizinhança, o raio máximo k'_{max} e o limiar β'.

Algoritmo 17: BuscaSolução.

Entrada: grafo $G_S = (V, E)$ com numeração S; raio da vizinhança k; raio inicial k_{min}; fator k_{incr} de incremento de raio de busca na vizinhança; raio máximo k'_{max}; limiar β';
Saída: renumeração S';

```
1  início
       // algoritmo 15 ou 16
2      se ( k ≤ k'_max ) então  S' ← BuscaSolução1(G_S, k, β');
3      senão  S' ← BuscaSolução2(G_S, ⌊(k − k_min)/k_incr⌋);
4      retorna S';
5  fim.
```

O algoritmo 17 determina se será utilizado o algoritmo de busca por nova solução candidata 15 ou 16. No algoritmo 17, o valor k'_{max} é definido de acordo com a necessidade de alteração na solução candidata

corrente para se "escapar de um vale" no espaço de soluções. Se $k \leq k'_{max}$, utiliza-se o algoritmo 15; caso contrário, utiliza-se o algoritmo 16.

Mladenović et al. [94] verificaram que o valor k, quando passado ao algoritmo 16, gerava alterações desnecessárias, pois necessita-se de $k = \left\lfloor \frac{k - k_{min}}{k_{incr}} \right\rfloor$, em que k_{min} é o raio inicial para a busca na vizinhança e k_{incr} é o fator de incremento do raio de busca de vizinhança a cada passo. Quando a busca "não consegue escapar" de uma região próxima a um mínimo local, o raio da vizinhança $N_k(S)$ é incrementado pelo fator k_{incr}, possibilitando encontrar novas soluções.

4.3.4 Busca local

A busca local da heurística VNS-band é baseada na busca local proposta por Lim et al. [85]. Essa busca local é basicamente a mesma da heurística de Lim et al. [86] que é descrita na subseção 4.2.2. Segundo Mladenović et al. [94], a busca local proposta por Lim, Rodrigues e Xiao [85] apresentou resultados melhores na redução de largura de banda do que a busca local original utilizada pela meta-heurística VNS [93].

Nesta etapa, um conjunto de vértices críticos do grafo é selecionado para que seus rótulos sejam trocados. Um vértice $v \in V$ é um vértice crítico de $G_S = (V, E)$ se existe um vértice $u \in V$ adjacente a v, em que $\{u, v\} \in E$ é uma *aresta crítica* de $G_S = (V, E)$. Uma aresta $\{u, v\} \in E$ é crítica se $|s(u) - s(v)| = \beta(G_S)$. O conjunto E_c, conjunto de arestas críticas de $G = (V, E)$, é definido por $E_c = \{\{u, v\} \in E : |s(u) - s(v)| = \beta(G_S)\}$. O conjunto de vértices críticos V_c de $G_S = (V, E)$ é todo vértice de arestas de E_c. (Veja também as definições 1.39 e 1.40, na página 16, na seção 1.3.)

Para a aplicação da busca local, é definida a vizinhança reduzida $\mathrm{N}_S(v)$ para a troca de rótulo do vértice crítico v como $\mathrm{N}_S(v) = \{u : (\{u, v\} \in E_c) \ |med(v) - s(u)| < |med(v) - s(v)|\}$, em que $med(v) = \left\lfloor \frac{s_{\max}(v) + s_{\min}(v)}{2} \right\rfloor$, $s_{\max}(u) = \max_{\{u,u'\} \in E}[s(u')]$ e $s_{\min}(u) = \min_{\{u,u'\} \in E}[s(u')]$, como visto também na seção anterior. O conjunto de vértices para a aplicação da busca local, ou o conjunto de vizinhanças reduzidas para a troca de numeração em relação à solução S, é definido como $N(S) = \bigcup_{v \in V_c} \mathrm{N}_S(v)$.

Como a etapa da busca local é depois da etapa de busca por nova solução candidata (perturbação), a solução corrente é S' neste passo. Com $N(S')$, forma-se um conjunto de vértices apropriados para a troca. A busca local é aplicada no conjunto $N(S')$. Troca-se a numeração do vértice crítico v com cada numeração dos vértices $u \in \mathrm{N}_{S'}(v) \subseteq N(S')$ até que uma melhora em relação à solução corrente S' seja constatada. Com isso, gera-se a solução S''.

4.3.5 Permutação de vizinhança

A etapa de permutação de vizinhança consiste em decidir sobre duas questões: quando permutar ou não permutar a solução corrente e qual será a próxima vizinhança. Para isso, muitas estratégias foram propostas e Hansen e Mladenović [69] apresentaram uma revisão.

1. Quando permutar (mover) ou não: para a primeira questão, necessita-se saber qual critério utilizar para permutar a solução corrente e quando não permutar. Na redução de largura de banda, são utilizados três critérios para permutar a solução S corrente para a solução S''.

   (a) Verifica-se qual solução possui a menor largura de banda. Se $\beta(G_S) > \beta(G_{S''})$, então, a solução candidata S'' passa a ser a solução corrente, ou seja, $S \leftarrow S''$. Em caso contrário, S continua como a solução corrente.

   (b) Se $\beta(G_S) = \beta(G_{S''})$, então, avalia-se o número de vértices críticos de cada solução. Se $|V_c(S)| > |V_c(S'')|$, então, a solução candidata S'' passa a ser a solução corrente. Em caso contrário, a solução candidata S continua como a solução corrente.

   (c) Se $\beta(G_S) = \beta(G_{S''})$ e $|V_c(S)| = |V_c(S'')|$, então, avalia-se a diferença (distância) entre as soluções S e S''. Troca-se a solução S pela solução candidata S'' se $\mathfrak{h}(S, S'') > \mathfrak{t}$, em que $\mathfrak{t}$ é um parâmetro constante que, segundo Mladenović et al. [94], nos testes realizados, obteve-se resultados satisfatórios para $\mathfrak{t} = 10$. Este terceiro critério baseia-se em uma variação da meta-heurística VNS proposta por Hansen e Mladenović [68], em que se admite a permutação da solução S corrente por uma solução candidata S' com largura de banda maior, se a solução candidata S' estiver relativamente distante da solução candidata S. Porém, para o problema de redução de largura de banda, se as soluções candidatas S e S'' possuírem a mesma largura de banda e o mesmo número de vértices críticos, permuta-se de S para a solução candidata S'' apenas se $\mathfrak{h}(S, S'') > \mathfrak{t}$.

2. Próxima vizinhança: a segunda questão é descobrir qual será a vizinhança no caso de haver permutação da solução corrente e como alterar a vizinhança corrente quando não houver permutação. Os parâmetros k_{min}, k_{incr} e k_{max} são utilizados para determinar se o raio de busca na vizinhança será incrementado, se voltará ao raio inicial ou se excedeu o limite.

- O parâmetro k_{min} é o raio inicial para a busca na vizinhança. Se a solução S corrente é trocada para a solução S'', então, atualiza-se o raio da busca de $N_k(S'')$ para k_{min}, ou seja, utiliza-se $N_{k_{min}}(S'')$. Um exemplo pode ser visto na figura 4.1, na página 79.
- O parâmetro k_{incr} é o incremento do raio de busca na vizinhança quando uma troca não é realizada. Se após a busca por nova solução candidata (perturbação) e a busca local não houver melhorias em relação à solução S corrente, então, estabelece-se $k \leftarrow k + k_{incr}$. Com isso, se $k < k_{max}$, os passos de busca são repetidos, em que o parâmetro k_{max} é o raio máximo em que é realizada a busca.

Mostra-se a sub-rotina que verifica se haverá ou não a permutação da solução S corrente pela solução candidata S'' no algoritmo 18. A sub-rotina recebe o grafo $G_S = (V, E)$ com a solução S corrente, a solução candidata S'' e o fator de distância entre soluções candidatas $\mathfrak{t} = 10$. O algoritmo retorna verdadeiro se é viável ou falso se não é viável realizar a permutação da solução S corrente pela solução S''.

Algoritmo 18: *Troca*.

Entrada: grafo $G_S = (V, E)$ com numeração S; renumeração S''; fator de distância entre soluções candidatas $\mathfrak{t}$;
Saída: *verdadeiro* ou $falso$;

```
1 início
    // permuta ou não a solução corrente
2   retorna (β(G_S) > β(G_S'')) ∨ (β(G_S) = β(G_S'') ∧
    |V_c(S)| > |V_c(S'')|) ∨ (β(G_S) = β(G_S'') ∧ |V_c(S)| = |V_c(S'')| ∧
    ɬ(S, S'') > t);
3 fim.
```

4.3.6 Heurística VNS-band

Nesta subseção, apresenta-se a heurística VNS-band completa. A heurística VNS-band é mostrada no algoritmo 19. Mladenović et al. [94] utilizaram o tempo de execução como critério de parada.

O algoritmo recebe o grafo $G = (V, E)$, o raio inicial k_{min}, o fator k_{incr} de incremento do raio de busca na vizinhança, o raio máximo k_{max}, o raio pré-definido k'_{max}, o limiar β', o fator de distância entre soluções candidatas $\mathfrak{t}$ e o tempo máximo t_{max}. Mladenović et al. [94] utilizaram, em todos os testes, os seguintes valores dos parâmetros: $k_{min} = k_{incr} = 5,\ k'_{max} = 100,\ k_{max} = 200,\ \mathfrak{t} = 10$ e $t_{max} = 500s$.

Algoritmo 19: VNS-band.

Entrada: grafo $G = (V, E)$; raio inicial k_{min}; fator k_{incr} de incremento de raio de busca na vizinhança; raio máximo k_{max}; raio pré-definido k'_{max}; limiar β'; distância entre soluções candidatas $\mathfrak{t}$; tempo máximo t_{max};

Saída: renumeração final S^*;

```
1  início
       // β_min terá a menor largura de banda encontrada
2      β_min ← +∞;
3      t_ini ← TempoUCP();
       // número máximo de iterações utilizado na estrutura
4      i_max ← ⌊(k_max − k_min)/k_incr⌋; // de repetição nas linhas 10-21
5      repita
           // solução candidata inicial a partir da numeração
6          S ← SoluçãoInicial(G_{S_o}); // original: algoritmo 14
7          S ← BuscaLocal(S); // explicada na subseção 4.3.4
8          S* ← S;
9          k ← k_min;
10         i ← 1;
11         enquanto ( i ≤ i_max ) faça
               // algoritmo 17
12             S' ← BuscaSolução(G_S, k, k_min, k_incr, k'_max, β');
13             S'' ← BuscaLocal(S'); // veja a subseção 4.3.4
               // algoritmo 18
14             se ( Troca(G_S, S'', t) = verdadeiro ) então
15                 S ← S''; // aceita a solução candidata S''
16                 k ← k_min; // raio k retorna para k_min
17                 i ← 1; // alterações realizadas i_max vezes
18             senão
19                 k ← k + k_incr; // incrementa o raio de busca na
20                 i ← i + 1; // vizinhança, realizada i_max vezes
21             fim-se;
22         fim-enquanto;
23         se ( β(G_S) < β_min ) então
24             β_min ← β(G_S);
25             S* ← S; // S* é a numeração final
26         fim-se;
27         t_dec ← TempoUCP();
28     até que (t_dec − t_ini ≥ t_max) ;
29     retorna S*;
30 fim.
```

Nas linhas 2 a 4, as variáveis β_{min}, t_{ini} e i_{max} são inicializadas, respectivamente. A variável β_{min} terá a menor largura de banda encontrada. A variável t_{ini} é o tempo inicial e i_{max} recebe $\left\lfloor \frac{k_{max}-k_{min}}{k_{incr}} \right\rfloor$.

Na estrutura de repetição *enquanto* externa, nas linhas 5 a 28, a solução inicial S é construída pelo algoritmo 14 na linha 6. Após a construção da solução inicial S pelo algoritmo 14, aplica-se a busca local nessa solução na linha 7 e a o raio de vizinhança k é inicializado na linha 9.

Na estrutura de repetição *para* interna nas linhas 11 a 22, a rotina aplica a alteração na solução S corrente na linha 12. Assim, a solução candidata S' é gerada. Em seguida, a rotina aplica a busca local na solução S', gerando a solução candidata S'' na linha 13.

Ainda na estrutura de repetição *para* interna nas linhas 11 a 22, a rotina verifica se haverá troca da solução S corrente pela solução candidata S'' na estrutura condicional na linha 14 por meio da sub-rotina $Troca()$, mostrada no algoritmo 18. Caso troque, a solução corrente passa a ser a solução candidata S'' e atualiza-se o raio de vizinhança k para k_{mim} nas linhas 15 e 16, respectivamente. Caso contrário, incrementa-se o raio de vizinhança k com o fator k_{incr} de incremento de busca na vizinhança na linha 19. Isso é realizado i_{imax} vezes (veja a linha 20).

Na estrutura condicional nas linhas 23 a 26, a rotina verifica se a largura de banda da solução corrente S é menor do que a menor largura de banda já encontrada, armazenada em β_{min} (inicializada na linha 2 com $+\infty$). Caso seja, a rotina armazena a largura de banda em β_{min} na linha 24 e a numeração correspondente em S^* na linha 25.

A sub-rotina $TempoUCP$ na linha 27 retorna o tempo decorrido. O algoritmo para quando o tempo de processamento excede o limite t_{max} (linha 28). O algoritmo retorna a renumeração S^* do grafo na linha 29.

4.4 Heurística DRSA

A heurística *Dual Representation Simulated Annealing* (DRSA) foi projetada pela meta-heurística recozimento simulado (*Simulated Annealing*). A heurística utiliza internamente duas representações do problema de redução de largura de banda.

1. Para o grafo $G = (V, E)$, a primeira representação considera a numeração S, de forma consistente com a representação utilizada neste texto, em que $s(v)$ retorna um número natural no intervalo $[1, |V|]$, como o rótulo (ou identificador) do vértice $v \in V$.

2. A segunda representação interpreta a função inversa $s^{-1}(v)$ como o vértice $v \in V$ identificado pelo número natural no intervalo

$[1, |V|]$.

Essas representações são utilizadas em conjunto com uma função de vizinhança composta de três operadores para alteração (etapa de perturbação) da solução candidata corrente. As representações foram utilizadas dessa forma porque dois dos três operadores operam sobre a primeira representação e o terceiro operador opera sobre a segunda representação.

Esta seção está dividida como descrito a seguir. Comenta-se a meta-heurística *Simulated Annealing* na subseção 4.4.1. A estrutura da heurística DRSA é mostrada na subseção 4.4.2. A função de vizinhança é abordada na subseção 4.4.3. Finalmente, a função de avaliação das soluções candidatas é apresentada na subseção 4.4.4.

4.4.1 Meta-heurística *Simulated Annealing*

A meta-heurística *Simulated Annealing* é uma técnica probabilística para projetar métodos heurísticos que retornam soluções aproximadas à solução ótima para problemas de otimização. A técnica é uma analogia com recozimento em metalurgia, uma técnica que envolve o processo de aquecimento e resfriamento controlado para alterar as propriedades físicas de um material, obtendo-se uma estrutura cristalina altamente estável.

O processo de aquecimento e resfriamento controlado para alterar as propriedades físicas de um material consiste em aumentar a temperatura de forma que o material derreta. Em seguida, a temperatura é diminuída até que o material derretido alcance o seu estado de equilíbrio. Nesse estado, as partículas do metal estão organizadas de tal forma que a energia do sistema é mínima.

4.4.2 Estrutura da heurística DRSA

Os parâmetros para a simulação de escalonamento de temperatura são: temperaturas inicial T_0 e final T_f, a taxa de resfriamento $\alpha \in (0, 1)$ e os tamanhos inicial L e final L_f da cadeia de Markov. A temperatura t é inicializada com T_0 no início do processamento. Durante o processamento, a temperatura é reduzida com $t = \alpha \cdot t$ até alcançar T_f.

Para simular as mudanças de estado no material, o algoritmo mantém um estado corrente x com energia E_x. O próximo estado y, com energia E_y é obtida ao alterar o estado corrente x. Se $E_y < E_x$, então, y se torna o novo estado corrente; caso contrário, o estado y é aceito com probabilidade $exp(-\frac{E_y - Ex}{t})$. A meta-heurística também requer o parâmetro L, que é o tamanho da cadeia de Markov, que define o número de alterações (movimentos ou perturbações) na solução candidata corrente realizadas com a mesma temperatura corrente t.

Mostra-se a heurística DRSA no algoritmo 20. O algoritmo recebe o grafo $G_S = (V, E)$ conexo, a temperatura inicial $T_0 = 1000$, a temperatura final $T_f = 10^{-7}$, a taxa de resfriamento $\alpha = 0,99$, o tamanho inicial da cadeia de Markov $L = 40$ e o tamanho final da cadeia de Markov $L_f = 10 \cdot |V| \cdot |E|$.

Algoritmo 20: DRSA [113].

Entrada: grafo $G_S = (V, E)$ conexo; temperatura inicial T_0; temperatura final T_f; taxa de resfriamento α; tamanho inicial da cadeia de Markov L; tamanho final da cadeia de Markov L_f;
Saída: renumeração final S^*;

```
1  início
2      S ← SolucaoInicialAleatória(G_S);
3      S* ← S;
4      γ ← exp( log(α) · (log(L_f) − log(L)) / (log(T_f) − log(T_0)) );
5      t ← T_0; // temperatura inicial T_0
6      enquanto ( t > T_f ) faça
7          para ( i de 1 a L ) faça
8              S' ← g(S); // algoritmo 21
9              se ( Aleatório(0,1) < exp( −(f(S') − f(S)) / t ) ) então
10                 S ← S';
11                 se ( f(G_S) < f(G_S*) ) então S* ← S;
12             fim-se;
13         fim-para;
           // Se S* ≠ S, então, não houve melhoria em S*
14         se ( S* ≠ S ) então
15             t ← t · α; // α é a taxa de resfriamento
16             L ← L · γ;
17         fim-se;
18     fim-enquanto;
19     retorna S*;
20 fim.
```

Uma solução inicial aleatória S é dada na linha 2. A melhor solução S^* encontrada é inicializada com a solução inicial S na linha 3.

Na linha 4, $\gamma > 1$ é o fator de incremento do tamanho da cadeia de Markov. Esse fator de incremento é estabelecido de forma que, quando a temperatura t alcance a temperatura final T_f na linha 15, o tamanho da cadeia de Markov também alcance o tamanho final da cadeia de Markov L_f, na linha 16.

O algoritmo 20 é baseado na estrutura de repetição *enquanto*, mostrada nas linhas 6 a 18. Essa estrutura de repetição externa executa

enquanto a temperatura corrente t não alcança a temperatura final T_f. Basicamente, esse laço de repetição externo da heurística DRSA consiste de uma sequência de execuções do algoritmo Metropolis [92], com a mesma solução inicial dada na linha 2 e com decremento lento e progressivo da temperatura t.

A estrutura de repetição *para* interna, mostrada nas linhas 7 a 13, executa L perturbações na solução candidata corrente com a função de vizinhança $g(x)$, na linha 8, que utiliza operadores específicos para encontrar nova solução candidata S' na vizinhança da solução S. A função de vizinhança $g(x)$ é mostrada no algoritmo 21. Se não há melhoria na solução, avaliada pela função $f(G_S)$ (descrita na subseção 4.4.4, na página 94), na estrutura condicional nas linhas 9 a 12, então, a temperatura t é resfriada pela taxa α, na linha 15, e o tamanho da cadeia de Markov L é aumentado pelo fator γ, na linha 16. A solução S^* é retornada na linha 19.

4.4.3 Função de vizinhança

Mostra-se a função de vizinhança $g(x)$ no algoritmo 21. A heurística DRSA utiliza a função de vizinhança g para perturbar a solução corrente e obter uma nova solução.

Algoritmo 21: g [113].

Entrada: numeração S;
Saída: numeração S';

```
1 início
2 |  a ← 0,6;
3 |  b ← 0,2;
4 |  p ← Aleatório(0,1);
5 |  se ( p ≤ a ) então  S' ← REX(S);
6 |  senão se( p ≤ a + b ) então  S' ← NEX(S);
7 |  senão  S' ← ROT(S);
8 |  retorna S';
9 fim.
```

Como exemplo das representações utilizadas na heurística DRSA, considere o grafo mostrado na figura 1.1. Considere que a cada vértice esteja associado um número natural no intervalo $[1, |V|]$ como sua posição na forma mostrada nas primeira e segunda linhas da tabela 4.1. Dessa forma, como exemplos, $s^{-1}(9) = 1$, $s^{-1}(2) = 2$, ..., $s^{-1}(4) = 10$. A função de vizinhança consiste de três operadores de perturbação, descritos a seguir.

- O operador *REX* (*Random Exchance Operator*) opera sobre a

vértice	1	2	3	4	5	6	7	8	9	10
$s(v)$	9	2	6	1	7	5	3	8	10	4
REX	9	2	6	**4**	7	5	3	8	10	**1**
NEX	9	2	6	1	7	**3**	**5**	8	10	4
ROT	9	**5**	6	1	7	**4**	**2**	8	10	**3**

Tabela 4.1: Numeração e rótulos do grafo mostrado na figura 1.1 antes e após execuções dos operadores REX para os vértices 4 e 10, NEX para a aresta $\{6,7\} = \{s^{-1}(5), s^{-1}(3)\}$ e ROT com $i = 2$ e $j = 5$.

representação S^{-1}. Esse operador seleciona aleatoriamente dois vértices i e j e troca seus rótulos.

- O operador NEX (*Neighbor Exchange Operator*) também opera sobre a representação S^{-1}. Este operador seleciona aleatoriamente um vértice $v \in V$ e troca seu rótulo com o rótulo de um de seus vértices adjacentes, também selecionado aleatoriamente.

- O operador ROT (*Rotation Operator*) [102] opera sobre a representação S. O operador seleciona dois rótulos i e j, com $i < j$. O vértice com rótulo i passa a ter rótulo j e os vértices com rótulos de $i+1$ a j têm seus rótulos decrementados de 1. Outra possibilidade é o vértice com rótulo j passar a ter rótulo i e os vértices com rótulos de i a $j-1$ terem seus rótulos incrementados de 1.

Considere exemplos de execuções dos operadores no grafo mostrado na figura 1.1. Como exemplo para o operador REX, suponha que as escolhas aleatórias sejam os vértices 4 e 10. Como exemplo para o operador NEX, suponha que a escolha aleatória seja o vértice 6, com $s(6) = 5$ e, então, entre os vértices adjacentes ao vértice 6, seja selecionado aleatoriamente o vértice 7, com $s(7) = 3$. Como exemplo para o operador ROT, suponha que a escolha seja para os rótulos 2 e 5. São mostradas as numerações resultantes na tabela 4.1.

4.4.4 Função de avaliação das soluções candidatas

A função de avaliação, utilizada na heurística DRSA, é $f(G_S) = \beta(G_S) + \delta(G_S)$, em que

$$\delta(G_S) = \frac{\frac{\frac{\frac{d_0(G_S)}{\nu(0)} + d_1(G_S)}{\nu(1)} + d_2(G_S)}{\nu(2)} + \cdots}{\cdots} + d_{n-2}(G_S)}{\nu(n-2)} + d_{n-1}(G_S)}{\nu(n-1)} \tag{4.4.1}$$

é a parte fracionária, computada a partir do conjunto $D(G_S)$, mostrado a seguir. A inclusão da parte fracionária $\delta(G_S)$ permite discriminar melhor as soluções candidatas S do que somente utilizar a largura de banda $\beta(G_S)$. Dicriminar melhor as soluções candidatas permite "escapar de vales" no espaço de soluções.

Sejam o grafo não direcionado $G_S = (V, E)$ com numeração S, $V = \{v_1, v_2, \ldots, v_n\}$ e o conjunto $D(G_S) = \{d_0(G_S), d_1(G_S), \ldots, d_{n-1}(G_S)\}$, composto de $n = |V|$ elementos. O elemento $d_i(G_S)$ é o número de arestas em que a diferença absoluta entre os rótulos dos vértices da aresta é i. O elemento $d_i(G_S) = \mathfrak{f}_{G_S}(\{v_j, v_k\})$, para $j = n, n-1, \ldots, i+1$ e $k = j - i$, em que $\mathfrak{f}_{G_S}(\{v_j, v_k\}) = \begin{cases} 1, & \text{se } \{v_j, v_k\} \in E \\ 0, & \text{se } \{v_j, v_k\} \notin E \end{cases}$.

Considerando-se os elementos à esquerda da diagonal principal da matriz de adjacências A_S do grafo não direcionado $G_S = (V, E)$, $d_i(A_S)$ é o número de coeficientes não nulos na i-ésima sub-diagonal a partir da diagonal principal. Portanto, o elemento $d_i(A_S) = \mathfrak{f}_{A_S}(a_{jk})$, para $j = n, n-1, \ldots, i+1$ e $k = j - i$, em que $\mathfrak{f}_{A_S}(a_{jk}) = \begin{cases} 1, & \text{se } a_{jk} \neq \emptyset \\ 0, & \text{se } a_{jk} = \emptyset \end{cases}$ e $\emptyset$ é a representação de nulo.

O elemento $d_i(G_S)$ avalia a existência de $n-i$ arestas, assim como a i-ésima sub-diagonal, a partir da diagonal principal, tem $n-i$ entradas na matriz de adjacências de um grafo, para $i \in [0, n-1]$. Portanto, o elemento $d_i(G_S)$ pode ter $n - i + 1$ valores: $0, 1, \ldots, n - i$.

Seja o vetor $\mathcal{V}$ composto de $n = |V|$ entradas $\nu_0, \nu_1, \ldots, \nu_{n-1}$. A entrada ν_i é o número de possibilidades que o elemento $d_i(G_S)$ pode ter. Dessa forma, $\nu_0 = n + 1, \nu_1 = n, \ldots, \nu_{n-1} = 2$. Assim, define-se $\nu : \{0, 1, \ldots, n-1\} \to \{n+1, n, \ldots, 2\}$ como a bijeção $\nu(i) = n + 1 - i$, para $i \in [0, n-1]$.

Como exemplo, considere o grafo não direcionado $G_S = (V, E)$ composto de $V = \{v_1, v_2, v_3, v_4\}$ e $E = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_4\}, \{v_3, v_4\}\}$. Portanto:

- $d_0(G_S) = 0$, pois o grafo não contém nenhum vértice conectado a si próprio,
- $d_1(G_S) = 1$, pois $\{v_4, v_3\} \in E$ e $\{v_3, v_2\}, \{v_2, v_1\} \notin E$,
- $d_2(G_S) = 2$, pois $\{v_4, v_2\}, \{v_3, v_1\} \in E$,
- $d_3(G_S) = 1$, pois $\{v_4, v_1\} \in E$.

Assim, $\delta(G_S) = \frac{\frac{\frac{0}{5}+1}{\frac{4}{3}}+2}{2}+1 = 0,875$. Mostra-se a computação da parte fracionária $\delta(G_S)$ na linha 3 do algoritmo 22, conforme mostrado na equação (4.4.1).

Algoritmo 22: Mapeamento do vetor $d(G_S)$ para um valor $\delta(G_S) \in [0, 1)$ normalizado [113].

Entrada: grafo $G_S = (V, E)$ com numeração S; vetor $d(G_S)$;
Saída: $\delta \in [0, 1)$;
1 **início**
2 $\quad \delta \leftarrow 0$;
3 $\quad$ **para** ($i \leftarrow 0$; $i \leq \beta(G_S)$; $i \leftarrow i + 1$) **faça** $\delta \leftarrow \frac{\delta + d_i(G_S)}{\nu(i)}$;
4 $\quad$ **retorna** δ;
5 **fim.**

Para evitar que a parte fracionária $\delta(G_S)$ tenha valores muito pequenos, $\delta(G_S)$ é computado até $d_{\beta(G_S)}(G_S)$, ignorando-se os termos $d_{\beta(G_S)+1}(G_S), \ldots, d_{n-1}(G_S)$. O algoritmo 22 retorna δ na linha 4.

A equação (4.4.1) pode ser reescrita de forma recursiva como

$$\begin{cases} \delta_0(G_S) = \frac{d_0(G_S)}{\nu(0)} \\ \delta_n(G_S) = \frac{\delta_{n-1}(G_S) + d_{n-1}(G_S)}{\nu(n-1)} \end{cases} .$$

Com essa recorrência, o algoritmo 23 é uma versão recursiva do algoritmo 22. O algoritmo 23 pode ser invocado com $\delta(G_S, d_n, n)$, para $n = |V|$, ou $\delta(G_S, d_{\beta(G_S)+1}, \beta(G_S) + 1)$, para considerar apenas os termos até $\beta(G_S)$, como no algoritmo 22.

Algoritmo 23: δ: versão recursiva do algoritmo 22.

Entrada: grafo $G_S = (V, E)$ com numeração S; vetor $d(G_S)$; inteiro positivo n;
Saída: $\delta \in [0, 1)$;
1 **início**
2 $\quad$ **se** ($n = 0$) **então retorna** $\frac{d_0(G_S)}{\nu(0)}$;
3 $\quad$ **retorna** $\frac{\delta(G_S, d_{n-2}, n-2) + d_{n-1}}{\nu(n-1)}$;
4 **fim.**

Como mencionado na seção 4.1, a heurística DRSA [113] é o atual algoritmo meta-heurístico no estado da arte para matrizes com dimensões até aproximadamente 1.000. Uma implementação da heurística DRSA está disponível em https://github.com/Pigzaum/drsa_bmp.

Apêndice A

Exemplos de áreas de aplicação

Exemplos de áreas de aplicação do problema de redução de largura de banda de matrizes: análise numérica, dinâmica de fluidos computacional, problemas em eletromagnetismo, problemas em estatística, na simulação de circuito no domínio do tempo/frequência, na modelagem dinâmica e estática de processos químicos, criptografia, problemas em magneto-hidrodinâmica, sistemas de energia elétrica, química quântica, mecânica quântica, mecânica estrutural (projeto e modelagem de edifícios, navios, aeronaves, partes do corpo humano etc.), aerodinâmica, termodinâmica, transferência de calor, reconstruções em ressonância magnética, vibroacústica, otimização linear e não linear, portfólios financeiros, simulação de processos de semicondutores, modelagem em economia, modelagem de reservatórios de petróleo, astrofísica, propagação de rachaduras, classificação de páginas do Google, visão computacional 3D, posicionamento de torres de telefones celulares, tomografia, simulação de multicorpos, redução de modelos, nanotecnologia, radiação acústica, teoria do funcional de densidade, atribuição quadrática, propriedades elásticas de cristais, processamento de linguagem natural, eletroforese de DNA, cálculo de autovalores, programação linear, problemas de mínimos quadrados lineares, problemas de sistemas de controle, computação gráfica, aprendizado de máquina, sistemas de transmissão de energia, geofísica etc.

Bibliografia

[1] ABREU, A. A. A. M.; BERNARDES, J. A. B.; GONZAGA DE OLIVEIRA, S. L. A comparison of pseudoperipheral vertex finders for the reverse Cuthill-Mckee method. In: *Anais do LII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. Rio de Janeiro, RJ: Sociedade Brasileira de Pesquisa Operacional, 2020.

[2] ALWAY, G. G.; MARTIN, D. W. An algorithm for reducing the bandwidth of a matrix of symmetrical configuration. *The Computer Journal*, v. 8, n. 3, p. 264–272, 1965.

[3] BENZI, M. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, v. 182, p. 418–477, 2002.

[4] BOLDRINI, J. L. et al. *Álgebra Linear*. 3. ed. São Paulo: Harbra, 1986.

[5] BOOST. *Boost C++ Libraries*. 2017. Accessado em 21/11/2022. Disponível em: <http://www.boost.org>.

[6] BREMERMANN, H. J. *The evolution of intelligence. The nervous system as a model of its environment*. Seattle, WA, 1958. Relatório técnico No. 1.

[7] CAMPOS, V.; PIÑANA, E.; MARTÍ, R. Adaptive memory programming for matrix bandwidth minimization. *Annals of Operations Research*, Springer Science+Business Media, v. 183, p. 7–23, 2011.

[8] CAPRARA, A.; SALAZAR-GONZÁLEZ, J.-J. Laying out sparse graphs with provably minimum bandwidth. *INFORMS Journal on Computing*, v. 17, n. 3, p. 356–373, 2005.

[9] CHAGAS, G. O.; GONZAGA DE OLIVEIRA, S. L. Metaheuristic-based heuristics for symmetric-matrix bandwidth reduction: A systematic review. *Procedia Computer Science*, v. 51, p. 211–220, 2015.

[10] CHENG, K. Minimizing the bandwidth of sparse symmetric matrices. *Computing*, Springer Wien, v. 11, p. 107–110, 1973.

[11] CHHUGANI, J. et al. Fast and efficient graph traversal algorithm for cpus: Maximizing single-node efficiency. In: *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium - IPDPS*. Shanghai, China: IEEE, 2012. p. 378–389.

[12] COLORNI, A.; DORIGO, M.; MANIEZZO, V. Distributed optimization by ant colonies. In: VARELA, F. J.; BOURGINE, P. (Ed.). *Proceedings of the First European Conference on Artificial Life*. Cambridge, MA, USA: MIT Press, 1992. p. 134–142.

[13] CRANE, H. L. J. et al. Algorithm 508: Matrix bandwidth and profile reduction [f1]. *ACM Transactions on Mathematical Software*, v. 2, n. 4, p. 375–377, 1976.

[14] CUTHILL, E.; MCKEE, J. Reducing the bandwidth of sparse symmetric matrices. In: *ACM Proceedings of the 1969 24th national conference*. New York, USA: ACM, 1969. p. 157–172.

[15] CYGAN, M.; PILIPCZUK, M. Faster exact bandwidth. In: BROERSMA, H. et al. (Ed.). *Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science. Vol. 5344*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 101–109.

[16] CYGAN, M.; PILIPCZUK, M. Exact and approximate bandwidth. *Theoretical Computer Science*, v. 411, n. 40–42, p. 3701–3713, 2010.

[17] CYGAN, M.; PILIPCZUK, M. Bandwidth and distortion revisited. *Discrete Applied Mathematics*, v. 160, n. 4–5, p. 494–504, 2012.

[18] CYGAN, M.; PILIPCZUK, M. Even faster exact bandwidth. *ACM Transactions on Algorithms*, Association for Computing Machinery (ACM), v. 8, n. 1, p. 8:1–8:14, 2012.

[19] DAVIS, T. A. *Direct methods for sparse linear systems*. Illustrated edition. Gainesville, Florida, USA: Society for Industrial and Applied Mathematic (SIAM), 2006.

[20] DAVIS, T. A.; HU, Y. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, v. 38, n. 1, p. 1–25, 2011.

[21] DEL CORSO, G. M.; MANZINI, G. Finding exact solutions to the bandwidth minimization problem. *Computing*, v. 62, n. 3, p. 189–203, 1999.

[22] DONGARRA, J.; GROSSE, E. *Netlib Repository*. 1980–2023. Acessado em 1º/1/2023. Disponível em: <netlib.org>.

[23] DORIGO, M. *Optimization, Learning and Natural Algorithms*. Tese (Doutorado) — Politecnico di Milano, Milão, Itália, 1992.

[24] DUECK, G. W.; JEFFS, J. A heuristic bandwidth reduction algorithm. *Journal of Combinatorial Mathematics and Combinatorial Computing*, v. 18, p. 97–108, 1995.

[25] EATON, J. W. et al. *GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations*. 2015. Disponível em: <http://www.gnu.org/software/octave/doc/interpreter>.

[26] ESPOSITO, A. et al. A new matrix bandwidth reduction algorithm. *Operations Research Letters*, v. 23, p. 99–107, 1998.

[27] ESPOSITO, A.; TARRICONE, L. Matrix bandwidth reduction with Tabu search: parallel implementation on Cray T3D and applications to microwave circuit design. In: VOLI, E. (Ed.). *Science and Supercomputing at CINECA: Report*. Bologna, Italy: CINECA, 1996. p. 440–450.

[28] EVERSTINE, G. C. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. *International Journal for Numerical Methods in Engineering*, v. 14, p. 837–853, 1979.

[29] FEIGE, U.; KILIAN, A. Coping with the NP-hardness of the graph bandwidth problem. In: HALLDORSOON, M. M. (Ed.). *Proceedings of the Scandanavian Workshop on Algorithm Theory (SWAT'00). Lecture Notes in Computer Science (LNCS, volume 1851)*. Bergen, Norway: Springer, 2000. p. 10–19.

[30] FELIPPA, C. A. Solution of linear equations with skyline-stored symmetric matrix. *Computers and Structures*, v. 5, p. 13–29, 1975.

[31] FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, v. 8, n. 2, p. 67–71, 1989.

[32] FRASER, A. S. Simulation of genetic systems by automatic digital computers. II: Effects of linkage on rates under selection. *Australian Journal of Biological Sciences*, v. 10, p. 492–499, 1957.

[33] FR̈ER, M.; GASPERS, S.; KASIVISWANATHAN, S. P. An exponential time 2-approximation algorithm for bandwidth. *Theoretical Computer Science*, v. 511, p. 23–31, 2013.

[34] GAREY, M. R. et al. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, v. 34, p. 477–495, 1978.

[35] GEORGE, A.; LIU, J. W. H. An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software*, v. 5, n. 3, p. 284–295, 1979.

[36] GEORGE, A.; LIU, J. W. H. *Computer solution of large sparse positive definite systems*. Englewood Cliffs: Prentice-Hall, 1981.

[37] GEORGE, J. A. *Computer implementation of the finite element method*. 228 p. Tese (Doutorado) — Computer Science Department, Stanford University, CA, USA, 1971.

[38] GIBBS, N. E. Algorithm 509: A hybrid profile reduction algorithm. *ACM Transactions on Mathematical Software*, ACM, New York, NY, USA, v. 2, n. 4, p. 378–387, dez. 1976.

[39] GIBBS, N. E. *A survey of bandwidth and profile reduction algorithms*. Williamsburg, USA, 1980. Relatório técnico.

[40] GIBBS, N. E.; POOLE, W. G.; STOCKMEYER, P. K. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, v. 13, n. 2, p. 236–250, 1976.

[41] GKOUNTOUVAS, T. et al. Improving the performance of the symmetric sparse matrix-vector multiplication in multicore. In: *27th International Symposium on Parallel & Distributed Processing (IPDPS)*. Boston, MA: IEEE Computer Society, 2013. p. 273–283.

[42] GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, v. 13, n. 5, p. 533–549, 1986.

[43] GOLUB, G. H.; Van Loan, C. F. *Matrix Computations*. 3. ed. Baltimore: The Johns Hopkins University Press, 1996.

[44] GONZAGA DE OLIVEIRA, S.; KISCHINHEVSKY, M. Sierpiński curve for total ordering of a graph-based adaptive simplicial-mesh refinement for finite volume discretizations. In: *Congresso Nacional de Matemática Aplicada e Computacional (CNMAC)*. Belém: Anais do Congresso Nacional de Matemática Aplicada e Computacional, 2008. p. 581–585.

[45] GONZAGA DE OLIVEIRA, S. L. *Introdução à geração de malhas triangulares*. São Carlos: Sociedade Brasileira de Matemática Aplicada e Computacional, 2015.

[46] GONZAGA DE OLIVEIRA, S. L. *Algoritmos e seus fundamentos*. 2. ed. Lavras: Editora UFLA, 2020.

[47] GONZAGA DE OLIVEIRA, S. L. An evaluation of heuristic methods for the bandwidth reduction of large-scale graphs. *Revista Pesquisa Operacional*, v. 43, n. e268255, p. 1–22, 2023.

[48] GONZAGA DE OLIVEIRA, S. L.; ABREU, A. A. A. M. An evaluation of pseudoperipheral vertex finders for the reverse Cuthill-Mckee method for bandwidth and profile reductions of symmetric matrices. In: *37th International Conference of the Chilean Computer Science Society (SCCC)*. Santiago, Chile: IEEE, 2018.

[49] GONZAGA DE OLIVEIRA, S. L.; ABREU, A. A. A. M. A pseudoperipheral vertex finder based on the George-Liu and Kaveh's b algorithms for small symmetric matrices. In: *Anais do L Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. Rio de Janeiro, RJ: Sociedade Brasileira de Pesquisa Operacional, 2018.

[50] GONZAGA DE OLIVEIRA, S. L. et al. An evaluation of four reordering algorithms to reduce the computational cost of the Jacobi-preconditioned conjugate gradient method using high-precision arithmetic. *International Journal of Business Intelligence and Data Mining*, v. 12, n. 2, p. 190–209, 2017.

[51] GONZAGA DE OLIVEIRA, S. L. et al. Finding a starting vertex for the reverse Cuthill-McKee method for bandwidth reduction: a comparative analysis using asymmetric matrices. In: *Proceedings of 18th International Conference on Computational Science and Its Applications, ICCSA. Lecture Notes in Computer Science book series (LNCS)*. Melbourne, Australia: Springer International Publishing, 2018. v. 10960, p. 123–137.

[52] GONZAGA DE OLIVEIRA, S. L.; BERNARDES, J. A. B.; CHAGAS, G. O. An evaluation of several heuristics for bandwidth and profile reductions to reduce the computational cost of the preconditioned conjugate gradient method. In: *Anais do Simpósio Brasileiro de Pesquisa Operacional - SBPO*. Vitória, ES: Sociedade Brasileira de Pesquisa Operacional - Sobrapo, 2016.

[53] GONZAGA DE OLIVEIRA, S. L.; BERNARDES, J. A. B.; CHAGAS, G. O. An evaluation of low-cost heuristics for matrix bandwidth and profile reductions. *Computational & Applied Mathematics*, v. 37, p. 1412–1471, 2018.

[54] GONZAGA DE OLIVEIRA, S. L.; BERNARDES, J. A. B.; CHAGAS, G. O. An evaluation of reordering algorithms to reduce the computational cost of the incomplete Cholesky-conjugate gradient method. *Computational & Applied Mathematics*, v. 37, p. 2965–3004, 2018.

[55] GONZAGA DE OLIVEIRA, S. L.; CARVALHO, C. Metaheuristic algorithms for the bandwidth reduction of large-scale matrices. *Journal of Combinatorial Optimization*, v. 43, p. 727–784, 2022.

[56] GONZAGA DE OLIVEIRA, S. L.; CHAGAS, G. O. *Introdução a Heurísticas para Redução de Largura de Banda de Matrizes*. São Carlos, SP: Sociedade Brasileira de Matemática Aplicada e Computacional (SBMAC), 2014.

[57] GONZAGA DE OLIVEIRA, S. L.; CHAGAS, G. O. A systematic review of heuristics for symmetric-matrix bandwidth reduction: methods not based on metaheuristics. In: *Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. Porto de Galinhas: Sociedade Brasileira de Pesquisa Operacional, 2015.

[58] GONZAGA DE OLIVEIRA, S. L.; CHAGAS, G. O. Um análise comparativa entre quatro algoritmos para reduções de largura de banda de matrizes. In: *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics - CNMAC*. Campinas, SP: Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC, 2018.

[59] GONZAGA DE OLIVEIRA, S. L.; CHAGAS, G. O.; BERNARDES, J. A. B. An analysis of reordering algorithms to reduce the computational cost of the jacobi-preconditioned cg solver using high-precision arithmetic. In: AL., O. G. et (Ed.). *International Conference on Computational Science and Its Applications - ICCSA*. Cham, Switzerland: Springer, 2017, (Lecture Notes in Computer Science, v. 10404). p. 3–19.

[60] GONZAGA DE OLIVEIRA, S. L. et al. An assessment of reordering algorithms to speed up the ICCG method applied to CFD problems. In: AL., O. G. et (Ed.). *International Conference on Computational Science and Its Applications - ICCSA*. Cham, Switzerland: Springer, 2018, (Lecture Notes in Computer Science, v. 10960). p. 35–52.

[61] GONZAGA DE OLIVEIRA, S. L.; KISCHINHEVSKY, M.; TAVARES, J. M. R. S. Novel graph-based adaptive triangular mesh refinement for finite-volume discretizations. *CMES: Computer Modeling in Engineering & Sciences*, v. 95, n. 2, p. 119–141, 2013.

[62] GONZAGA DE OLIVEIRA, S. L.; SILVA, L. M. An ant colony hyperheuristic approach for matrix bandwidth reduction. *Applied Soft Computing*, v. 94, p. 106434, 2020.

[63] GONZAGA DE OLIVEIRA, S. L.; SILVA, L. M. Evolving reordering algorithms using an ant colony hyperheuristic approach for accelerating the convergence of the ICCG method. *Engineering with Computers*, v. 36, p. 1857–1873, 2020.

[64] GONZAGA DE OLIVEIRA, S. L.; SILVA, L. M. Experiments with pseudoperipheral vertex finders for heuristics for bandwidth reduction evolved by an ant colony hyperheuristic approach. In: *39th International Conference of the Chilean Computer Science Society (SCCC)*. Coquimbo, Chile: IEEE, 2020. p. 1–5.

[65] GONZAGA DE OLIVEIRA, S. L.; SILVA, L. M. Low-cost heuristics for matrix bandwidth reduction combined with a Hill-Climbing strategy. *Rairo - Operations Research*, v. 55, n. 4, p. 2247–2264, 2021.

[66] GURARI, E. M.; SUDBOROUGH, I. H. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *Journal of Algorithms*, v. 5, n. 4, p. 531–546, 1984.

[67] HAMMING, R. W. Error detecting and error correcting codes. *Bell System Technical Journal*, v. 29, n. 2, p. 147–160, 1950.

[68] HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search. In: GLOVER, F.; KOCHENBERGER, G. et al. (Ed.). *Handbook of Metaheuristics*. 1st edition. ed. New York: Springer, 2003. p. 145–184.

[69] HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search methods. In: FLOUDAS, C. A.; PARDALOS, P. M. (Ed.). *Encyclopedia of Optimization*. 2nd edition. ed. New York: Springer, 2009. p. 3975–3989.

[70] HARARY, F. *Graph Theory*. Reading, MA: Addison-Wesley, 1969.

[71] HESTENES, M. R.; STIEFEL, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, v. 49, n. 36, p. 409–436, 1952.

[72] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems - An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press, 1975.

[73] HONG, S.; OGUNTEBI, T.; OLUKOTUN, K. Efficient parallel graph exploration on multicore CPU and GPU. In: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'11)*. Washington, DC: IEEE Computer Society, 2011. p. 100–113.

[74] JENNINGS, A. A compact storage scheme for the solution of symmetric linear simultaneous equations. *The Computer Journal*, v. 9, p. 281–285, 1966.

[75] KAVEH, A. *Structural Mechanics: Graph and Matrix Methods*. Baldock, England: Research Studies Press LTD, 2004.

[76] KAVEH, A.; SHARAFI, P. Nodal ordering for bandwidth reduction using ant system algorithm. *Engineering Computations*, v. 26, p. 313–323, 2009.

[77] KING, I. P. An automatic reordering scheme for simultaneous equations derived from network systems. *International Journal for Numerical Methods in Engineering*, v. 2, n. 4, p. 523–533, 1970.

[78] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.

[79] KOOHESTANI, B. On the solution of the graph bandwidth problem by means of search methods. *Applied Intelligence*, v. 53, p. 7988–8004, 2022.

[80] KOOHESTANI, B.; POLI, R. A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In: *Research and Development in Intelligent Systems XXVIII*. London, UK: Springer London, 2011. p. 93–106.

[81] KRYLOV, A. N. On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined (in Russian). *Izvestija AN SSSR (News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk, 1931, VII*, v. 491-539, n. 4, 1931.

[82] LANCZOS, C. Solutions of systems of linear equations by minimized iterations. *Journal of Research of the National Bureau of Standards*, v. 49, n. 3, p. 33–53, 1952.

[83] LEWIS, J. G. Implementations of the Gibbs-Poole-Stockmeyer algorithms and Gibbs-King algorithms. *ACM Transactions on Mathematical Software*, v. 8, p. 180–189, 1982.

[84] LIM, A.; LIN, J.; XIAO, F. Particle Swarm optimization and hill climbing for the bandwidth minimization problem. *Applied Intelligence*, v. 3, n. 26, p. 175–182, 2007.

[85] LIM, A.; RODRIGUES, B.; XIAO, F. Heuristics for matrix bandwidth reduction. *European Journal of Operational Research*, p. 69–91, 2006.

[86] LIM, A.; RODRIGUES, B.; XIAO, F. A fast algorithm for bandwidth minimization. *International Journal on Artificial Intelligence Tools*, v. 3, p. 537–544, 2007.

[87] LIU, J. W. *On reducing the profile of sparse symmetric matrices*. 208 p. Tese (Doutorado) — University of Waterloo, fev. 1976.

[88] LIU, W.; SHERMAN, A. H. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, v. 13, n. 2, p. 198–213, april 1976.

[89] LIVESLEY, R. K. The analysis of large structural systems. *The Computer Journal*, v. 3, n. 1, p. 34–39, 1960.

[90] MARTÍ, R.; CAMPOS, V.; PIÑANA, E. A branch and bound algorithm for the matrix bandwidth minimization. *European Journal of Operational Research*, v. 186, n. 2, p. 513–528, 2008.

[91] MARTÍ, R. et al. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, v. 135, n. 2, p. 450–459, 2001.

[92] METROPOLIS, N. et al. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, v. 21, p. 1087–1092, 1953.

[93] MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, 1997.

[94] MLADENOVIĆ, N. et al. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, v. 200, p. 14–27, 2010.

[95] MONIEN, B.; SUDBOROUGH, I. H. Bandwidth problems in graphs. In: *Proceedings of the 18th Annual Allerton Conference on Communication, Control, and Computing*. Monticello, Illinois: University of Illinois at Urbana-Champaign, 1980. p. 650–659.

[96] MOORE, E. F. The shortest path through a maze. In: *Proceedings of the International Symposium on the Theory of Switching, Part II*. Cambridge, Massachusetts: Harvard University Press, 1959. p. 285–292.

[97] NETWORKX. *Networkx - Network Analysis in Python*. 2022. Networkx.org. Accessado em 21/11/2022.

[98] PAPADIMITRIOU, C. The NP-completeness of bandwidth minimization problem. *Computing Journal*, v. 16, p. 177–192, 1976.

[99] PETIT, J. Addenda to the survey of layout problems. *Bulletin of the EATCS*, European Association for Theoretical Computer Science, v. 105, p. 177–201, 2011.

[100] PIÑANA, E. et al. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, v. 153, n. 1, p. 200–210, 2004.

[101] REID, J. K.; SCOTT, J. A. Reducing the total bandwidth of a sparse unsymmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, v. 28, n. 3, p. 805–821, 2006.

[102] RODRIGUEZ-TELLO, E.; KAO, H. J.; TORRES-JIMENEZ, J. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, v. 185, p. 1319–1335, 2008.

[103] ROSEN, R. Matrix bandwidth minimization. In: ACM. *Proceedings of 23rd National Conference*. Princeton, NJ: Brandon/System Press, 1968. p. 585–595.

[104] SAAD, Y. *Iterative Methods for Sparse Linear Systems*. Segunda edição. Philadelphia: Society for Industrial and Applied Mathematics, 2003.

[105] SCIPY. *SciPy - Fundamental algorithms for scientific computing in Python*. 2019. Scipy.org. Accessado em 21/11/2022.

[106] SCOTT, J.; TŮMA, M. *Algorithms for Sparse Linear Systems*. Cham, Switzerland: Birkhauser, 2023.

[107] SILVA, P. H. G. et al. A biased random-key genetic algorithm for bandwidth reduction. In: AL., O. G. et (Ed.). *Proceedings of the The 20th International Conference on Computational Science and its Applications - ICCSA. Lecture Notes in Computer Science*. Cagliari, Italy: Springer, 2020. v. 12249, p. 312–321.

[108] SKIENA, S. S. *The Algorithm Design Manual*. Second edition. London, UK: Springer, 2008.

[109] STFC. *The Science and Technology Facilities Council. HSL. A collection of Fortran codes for large scale scientific computation*. 2023. `http://www.hsl.rl.ac.uk`.

[110] TARJAN, R. E. *Graph theory and gaussian elimination*. Computer Science Department, Stanford University, novembro 1975. Relatório técnico.

[111] The MathWorks, Inc. *MATLAB*. 1994–2023. Acessado em 1º/1/2023. Disponível em: <http://www.mathworks.com/products/matlab/index.html>.

[112] TITHI, J. J. et al. Avoiding locks and atomic instructions in shared-memory parallel BFS using optimistic parallelization. In: *Proceedings of the 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum - IPDPSW*. Washington, DC: IEEE, 2013. p. 1628–1637.

[113] TORRES-JIMENEZ, J. et al. A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs. *Information Sciences*, v. 303, p. 33–49, 2015.

[114] VIENNACL. *ViennaCL*. 2016. Viennacl.sourceforge.net. Accessado em 21/11/2022.

[115] WOLFRAM. *Wolfram Mathematica*. 2023. Wolfram.com/mathematica. Accessado em 1º/1/2023.

Índice