USO DA METAHEURÍSTICA DE BUSCA EM VIZINHANÇA VARIÁVEL PARA REDUÇÃO DE LARGURA DE BANDA EM MATRIZES ESPARSAS

Rogério Gomes de Lima Tostas

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Civil.

Orientadores: Nelson Maculan Filho
Webe João Mansur

Rio de Janeiro
Novembro de 2019

USO DA METAHEURÍSTICA DE BUSCA EM VIZINHANÇA VARIÁVEL PARA REDUÇÃO DE LARGURA DE BANDA EM MATRIZES ESPARSAS

Rogério Gomes de Lima Tostas

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA CIVIL.

Examinada por:

Prof. Webe João Mansur, Ph.D.

Prof. Nelson Maculan Filho, D.Sc.

Prof. Michael Ferreira de Souza, D.Sc.

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

Prof. Marcone Jamilson Freitas Souza, D.Sc.

Prof. Laura Silvia Bahiense da Silva Leite, D.Sc.

Prof. Carlos Andrés Reyna Vera-Tudela, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2019

Tostas, Rogério Gomes de Lima

Uso da metaheurística de busca em vizinhança variável para redução de largura de banda em matrizes esparsas/Rogério Gomes de Lima Tostas. – Rio de Janeiro: UFRJ/COPPE, 2019.

XII, 99 p.: il.; 29, 7cm.

Orientadores: Nelson Maculan Filho

Webe João Mansur

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Civil, 2019.

Referências Bibliográficas: p. 93 – 99.

1. VNS. 2. Largura de banda. 3. Matrizes esparsas. I. Maculan Filho, Nelson *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Civil. III. Título.

A Deus, minha família e amigos.

Agradecimentos

Gostaria de agradecer a Deus por ter colocado pessoas tão sábias e bondosas no meu caminho, e por me ensinar superar todas as dificuldades apresentadas durante toda a minha vida.

À minha amada esposa Priscila Tostas, por ter sido sempre companheira e ter compreendido os muitos momentos de ausência necessários para a execução deste trabalho, não teria conseguindo vencer essa etapa sem sua presença ao meu lado.

Aos meus pais Almir e Elisete pelo grande apoio e dedicação ao longo de toda a minha vida.

Ao meu orientador professor Webe João Mansur por ter me aceito como orientando, ter provido os meios para que esta tese fosse realizada, pelas conversas no café, onde muito aprendi, e pelo grande crescimento pessoal que essa jornada me proporcionou.

Ao meu orientador e grande amigo professor Nelson Maculan Filho, sua dedicação e conselhos me ajudaram em vários momentos, sempre foi um exemplo de educador, pesquisador e mestre. Seus cuidados comigo foram muitos, não medindo esforços para a conclusão deste trabalho.

Ao meu grande amigo professor Michael Ferreira de Souza pela grande paciência, conselhos, bondade e compreensão. Espero um dia poder retribuir da mesma forma.

Ao professor José Roberto Linhares de Mattos que desde a época da graduação tem me incentivado a prosseguir em meus estudos. Um grande amigo e exemplo.

Aos professores do LAMEMO, Cid, Franciane, Eduardo e a todos os colegas de laboratório, em especial a Ivone pelos cuidados em todas as etapas desse processo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

USO DA METAHEURÍSTICA DE BUSCA EM VIZINHANÇA VARIÁVEL PARA REDUÇÃO DE LARGURA DE BANDA EM MATRIZES ESPARSAS

Rogério Gomes de Lima Tostas

Novembro/2019

Orientadores: Nelson Maculan Filho
Webe João Mansur

Programa: Engenharia Civil

Neste trabalho, apresentamos duas propostas para a resolução do problema de redução de banda em matrizes esparsas (PRB). Devido à sua grande aplicabilidade em engenharia, computação e otimização, o PRB é objeto de extensa pesquisa via tanto modelos exatos quanto heurísticas. Apresentamos um método exato para o PRB via modelagem matemática que apresenta como grande vantagem a garantia da otimalidade das soluções obtidas. O PRB pertence à classe de problemas NP-difíceis, sendo assim, em geral, o tempo computacional para a obtenção de soluções exatas cresce exponencialmente com o tamanho da entrada. Como alternativa ao alto custo computacional de obtenção de soluções exatas, propomos o uso de uma variante da metaheurística de Busca em Vizinhança Variável (VNS).

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

USING VARIABLE NEIGHBORHOOD SEARCH METAHEURISTIC TO REDUCE BANDWIDTH ON SPARSE MATRICES

Rogério Gomes de Lima Tostas

November/2019

Advisors: Nelson Maculan Filho
Webe João Mansur

Department: Civil Engineering

In this thesis, we present two proposals to solve the problem of bandwidth reduction on sparse matrices (PRB). Due to its wide applicability in engineering, computing and optimization, PRB is the subject of extensive research via both exact and heuristic models. We present an exact method for PRB via mathematical programming that has as great advantage the guarantee of optimality of the obtained solutions. PRB belongs to the class of NP-hard problems, so in general the computational time to get exact solutions grows exponentially with the size of the input. As an alternative to the high computational cost of obtaining exact solutions, we propose the use of a variant of the Variable Neighborhood Search (VNS) metaheuristic.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O desenvolvimento de métodos eficientes para a resolução de sistemas de equações lineares do tipo $Ax = b$, onde matriz A é esparsa e de grande porte tem sido uma parte central de diversas pesquisas ao longo dos anos, principalmente devido às suas aplicações, como por exemplo, a discretização de modelos matemáticos, o uso de redes de sistemas computacionais, o desenvolvimento de modelos econômicos e o uso de sistemas de distribuição de energia. Conforme KAVEH e SHARAFI [1], na mecânica das estruturas, cerca de 30% a 50% do custo computacional pode ser relacionado á resolução de sistemas de equações lineares do tipo $Ax = b$. Para uma grande gama de problemas não-lineares de otimização de estruturas esse percentual pode chegar até a 80% do custo computacional empregado na busca por uma solução viável. A sua maior dificuldade na busca por uma solução eficiente para esse tipo de problema é devido a uma característica inerente a todos os problemas de grande porte, que em geral, necessitam de muita memória e um alto custo de processamento para armazenar e resolver esses sistemas de equações lineares.

A maior parte do alto custo ocorre devido à discretização de modelos matemáticos que reduzem o problemas a uma série de sistemas lineares obtidos por meio de discretização e linearização de equações diferenciais parciais ($EDPs$), equações diferenciais elípticas ou parabólicas. Em engenharia, segundo BENZI [2], alguns dos métodos numéricos mais utilizados 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, onde sistemas de equações lineares são gerados na discretização desses modelos. Recentemente o uso de pré-condicionadores associado a métodos diretos e iterativos com uma alternativa eficiente na busca de soluções ótimas em tempo computacional aceitável.

Os *solvers* de métodos de elementos finitos para aproximação de soluções de equações diferenciais parciais, de sistemas de transmissão de energia de grande escala, de desenho de circuitos e de geofísica numérica, de programação matemática baseiam-se na resolução de sistemas lineares e, frequentemente, se utilizam de algo-

ritmos para a redução da largura de banda das matrizes envolvidas. Esse tratamento preliminar da matriz esparsa visa reduzir o tempo de obtenção de solução e os recursos computacionais exigidos.

Em relação a estruturas físicas, resultados de alta precisão e cada vez mais eficientes têm sido obtidos com a utilização de simulações computacionais de estruturas cada vez mais complexas. Isso só tem sido possível uma vez que as estruturas analisadas pelo computador são discretizadas por malhas complexas que quanto melhor a resolução dessas malhas, maior é a precisão numérica correspondente. Entretanto, aumentando-se esta resolução, o tamanho do sistema linear correspondente a ser resolvido numericamente também aumenta significativamente. Problemas com milhões de variáveis estão sendo construídos hoje em dia e o tamanho destas malhas irá crescer substancialmente em um futuro próximo. Desta forma, o grande obstáculo e também o maior desafio no desempenho de simulações numéricas de grande escala é a velocidade na qual sistemas de equações lineares são resolvidos.

Alguns exemplos que ilustram com precisão como essa discretização pode ser realizada são apresentados a seguir. Em CARMO [3], os autores apresentam o resultado de um estudo feito pelo *Research Institute for Advanced Computer Science* da NASA envolvendo a modelagem bidimensional do fluxo sobre a asa de um avião com aerofólio. Nesta simulação, os pontos críticos são representados por uma maior quantidade de informação, aumentando a resolução nestes locais. Algumas informações são de grande importância para a precisão do modelo como as variáveis de pressão hidrodinâmica e componentes de velocidade que resultam em um sistema de equações diferenciais parciais (não lineares). Nesse processo de resolução, cada passo de uma iteração não-linear requer a solução de um sistema de equações lineares $Ax = b$. Como a conectividade e localização geométrica dos pontos da malha são conhecidos, é possível construir o grafo associado à matriz de coeficientes.

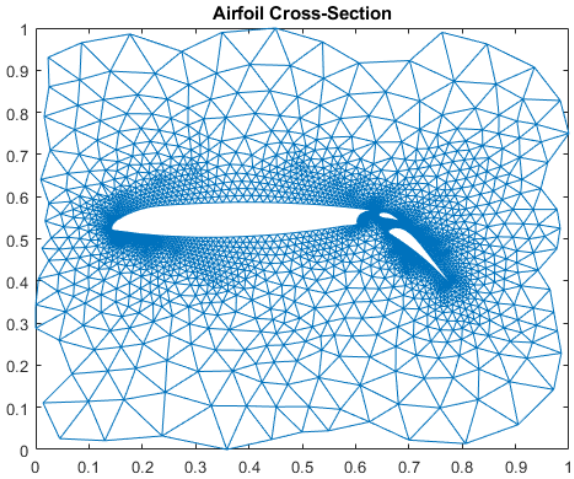


Figura 1.1: Malha de elementos finitos de um aerofólio da NASA.

Fonte: www.nasa.gov/image-feature/langley/100/naca-airfoils

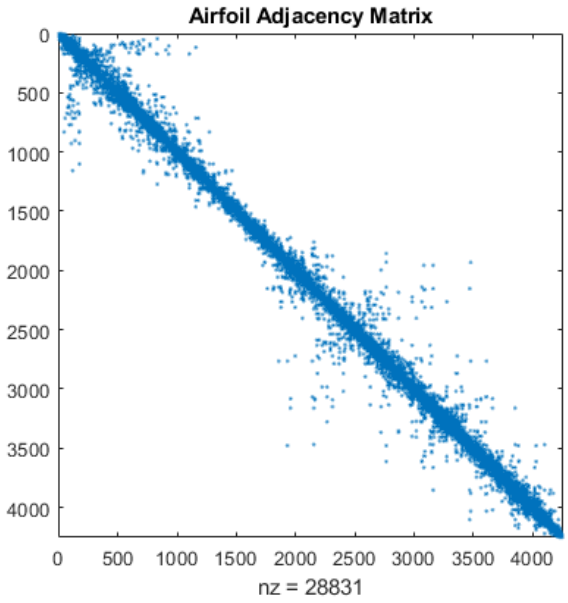


Figura 1.2: Estrutura da matriz de coeficientes.

Fonte: www.nasa.gov/image-feature/langley/100/naca-airfoils

A malha indicada na Figura 1.1 possui 4253 pontos e 28831 conexões. A estrutura da matriz de coeficientes correspondente a esta malha pode ser vista na Figura 1.2. Esta matriz é altamente esparsa, ou seja, possui a maioria dos elementos iguais a zero e a sua densidade igual a 0,0016.

Estruturas mais complexas também podem ser transformadas em matrizes de coeficientes com um alto grau de esparsidade. O modelo indicado na Figura 1.3 apresenta a malha de elementos finitos criada a partir do corte longitudinal de um navio Destroyer que possui 25026 pontos. O modelo de elementos finitos que gerou

essa matriz contém 2690 nós, 3172 elementos de ligação, 134 elementos de placas triangulares e 2793 elementos de placas quadriláteras. A largura de banda é de 11173 e a sua densidade igual a 0,0035, conforme ilustrado na Figura 1.4.

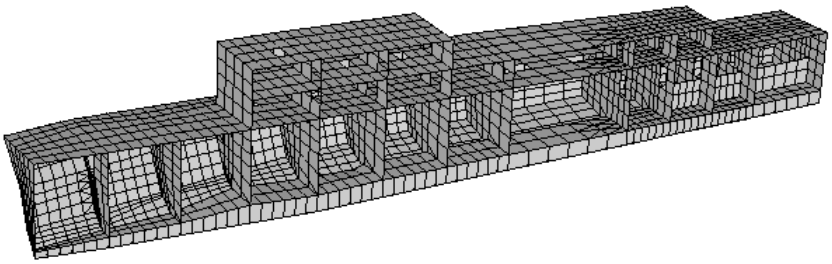


Figura 1.3: Malha de elementos finitos corte longitudinal de um navio Destroyer.

Fonte: https://sparse.tamu.edu/HB/dwt-2680

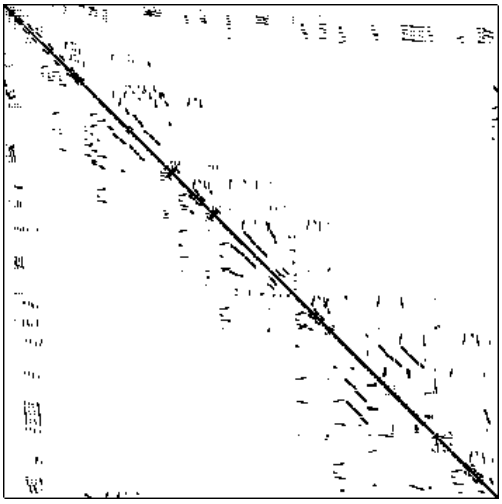


Figura 1.4: Estrutura da matriz de coeficientes.

Fonte: https://sparse.tamu.edu/HB/dwt-2680

Na maioria dos casos, as reduções de largura de banda afetam a taxa de convergência de métodos baseados no subespaço de Krylov [4], [5] como é o caso do método dos gradientes conjugados [6].

Os métodos empregados para resolver sistemas de equações lineares podem ser classificados em métodos diretos ou iterativos. A escolha de cada método está diretamente ligado a algumas características do problema que deseja resolver.

Os métodos diretos, tem como base a fatoração da matriz de coeficientes A em matrizes facilmente inversíveis, esses métodos são amplamente utilizados como solucionador em muitos códigos industriais, especialmente onde a confiabilidade é a principal preocupação. Segundo os autores GEORGE e LIU [7] e DUFF *et al.* [8], um outro motivo que leva a escolha de métodos diretos é devido sua robustez,

desta formam tendem a exigir uma quantidade previsível de recursos em termos de tempo e armazenamento. De acordo com BENZI [2], os métodos diretos têm sido tradicionalmente utilizados em análise de estruturas e modelagens de periféricos semicondutores, em grande parte da área de dinâmica de fluidos e outros exemplos não são oriundos de equações diferenciais parciais como circuitos e redes de sistemas de potência. Como exemplos de métodos diretos podemos citar as decomposições LU e de Cholesky. Os métodos diretos são baseados na eliminação gaussiana. De acordo com TARJAN [9], a eliminação gaussiana necessita de $O(n \cdot \beta^2)$ operações para a resolução de um sistema de n equações lineares e largura de banda β. Desta forma, se $\beta \cong n$, então são realizadas $O(n^3)$ operações, sendo assim, a eliminação gaussiana requer $O(n \cdot \beta)$ espaços na memória ao se utilizar armazenamento baseado em vetores. Para saber mais detalhes sobre os métodos diretos para a solução de sistemas de equações lineares recomendamos a leitura de DAVIS [10]. Em WATKINS [11] são apresentados resultados que mostram a relação direta entre eficiência da aplicação do método de Cholesky e largura de banda.

O uso de métodos iterativos se faz necessário uma vez que métodos diretos têm escalabilidade ruim em relação ao tamanho do problema, em termos de contagem de operações e exigências de memória, especialmente em problemas tridimensionais oriundos da discretização de EDPs. Modelagens multifísicas e simulações tridimensionais como aqueles que estão sendo realizadas pelo *U.S. Department of Energy's ASCI program* que em sua formulação contém sistemas lineares que compreendem centenas de milhões ou até bilhões de equações e incógnitas. Outros exemplos podem ser encontrados em alguns círculos bem específicos como a indústria petrolífera e setores ligados energia nuclear, onde o uso de métodos iterativos sempre foram populares. Na verdade, essas áreas citadas anteriormente têm historicamente servido de forte estímulo para grande parte da pesquisa e desenvolvimento desde o início da utilização de métodos iterativos como podem ser observados nos textos clássicos propostos pelos autores VARGA [12], VARGA [13] e YOUNG [14].

Mesmo sem considerar esses problemas em escala extremamente grandes, os sistemas com vários milhões de incógnitas são cada vez mais rotineiramente encontrados em muitas aplicações, fazendo com que o uso de métodos iterativos seja praticamente obrigatório. Enquanto os métodos iterativos exigem menos armazenamento e muitas vezes exigem menos operações do que os métodos diretos, especialmente quando uma solução aproximada de baixa precisão relativa é pedida, eles não têm a confiabilidade dos métodos diretos. Em algumas aplicações, métodos iterativos muitas vezes falham e o pré-condicionamento é necessário, embora nem sempre suficiente, para atingir a convergência em uma quantidade razoável de tempo [2].

De acordo com BENZI [2], pode-se concluir que a classificação tradicional de métodos de solução como sendo direta ou iterativa é uma simplificação excessiva e

não é uma descrição satisfatória do estado atual das coisas. Essa afirmação ganha cada vez mais força de acordo com os seguintes argumentos, em primeiro lugar, as fronteiras entre as duas classes de métodos tornaram-se cada vez mais tênues, com um número de ideias e técnicas da área de solucionadores de métodos diretos utilizado que são aplicados em matrizes esparsas sendo transferidos, sob a forma de pré-condicionadores, para o campo dos métodos iterativo, com o resultado que os métodos iterativos estão tornando-se cada vez confiáveis. Em segundo lugar, enquanto solucionadores diretos são quase sempre baseados em alguma versão de eliminação de Gauss, o campo de métodos iterativos compreende uma enorme variedade de técnicas, que vão desde métodos verdadeiramente iterativos, como os clássico Jacobi, Gauss-Seidel e iterações SOR, á métodos no subespaço de Krylov, que teoricamente, convergem em um número finito de passos.

Esta tese está organizada da seguinte maneira: o Capítulo 2 introduz o problema de minimização de largura de banda em matrizes esparsas, também conhecido como problema de redução de banda (PRB) e apresenta alguns conceitos básicos e definições em grafos e matrizes.

O Capítulo 3 apresenta conceitos elementares de Programação Linear Inteira (PLI), relaxação linear e propõe-se um modelo exato baseado em PLI para a resolução do problema de redução de banda.

O Capítulo 4 realiza uma revisão sistemática entre as principais heurísticas e metaheurísticas utilizadas na resolução do PRB.

O Capítulo 5 apresenta a metaheurística de Busca em Vizinhança Variável (VNS) e suas principais variações.

No Capítulo 6, propõe-se uma nova busca local ao método metaheurístico VNS para resolução do problema de redução de banda.

O Capítulo 7 apresenta os resultados obtidos nos experimentos computacionais.

O Capítulo 8 expõe as conclusões e as sugestões de trabalhos futuros.

Capítulo 2

Minimização de Largura de Banda em Matrizes Esparsas

O Problema de Minimização de Largura de Banda em Matrizes (MBMP do inglês *Matrix Bandwidth Minimization Problem*), também conhecido como Problema de Redução de Banda (PRB), é um problema computacional que tem gerado um interesse considerável ao longo dos anos devido a uma gama significativa de aplicações em problemas de otimização global [15].

Podemos definir o PRB a partir de uma matriz M quadrada e esparsa, a matriz a ser analisada pode apresentar a característica de ser simétrica ou não. Em nosso trabalho vamos nos concentrar exclusivamente em matrizes simétricas. A banda de uma matriz é a banda diagonal que compreende as diagonais que contenham elementos não nulos mais distantes da diagonal principal em ambas as direções. O objetivo é, portanto, minimizar as distâncias de tais diagonais mais afastadas da diagonal principal da matriz à esquerda e à direita.

Na tentativa de se resolver esse problema utilizam-se heurísticas para as reduções de largura de banda que realizam permutações de linhas e colunas das matrizes tentando obter uma estrutura compacta e com coeficientes não nulos próximos à diagonal principal. Este problema foi mostrado por PAPADIMITRIOU [16] ser um problema NP-difícil, o mesmo autor também mostrou que o problema de se determinar se os vértices de um grafo têm uma ordenação que produz uma largura de banda de tamanho B ou menor é NP-Completo, portanto, os esquemas discutidos nesta tese são estratégias para reduzir a largura de banda.

A perda de elementos nulos ao realizar a fatoração de Cholesky é denominada de preenchimento e deve ser evitada sempre que possível. Em [11] mostra-se que a fatoração de Cholesky preserva a banda e envelope da matriz. Por esse motivo, ao utilizar matrizes de banda e envelope pequenos o preenchimento da matriz também será pequeno.

As heurísticas para as reduções de largura de banda realizam permutações de

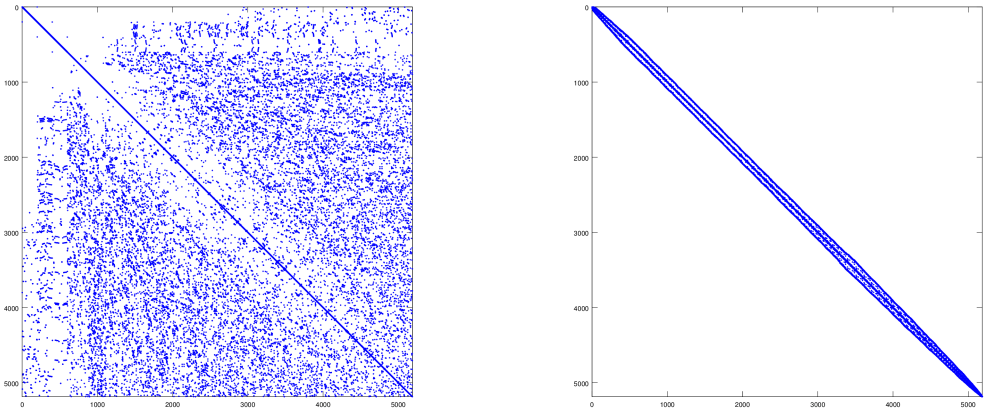


Figura 2.1: Banda original: 5128 Banda Minimizada: 119

linhas e colunas das matrizes, deixando-as com uma estrutura compacta e com coeficientes não nulos próximos à sua diagonal principal. Para uma sistema de equações lineares $Ax = b$, essas heurísticas produzem uma matriz de permutação Y tal que YAY^T têm uma largura bem menor que a matriz A original. Desta forma, o sistema de equações lineares permutado $YAY^T(Y\overline{x}) = Y\overline{b}$ continua esparso, simétrico e positivo definido. O principal motivo desta transformação é devido ao fato desse sistema permutado exigir um número menor de operações aritméticas e ou exigência de armazenamento do que o sistema original.

Ao se trocar as linhas e colunas da matriz de um sistema de equações lineares, altera-se a ordem em que as equações escritas e as incógnitas são renumeradas ou reordenadas. Em DUFF e MEURANT [17], pode ser encontrada a comparação entre 17 ordenações de incógnitas de sistemas de equações lineares oriundas de discretizações de diferenças finitas para a convergência do método dos gradientes conjugados. Foi observado que é uma condição suficiente na qual as ordenações que são *"locais"* fornecem os melhores resultados. Ordenações que são *"locais"* são aquelas em que vértices (ou incógnitas no sistema original) vizinhos na malha têm posições na ordenação que não são muito distantes entre si. Esse foi o caso das ordenações produzidas pelas heurísticas de Cuthill e McKee [18] e Cuthill-McKee reverso [19]. Esse método apresenta baixo custo computacional e gera resultados razoáveis [20]. A heurística consiste em utilizar para o reordenamento a sequência em que os vértices do grafo correspondente a uma matriz são visitados ao realizar uma busca em largura na estrutura de nível do grafo sendo que os vértices de cada nível encontram-se ordenados em ordem crescente de grau.

2.1 Definições em teoria dos grafos

Seja $G = (V, E)$ um grafo conexo, simples e não-orientado formado pelo conjunto de vértices $V = \{v_1, v_2, v_3, ..., v_n\}$ e pelo conjunto de arestas $E = \{e_1, e_2, e_3, ..., e_m\}$.

Os elementos de E são pares (i, j) com $i, j \in V$, $i \neq j$.

- Dois vértices são adjacentes quando existe uma aresta entre eles e $Adj(G, v) = \{u \in V : (v, u) \in E\}$ é o conjunto de vértices adjacentes ao vértice v;
- O grau de um vértice v é o número de vértices adjacentes a v, ou seja, $Grau(G, v) = |\{u \in V : (v, u) \in E\}|$;
- Caminho é uma sequência de vértices tal que, de cada um dos vértices parte uma aresta para o vértice seguinte, até o último vértice da sequência;
- O tamanho de um caminho é o seu número de arestas;
- Em um grafo conexo, existe um caminho entre qualquer par de vértices;
- A distância $d(v, u)$ entre dois vértices v e $u \in V$ é o tamanho do menor caminho entre eles.

Definição 2.1 *Uma rotulagem dos vértices de $G = (V, E)$ é uma função bijetora $f : V \to \{1, 2, \ldots, |V|\}$ que associa a cada vértice v do grafo um rótulo correspondente.*

Definição 2.2 *A largura de banda do vértice v com a rotulagem f é definida por $B_f(v) = \max_{u:(u,v)\in E}\{|f(u) - f(v)|\}$.*

Definição 2.3 *A largura de banda do grafo G com a rotulagem f é definida por $B_f(G) = max\{|f(u) - f(v)| \ \forall \ (u, v) \in E\}$*

Definição 2.4 *Uma aresta $(u, v) \in E$ é crítica se $|f(u) - f(v)| = B_f(G)$.*

Definição 2.5 *Para uma determinada rotulagem f definimos um conjunto de arestas críticas de $G = (V, E)$ por $E_C(f) = \{(u, v) \in E : |f(u) - f(v)| = B_f(G)\}$.*

Definição 2.6 *Um vértice $v \in V$ é um vértice crítico de $G = (V, E)$ se existe $u \in V$ tal que (u, v) é uma aresta crítica de $G = (V, E)$.*

Definição 2.7 *O conjunto de vértices críticos $V_C(f)$ de $G = (V, E)$ é todo vértice de arestas de $E_C(f)$*

Definição 2.8 *Dado grafo $G = (V, E)$ e uma rotulagem f,definimos o diâmetro do vértice $v \in V$, por $diam_f(v) = \max_{(u,v)\in E}\{|f(u) - f(v)|\}$.*

Definição 2.9 *A largura de nível $b(\mathscr{L}(u))$ é o número de vértices do nível com mais vértices, ou seja, $b(\mathscr{L}(u)) = \max_{1\leq i\ell(u)} |L_i(v)|$*

Definição 2.10 *Seja $G = (V, E)$ conexo e simples. A estrutura de nível é o particionamento $\mathscr{K}(v, ...) = \{K_0(v, ...), K_1(v, ...), ..., K_{l(v)}(v, ...)\}$, em que $v \in K_0(v, ...)$ e $K_i(v, ...) = Adjc(K_{i-1}(v, ...) - \bigcup_{j=0}^{i-1} K_j(v, ...)$ para $i = 1, 2, 3, ..., \ell(v)$ e $Adjc(.)$ retorna os vértices adjacentes aos vértices do argumento.*

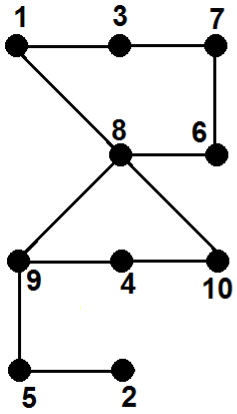
2.2 Grafos e Matrizes esparsas

A teoria dos grafos é uma poderosa ferramenta para a computação de matrizes esparsas. Em PARTER [21], quase cinquenta anos atrás, podemos ver o uso de grafos não direcionados para modelar a eliminação Gaussiana. Os grafos podem ser usados para modelar matrizes simétricas, não simétricas, fatorações etc. Além de tornar mais fácil à compreensão e análise de algoritmos para matrizes esparsas, eles também aumentam o escopo de manipulação destas matrizes usando algoritmos e técnicas já existentes.

Definição 2.11 *A Matriz de Adjacência é definida por* $A = A(G) = [a_{ij}]$, $1 \leq i \leq n$ *e* $1 \leq j \leq n$ *com*

$$a_{ij} = \begin{cases} 1, & \textit{quando } (i,j) \in E \\ 0, & \textit{caso contrário} \end{cases} \tag{2.1}$$

Observe no exemplo abaixo do grafo G e a sua matriz de adjacência $A(G)$.



$$A(G) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Figura 2.2: Grafo G e sua matriz de adjacência $A(G)$.

Definição 2.12 *O envelope de* A *é definido como* $Env(A) = \{(1 \leq i \leq n)(1 \leq j < i)(i,j) | (0 < i - j \leq B_i(A))\}$

Definição 2.13 *O profile de* A *é definido como* $|Env(A)| = \sum_{i=1}^{n} B_i(A)$.

Esta definição significa que o *profile* é a soma das distâncias, em cada linha, do primeiro coeficiente não nulo até a diagonal principal. Portanto, com a redução da largura de banda pode-se reduzir o *profile* de A, a redução de largura de banda não é condição suficiente nem necessária para a redução do profile de A. O inverso é verdadeiro. Com isso, heurísticas foram estudadas para reduções de largura de banda ou de *profile* ou ambas simultaneamente.

Definição 2.14 *Seja A uma matriz simétrica $n \times n$, com coeficientes a_{ij} e $1 \leq i, j \leq n$. A largura de banda da i-ésima linha de A é $B_i(A) = i - \min_{1 \leq j < i}(j : a_{ij} \neq 0)$.*

Definição 2.15 *A largura de banda $B(A)$ é a maior distância de coeficiente não nulo da matriz triangular inferior até a diagonal principal, considerando-se todas as n linhas da matriz, ou seja, $B(A) = \max_{1 \leq i \leq n}\{B_i(A)\} = \max_{1 \leq i \leq n, 1 \leq j < i} = \{|i-j| : a_{ij} \neq 0)\}$.*

Definição 2.16 *Uma matriz $A \in \mathbb{R}^{m \times n}$ é dita esparsa se existe um número de elementos por linha $rmax << m$ e um número de elementos por coluna $cmax << n$.*

Uma outra definição para matriz esparsa pode ser dada da seguinte da forma:

Definição 2.17 *Uma matriz $A \in \mathbb{R}^{m \times n}$ é dita esparsa se o número de elementos não nulos é expresso por $O(n^{1+\gamma})$ para $\gamma < 1$.*

Definição 2.18 *O grau de esparsidade representa a porcentagem de elementos nulos de uma matriz. Dada uma matriz $n \times n$, pode-se calcular o grau de esparsidade (GE) da seguinte forma:*

$GE = \frac{\textit{número de elementos nulos}}{\textit{número total de elementos}}.100\%$

Pode-se observar que problema de minimização de largura de banda em matrizes e em grafos são problemas equivalentes..

Vejamos os seguintes exemplos:

Dado $G = (V, E)$ um grafo conexo, e não-orientado com $|V| = 5$ e seja dada a seguinte rotulagem f : $f(v_1) = 3, f(v_2) = 1, f(v_3) = 2, f(v_4) = 5, f(v_5) = 4$ conforme a Figura 2.3.

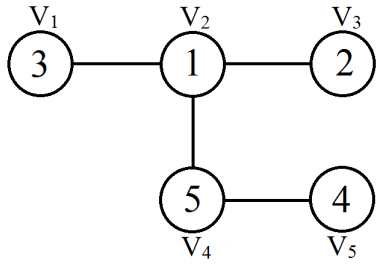


Figura 2.3: Rotulagem f do grafo G.

A largura de banda de cada vértice do grafo G dada a rotulagem f é

$$B_f(v_1) = \max\{|1-3|\} = 2$$
$$B_f(v_2) = \max\{|3-1|, |2-1|, |5-1|\} = 4$$
$$B_f(v_3) = \max\{|1-2|\} = 1$$
$$B_f(v_4) = \max\{|1-5|, |4-5|\} = 4$$
$$B_f(v_5) = \max\{|5-4|\} = 1$$

A largura de banda para o grafo G dada a rotulagem f é

$$B_f(G) = \max_{v \in V} B_f(v) = \{2, 4, 1, 4, 1\} = 4$$

A matriz de adjacência para o grafo G dada a rotulagem f é definida por

$$A(f) = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

A largura de banda da matriz de adjacência é $B(A(f)) = 4$.

Se trocamos o rótulo do vértice v_1 pelo rótulo do vértice v_2 o geramos grafo G' com a nova rotulagem f' dado pela Figura 2.4.

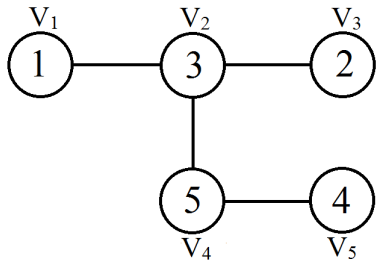


Figura 2.4: Rotulagem f' do grafo G'.

A largura de banda de cada vértice do grafo G dada a rotulagem f é

$$\begin{aligned} B_{f'}(v_1) &= \max\{|3-1|\} = 2 \\ B_{f'}(v_2) &= \max\{|1-3|, |2-3|, |5-3|\} = 2 \\ B_{f'}(v_3) &= \max\{|3-2|\} = 1 \\ B_{f'}(v_4) &= \max\{|3-5|, |4-5|\} = 2 \\ B_{f'}(v_5) &= \max\{|5-4|\} = 1 \end{aligned}$$

A largura de banda para o grafo G' dada a rotulagem f' é

$$B_{f'}(G') = \max_{v \in V} B_{f'}(v) = \{2, 2, 1, 2, 1\} = 2$$

A matriz de adjacência para o grafo G' dada a rotulagem f' é definida por

$$A(f') = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

A largura de banda da matriz de adjacência é $B(A(f')) = 2$.

Capítulo 3

Uma abordagem exata

Devido à sua grande importância, o problema de redução de banda (PRB) de matrizes esparsas tem sido objeto de extensa pesquisa [22]. Como será apresentado a seguir, as alternativas de solução para o PRB podem ser divididas em métodos exatos e métodos heurísticos.

Uma grande vantagem dos métodos exatos é a garantia de obtenção de soluções ótimas. No contexto do PRB, isto significa obter permutações de banda mínima. No trabalho proposto por GURARI e SUDBOROUGH [23], o PRB para uma matriz de ordem n tem complexidade $O(n^{\beta})$, onde β é a largura de banda pesquisada. Podemos ver outros dois algoritmos exatos propostos em DEL CORSO e MANZINI [24]. Ambos os métodos resolvem problemas de até 100 vértices, para grafos gerados aleatoriamente.

Embora métodos exatos encontrem a largura de banda ótima, a sua aplicação para problemas reais se torna inviável. Os autores CAPRARA e SALAZAR-GONZÁLEZ [25] conseguiram melhorar os métodos exatos introduzindo limites inferiores mais apertados denominados $\gamma(G)$ e $\alpha(G)$. A proposta deles permite resolver o problema de redução de banda para instâncias maiores.

Entre os principais métodos exatos, grande parte deles são métodos baseados em programação linear e inteira, tais como *branch-and bound*, *branch-and-cut*, *branch-and-price* e *branch-and-cut-and-price* [26].

Muitas destas técnicas são projetadas para serem flexíveis e independentes de domínio. O objetivo é ter a capacidade de ser aplicável a uma grande variedade de problemas práticos sem o uso intensivo de estratégias específicas para o problema tratado.

3.1 Programação linear inteira

Problemas de programação linear inteiros são aqueles em que uma ou mais variáveis de decisão são inteiras. Tais problemas são mais difíceis e precisam de algoritmos

mais complexos que o *Simplex*.

Considere o problema de Programação Linear Inteira (PLI).

$$\begin{aligned} (PLI) \quad \min \quad & z = cx & (3.1)\\ \text{s.a} \quad & Ax = b & (3.2)\\ & x \geq 0 \;\; e \;\; inteiro \end{aligned}$$

Sendo S um poliedro convexo, por exemplo $S = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ podemos escrever nosso (PLI) da seguinte forma reduzida:

$$(PLI) : z = min\{cx : x \in S \cap \mathbb{Z}^n_+\} \tag{3.3}$$

Note que os pontos extremos de S podem não ser pontos inteiros, ou seja, pontos em $\mathbb{Z}^n$. Seja $X = S \cap \mathbb{Z}^n$. Idealmente, seria adequado conhecer o menor poliedro convexo que contém X ao qual dá-se o nome de *envoltória convexa* de X e denota-se por $conv(X)$, uma vez que todos os pontos extremos deste poliedro são inteiros e, então, poderíamos utilizar o algoritmo Simplex para resolver o problema de (PLI) correspondente. Nem sempre é fácil determinar tal conjunto, logo na tentativa de tentar obter uma aproximação da solução candidata a solução viável pode-se usar um tipo de relaxação do problema inteiro original.

3.2 Relaxações e limitantes primais e duais

A habilidade de se estimar a qualidade de uma solução candidata é fundamental para otimização, sendo assim, é de extrema importância conhecer sob quais condições uma solução candidata pode ter sua qualidade estimada em relação à solução ótima, mesmo sem conhecermos a solução ótima. Este tipo de estimação é geralmente obtida através da solução ótima de problemas mais fáceis que, entretanto, possuem espaço de soluções mais abrangente que o do problema original. Estes problemas são ditos *relaxações*, podendo ser obtidos por meio da desconsideração de algumas restrições, da desconsideração das restrições de integralidade ou pela simplificação do problema.

Uma das técnicas para resolução de programação linear inteira (PLI) são os algoritmos do tipo *branch-and-bound*, onde o espaço de busca é decomposto e limitantes mais precisos são gerados iterativamente. Nestes algoritmos, precisamos de procedimentos que gerem os limites *(bounds)* que potencialmente nos levem ao valor ótimo inteiro z. Para isto, o método mais comumente utilizado é a resolução da relaxação linear do problema inteiro, obtida quando desconsideramos as restrições de integralidade de sua formulação, transformando o (PLI) em um problema linear,

digamos (PL).

Assim, como o conjunto viável do problema de programação linear inteiro é um subconjunto das soluções viáveis de um problema linear, quando tentamos resolver um problema de minimização, o valor ótimo da relaxação linear é um limite inferior *(lower bound)* ou *limitante dual* denotado por $\underline{z}$ para o valor ótimo do problema inteiro. O limitante superior é chamado de (*upper bound)* ou *limitante Primal* denotado por $\overline{z}$.

Em geral, a relaxação linear é muito mais fácil de ser resolvida que o problema inteiro; no entanto, seu valor ótimo pode estar bastante distante do valor ótimo inteiro. Essa "distância"é usualmente denominada *gap de integralidade.*

Portanto, um algoritmo para resolver um (PLI) precisa encontrar limitantes duais e primais que são melhorados iterativamente até que $\underline{z} - \overline{z} = \epsilon$ onde ϵ é um valor não-negativo pequeno suficiente para provar a otimalidade, isto é, $\underline{z} = \overline{z}$.

O valor de qualquer solução viável dá um limitante primal para z.

Definição 3.1 *Um problema* (PL) $z^R = min\{f(x) : x \in T \cap \mathbb{R}^n_+\}$ *é uma relaxação do problema* (PLI) $z = Max\ \{c(x) : x \in S \cap \mathbb{R}^n_+\}$ *se:*

i. $X \subseteq T$*;*

ii. $f(x) \geq c(x)$ *para todo* $x \in S$*.*

Proposição 3.1 *Se* (PL) *é uma relaxação de* (PLI) *então* $z^R \geq z$*.*

Proposição 3.2 *Dados um problema linear inteiro* PLI *e sua relaxação linear* PL*, temos:*

i. *Se a relaxação* PL *é infactível, então o problema original* PLI *é infactível*

ii. *Seja* x^* *uma solução ótima para* PL*. Se* $x^* \in S \bigcap \mathbb{Z}^n_+$*, então* x^* *com* $f(x^*) = c(x^*)$*, é uma solução ótima para* PLI*.*

3.3 Branch-and-bound

O algoritmo branch-and-bound foi proposto em 1960 por Land e Doig [27] para a resolução de problemas de programação inteira, já o termo *"Branch-and-bound"* como conhecemos hoje, foi empregado pela primeira vez em 1963 por Little, Murty, Sweeney e Karel et al. [28]. O algoritmo utiliza a estratégia "dividir para conquistar", no sentido em que está baseado na inspeção de tão somente partes do conjunto de soluções viáveis e, assim, obtendo estimativas para o valor da solução ótima do problema inteiro.

De acordo com o algoritmo *Branch-and-bound*, as divisões são feitas iterativamente, sempre observando que os subproblemas devem ser mais fáceis de serem resolvidos que o problema original, além de se procurar descartar subproblemas por

meio de enumeração implícita, isto equivale a dizer que,de alguma forma, podemos garantir que a solução ótima não é solução de um determinado subproblema, e assim descartá-lo.

3.3.1 Dividir-para-conquistar

Dada uma matriz $A \in \mathbb{R}^{m \times n}$ e vetores $b \in \mathbb{R}^m$ e $c^T \in \mathbb{R}^n$ com $0 < m \leq n$ considere o seguinte problema de programação inteira:

$$\begin{array}{lll} (PI) & \min & z = cx \\ & \text{s.a} & Ax = b \\ & & x \geq 0 \text{ e inteiro} \end{array} \tag{3.4}$$

Seu conjunto viável é $S_I = \{x \in \mathbb{Z}^n \mid Ax = b,\ x \geq 0\}$.

Para problemas de programação inteira, como é o caso do problema em estudo, não são conhecidas condições gerais de otimalidade como existem para a programação linear [29]. Na tentativa de obter uma solução viável para nosso problema vamos considerar a seguinte relaxação do problema linear (PI) dada por:

$$\begin{array}{lll} (P) & \min & z = cx \\ & \text{s.a} & Ax = b \\ & & x \geq 0 \end{array}$$

Seu conjunto viável é $S = \{x \in \mathbb{R} \mid Ax = b,\ x \geq 0\}$.

Como foi dito acima o problema (PI) pode ser dividido em subproblemas menores e depois recombinado de forma a obter a solução do problema original. Para isso, considere o conjunto S de soluções viáveis de (P) obtido quando realizamos uma relaxação linear de (PI) . Seja $S_1 \cup S_2 \cup ... \cup S_K$ uma decomposição de S em conjuntos menores de modo que $S = S_1 \cup S_2 \cup ... \cup S_K$ e seja $z^k = \min\{cx : x \in S_k\}$ para $k = 1, 2, ...K$. Logo temos que o valor de $z = \min_k z_k$. Observe que cada subproblema pode ser decomposto em outros subproblemas menores ainda, e assim sucessivamente, até que cada subconjunto contenha um número suficientemente pequeno de pontos para que o subproblema seja resolvido trivialmente.

Podemos representar essa decomposição através de uma *árvore de enumeração*, na qual cada nó representa um subconjunto do espaço de soluções e cada nó filho, uma componente de sua decomposição. A raiz da árvore representa o conjunto S, portanto, todo o espaço de soluções do problema. Este processo de decomposição

dos conjuntos de soluções também é conhecido como "*ramificação*" ou *branching*.

Algoritmos de branch-and-bound tiram proveito da decomposição do problema e evitam a enumeração explícita de toda a árvore, algo que seria muito custoso tanto em tempo quanto em espaço. Além disso, os algoritmos de branch-and-bound buscam a enumeração de suas soluções com auxílio de limitantes duais e primais.

Inicialmente, a *árvore de branching* de um algoritmo branch-and-bound contém apenas o nó raiz, que está associado ao problema (P). Um limitante dual é obtido resolvendo-se a relaxação linear de (P). Caso a solução ótima da relaxação linear de (P) seja inteira, a solução ótima de (PI) foi encontrada. Caso contrário, uma operação de *branching* é realizada, isso consiste em particionar o conjunto viável em dois subconjuntos através da imposição, respectivamente, das restrições $x_j \leq \lfloor \hat{x_j} \rfloor$ e $x_j \geq \lceil \hat{x_j} \rceil$, onde $\hat{x_j}$ é uma variável de valor fracionário da solução obtida através da relaxação linear (P), na iteração corrente. A árvore de branching, neste caso, é binária. Cada subproblema, por sua vez, dá origem a outros subproblemas, sendo criada uma árvore neste processo de enumeração, cujos nós correspondem aos subconjuntos criados. Para um dado nó da árvore, a regra de partição deve eliminar a solução fracionária do respectivo subproblema, e garantir que nenhuma das soluções válidas do problema original é perdida. Um nó pode sofrer uma "poda", ou *bounding*, que consiste em encontrarmos a melhor solução contida em cada subconjunto de S obtido na operação de *branching*. Podemos realizar a operação de *bounding* em uma árvore nos seguintes casos:

- Se o problema associado a ele é inviável;
- Já conhecemos sua solução ótima;
- Se o limitante dual do nó é menor que um limitante primal para (P).

Os nós que não foram podados e ainda não foram decompostos por operação de branching são chamados **nós ativos**. Os nós ativos são os únicos que podem gerar um limitante primal melhor para (P). Sendo assim, a otimalidade pode ser encontrada pelo algoritmo de *branch-and-bound* quando não há nós ativos ou quando o maior dos limitantes duais dos nós ativos for menor que o melhor limitante primal para (P). Portanto, quanto melhor forem os limitantes duais e primais, melhor será o desempenho de um algoritmo de *branch-and-bound*.

3.3.2 Critério de seleção do nó

Para determinarmos a escolha do proximo nó ativo a ser otimizado podemos escolher diversos critérios de pesquisa que variam de acordo com as características particulares de cada tipo problema. Os critérios de busca mais usados são: pesquisa em profundidade, pesquisa em largura e pesquisa do melhor limitante.

- Pesquisa em profundidade (*depth first search*) é um método em que são explorados em primeiro lugar os nós com uma maior profundidade em relação à raiz da árvore de pesquisa. Depois de cada partição é explorado o nó descendente mais à esquerda.

- Pesquisa em largura (*breadth first search*) é um método em que são explorados em primeiro lugar todos os nós de um mesmo nível de profundidade antes de passar aos nós do nível seguinte, começando pelo nó mais à esquerda.

- Pesquisa do melhor limitante (*best first*) é um método em que são explorados em primeiro lugar os nós com o valor de solução mais próximo do valor da solução na raiz da árvore de branching.

Em geral, o critério de seleção de nó com melhor limitante apresenta um resultado melhor. No entanto, a pesquisa por profundidade tem a vantagem de encontrar soluções primais mais cedo.

Além disso, estes critérios de pesquisa podem ainda ser combinados e dar lugar a regras de pesquisas compostas.

3.4 Formulação do Modelo Exato

Seja $A \in \mathbb{R}^{m \times n}$, uma matriz simétrica esparsa cujo elemento da linha i e coluna k é a_{ik}, $i, k = 1, 2, ..., n$. Construímos o seguinte grafo $\bar{G} = (\bar{V}, \bar{E})$, onde $\bar{V} = \{1, 2, 3, ..., n\}$ e as arestas de $\bar{E}$ serão $[i, k]$ $i \leq k$ quando $a_{ik} \neq 0$. Resolvendo $(PBand)$ no grafo $\bar{G}$ obteremos uma matriz de permutação $Y = y^*_{ik}$, $i, k = 1, 2, ..., n$ onde $y^*_{ik} \in \{0, 1\}$, $i, k = 1, 2, ..., n$ é uma solução ótima, isto é, a matriz $Y^T AY$ terá uma largura de banda mínima.

Dado o grafo $G(V, E)$, um grafo conexo, simples e não orientado, desejamos definir uma nova rotulagem $x_i \in \{1, 2, .., n\}, \forall i \in V$, que minimize o $w = max\{|x_i - x_j| : (i, j) \in E\}$. Apresentamos o modelo exato de programação linear inteira $(PBand)$, onde $minmax\{|x_i - x_j| : [i, j] \in E\}$ da seguinte forma:

$$(PBand) : \min w, \tag{3.5}$$

$$sujeito\ a :$$

$$x_i = \sum_{k=1}^{n} k y_{ik}, \quad \forall i \in V, \tag{3.6}$$

$$\sum_{k=1}^{n} y_{ik} = 1, \quad \forall i \in V, \tag{3.7}$$

$$\sum_{i \in V} y_{ik} = 1, \quad k = 1, 2, 3, ..., n, \tag{3.8}$$

$$w \geq x_i - x_j, \quad \forall [i, j] \in E, \tag{3.9}$$

$$w \geq x_j - x_i, \quad \forall [i, j] \in E, \tag{3.10}$$

$$y_{ik} \in \{0, 1\}, \quad \forall i \in V, \quad k = 1, 2, 3, ..., n. \tag{3.11}$$

Nesta formulação, temos as seguintes variáveis de decisão: $y_{ik} \in \{0, 1\}, \quad \forall i \in V, \quad k = 1, 2, 3, ..., n$, se $y_{ik} = 1$ significa que o nó i será rotulado com o número k, isto é, $x_i = k$, e se $y_{ik} = 0$ caso contrário. A restrição (3.6) define os rótulos nos vértices de acordo com as variáveis de atribuição, o vértice i toma o rótulo $x_i = j$. As restrições (3.7) e (3.8) garantem que cada vértice têm um único rótulo e cada rótulo é atribuído à um único vértice. As restrições (3.9) e (3.10) garantem que a largura de banda é maior ou igual à diferença absoluta entre os rótulos dos pares de vértices adjacentes. Finalmente, (3.5) define a função objetivo que minimizar a largura de banda do grafo.

A relaxação linear de $(PBand)$, pode ser obtida quando substituímos (3.11) por $y_{ik} \geq 0, \quad \forall i \in V, \quad k = 1, 2, 3, ..., n$. Esta relaxação poderá nos fornecer uma boa cota inferior do problema de programação linear inteira.

3.4.1 Limites Inferiores

O trabalho de Caprara e Salazar-González [25] permitiu um grande avanço na resolução do problema de largura de banda, sendo considerado uma referência com respeito a métodos exatos. Os autores propuseram dois limites inferiores denominados $\gamma(G)$ e a $\alpha(G)$ que podem ser computados em tempo $O(|V||E|)$. Para cada permutação parcial, a restrição de largura de banda é aplicada de forma mais forte, pois um vértice rotulado tem efeito sobre os domínios de rótulo de todos os vértices livres, não apenas sobre os de seus vizinhos. A sua abordagem funciona de forma mais eficiente em grafos esparsos do que os procedimentos proposto por DEL CORSO e MANZINI [24], sendo esse apenas capaz de resolver problemas envolvendo pequenas e médias instâncias. Os autores [25] também desenvolveram procedimentos eficientes para

o teste de extensibilidade, além de propor uma restrição para apertar os domínios de marcadores de vértices livres com base nos limites de vértices mais próximos do conjunto rotulado.

Caprara e Salazar-González [25] definiram um limite inferior $\gamma(G)$, e derivado do resultado de BLUM *et al.* [30] que apresenta outro limite inferior $\alpha(G)$.

Os limites são definidos do seguinte modo:

$$\alpha(G) = \max_{v \in V} \max_{h \in \{1,\ldots,d(v,V)\}} \left\lceil \frac{|N_h(v)| - 1}{2h} \right\rceil \tag{3.12}$$

$$\gamma(G) = \min_{v \in V} \max_{h \in \{1,\ldots,d(v,V)\}} \left\lceil \frac{|N_h(v)| - 1}{h} \right\rceil \tag{3.13}$$

Onde $N_h(v)$ é o conjunto de vértices cuja distância de v é no máximo h, temos que $d(v, V)$ é a distância máxima de v a qualquer vértice em V.

Desta forma os algoritmos começam a pesquisar a partir de um bom limite inferior definido por $b_{low} = \max\{\alpha(G), \gamma(G)\}$ em vez de um limite trivial, e indo para cima até que uma solução ótima seja encontrada ou o limite de tempo seja excedido. A utilização desses limites permite melhores resultados na utilização de métodos exatos em instâncias maiores e em tempo computacional aceitável.

Os autores DUMITRESCU e STÜTZLE [31] distinguem cinco classes para a cooperação entre os métodos exatos e busca local, que são:

- Utilizar algoritmo exato para exploração da vizinhança de grande porte em um algoritmo de busca local.
- Realizar várias execuções de uma busca local e explorar as informações em soluções de alta qualidade para definir problemas menores que são passíveis de solução com algoritmos exatos.
- Explorar limitantes em heurísticas construtivas.
- Utilizar as informações da relaxação de problemas de programação inteira para guiar a busca local.
- Utilizar algoritmos exatos para procedimentos específicos dentro de metaheurísticas híbridas.

Apresentamos um modelo exato que permite a resolução de problemas pequenos encontrando largura de banda ideal. Embora esse modelo tenha apresentado resultados promissores, sua aplicação para matrizes com maiores apresentou problemas em relação ao tempo de execução para a obtenção de solução ótima. Como alternativa para esse problema, propõe-se no capítulo 6 um algoritmo baseado em metaheurística.

Capítulo 4

Heurísticas e Metaheurísticas

As heurísticas e as metaheurísticas são estratégias de busca por uma solução ótima alternativas das técnicas clássicas de otimização. Desta forma, elas são alternativas para a solução de problemas de programação linear, programação não-linear, programação linear inteira mista e programação não-linear inteira mista.

Por exemplo, quando espera-se resolver um problema de otimização usando algumas das consideradas técnicas clássicas, o processo de otimização geralmente tem dois passos consecutivos que são definidos da seguinte forma: Inicialmente se desenvolve o modelo matemático a ser otimizado, em seguida se escolhe a técnica de otimização. Na formulação do modelo matemático geralmente a decisão mais difícil é escolher de forma adequada as variáveis de decisão do problema que permite formular a função objetivo e as restrições do modelo matemático.

Alguns fatores importantes que nos levam a resolver um problema de otimização usando as heurísticas e as metaheurísticas são, entre outros, o fato que a existência de um modelo matemático para o problema a ser resolvido pode ser de interesse secundário ou irrelevante, ou seja, uma metaheurística pode resolver um problema de otimização sem a necessidade de dispor do modelo matemático para esse problema.

Adicionalmente, as metaheurísticas podem ser muito importantes para resolver problemas que exigem uma solução em tempos de processamento pequenos (segundos ou minutos) o que acontece com as aplicações em tempo real ou em programação da operação de curto prazo. Assim como o aspecto mais importante na formulação de um modelo matemático é a escolha adequada e eficiente das variáveis de decisão, na aplicação da metaheurística a decisão que pode ser considerada mais importante é a escolha adequada e eficiente de uma forma de representar uma proposta de solução para o problema a ser otimizado.

Na utilização das metaheurísticas em geral e, particularmente da metaheurística VNS que veremos com mais detalhes mais a frente, deve- se especificar um conjunto de características relacionadas com a sua aplicação na resolução de um tipo de problema complexo.

Essas características são as seguintes:

1. A representação de uma proposta de solução que deve identificar de forma adequada um elemento do espaço de busca;
2. Uma forma adequada de identificação da qualidade da solução encontrada;
3. Identificar se uma proposta de solução é factível ou infactível;
4. Deve-se estabelecer uma estratégia de decisão em relação às soluções infactíveis;
5. Definir a estratégia para encontrar a primeira proposta de solução (ou o primeiro conjunto de soluções) que se transforma na solução corrente;
6. Estabelecer uma forma de caracterização de vizinhança;
7. Criar parâmetros de escolha do vizinho mais adequado para realizar a transição, isto é, o vizinho que deve ser escolhido como a nova solução corrente.

Em ROMERO [32], podemos encontrar com maiores detalhes todas as heurísticas e metaheurísticas citadas abaixo.

4.1 Heurísticas

As heurísticas são técnicas de otimização que geralmente encontram soluções de boa qualidade de problemas complexos [33]. Deve-se observar que entre as décadas de 1960 e 1970, as heurísticas foram as técnicas de otimização mais usadas e com maior sucesso para resolver problemas complexos do campo da otimização matemática, especialmente para aqueles problemas não lineares, discretos e não convexos. As heurísticas garantem que a solução obtida seja de boa qualidade, porém não temos garantia que essa solução encontrada seja o ótimo global do problema. Mesmo assim, o baixo esforço computacional e a facilidade na implementação fazem das heurísticas uma opção atraente quando é preciso resolver problemas com um grande espaço combinatório ou com numerosas soluções.

Um algoritmo heurístico começa em um determinado ponto do espaço de busca e realiza uma série de transições até chegar em um ótimo local do problema. O algoritmo mais popular é o algoritmo heurístico construtivo onde a cada passo é escolhida uma componente da solução e o processo termina quando é encontrada uma solução factível para o problema [33].

O uso de uma heurística pode ser muito simples sendo baseado até mesmo no uso de bom senso por se conhecer as principais características do problema em estudo ou

tendo por base a experiência de um especialista, de outra forma, o seu uso pode ser muito sofisticado, geralmente envolvendo a solução de modelos matemáticos mais complexos que devem ser relaxados em relação ao modelo original.

Segundo GLOVER e KOCHENBERGER [34], é recomendável usar técnicas heurísticas de otimização nos seguintes casos:

1. Quando não existe um método exato de otimização para resolver o problema em análise.

2. Quando a solução ótima não é muito importante do ponto de vista prático por diferentes motivos como, por exemplo, a existência de muitas soluções ótimas de qualidade muito próxima da solução ótima global.

3. Quando os dados usados apresentam incerteza elevada como acontece em muitos problemas relacionados com planejamento.

4. Quando existem limitações de tempo de processamento para encontrar a solução procurada e/ou problemas de memória para armazenamento de dados. Questões desse tipo podem aparecem com muita frequência em problemas de aplicação em tempo real.

5. Quando se pretende encontrar uma boa solução inicial para ser usada como ponto de partida na aplicação de uma técnica de otimização mais sofisticada como, por exemplo, quando se pretende usar uma metaheurísticas sofisticada ou um algoritmo *branch and bound*.

Não é uma tarefa simples classificar as técnicas heurísticas de otimização. De acordo com GLOVER e KOCHENBERGER [34], uma proposta de classificar as técnicas heurísticas é a seguinte: algoritmos heurísticos construtivos, algoritmos de decomposição, algoritmos de divisão, algoritmos de redução, algoritmos de manipulação do modelo matemático e algoritmos de busca através de vizinhança (*Steepest Descent Heuristic* - SDH).

Entre os vários tipos de heurísticas citadas anteriormente duas delas apresentam interesse especial devido ao fato de serem utilizadas nas diversas formas das metaheurísticas mais sofisticadas. Assim sendo, vamos expor rapidamente e de forma resumida o algoritmo heurístico construtivo (AHC) e a heurística de busca através de vizinhança (SDH).

4.1.1 O Algoritmo Heurístico Construtivo

O algoritmo heurístico construtivo (AHC) é uma das técnicas heurísticas de otimização mais utilizadas para resolver problemas complexos e ainda é muito usado

isoladamente ou integrado a metaheurísticas mais sofisticadas. O mais popular dos algoritmos heurísticos construtivos é o tipo guloso (*greedy*). O AHC é uma técnica de otimização que, em um processo passo a passo, gera uma solução geralmente de boa qualidade de um problema complexo. Em cada passo o AHC escolhe um elemento ou componente da solução que está sendo construída e no último passo termina de gerar uma solução factível. O elemento ou componente da solução que é escolhido em cada passo do AHC é identificado usando um indicador de sensibilidade que identifica o componente mais interessante a ser incorporada na solução em construção. Assim, a diferença fundamental entre os AHCs usados para resolver um mesmo problema está no indicador de sensibilidade usado.

Um AHC pode assumir a seguinte forma geral:

1. Armazenar os dados do problema e escolher o indicador de sensibilidade a ser usado. Escolher os componentes que podem ser incorporados na solução em construção (geralmente o processo é iniciado sem componentes).

2. Verificar se a solução em construção já representa uma solução factível. Caso seja factível então pare o processo porque foi encontrada a solução factível procurada. Em caso contrário ir ao passo 3.

3. Usando a solução em construção, resolver o problema que permite identificar o indicador de sensibilidade de todos os componentes do problema que ainda não foram incorporadas na solução em construção.

4. Usando a informação dos indicadores de sensibilidade encontrados no passo anterior identificar a componente que deve ser incorporada na solução em construção. Adicionar o componente identificado na solução em construção e voltar ao passo 2.

Os algoritmos heurísticos construtivos do tipo guloso apresentam as seguintes características:

- São processos iterativos onde em cada passo escolhe-se uma componente da solução em construção. O indicador de sensibilidade pode ser muito simples, por exemplo a menor distância no caso do caixeiro viajante e cuja informação já se encontra na matriz de distâncias, ou muito sofisticado como resolver um problema de programação linear no caso do caixeiro viajante.

- Apenas no último passo se encontra uma solução factível. Antes disso não existe nada útil disponível em relação ao problema a ser resolvido.

Esta característica nos lembra o algoritmo dual simplex em programação linear onde apenas na última iteração encontramos um ponto extremo que é adicionalmente ótimo e antes disso existe apenas uma sequência de pontos infactíveis.

4.1.2 O Algoritmo Heurístico de Busca Através de Vizinhança

O algoritmo heurístico de busca através de vizinhança (SDH) é significativamente diferente do algoritmo heurístico construtivo do tipo guloso. No AHC é gerada apenas uma solução factível através de uma sequência de passos usando um indicador de sensibilidade. No algoritmo SDH, o processo é geralmente iniciado a partir de uma solução factível e na sequência são encontradas novas soluções factíveis percorrendo o espaço de busca e passando sempre para a melhor solução vizinha.

A estratégia mais popular usada por uma técnica de otimização clássica consiste em resolver o modelo matemático do problema a partir de um ponto inicial (que pode ser factível ou infactível) o que significa em escolher valores específicos para as variáveis de decisão. A partir desse ponto inicial gera-se uma sequência de outros pontos (factíveis ou infactíveis) até atingir a convergência para um ponto factível e ótimo (local ou global). Nesse tipo de análise o conceito de região factível é fundamental.

De acordo com ROMERO [32], a estratégia fundamental da heurística SDH pode ser resumida da seguinte forma:

- O processo de otimização é iniciado através de uma solução inicial (factível ou infactível) que passa a ser chamada de solução corrente.
- Deve-se definir uma estrutura de vizinhança. Assim, deve existir uma forma de identificar as soluções que são consideradas vizinhas da solução corrente. As soluções vizinhas podem ser factíveis ou infactíveis.
- Na heurística SDH se passa da solução corrente para a melhor solução vizinha.
- O processo termina quando todas as soluções vizinhas são de pior qualidade que a solução corrente.

Deve-se observar que para implementar a heurística SDH não é obrigatoriamente necessário utilizar o modelo matemático do problema em análise. Na verdade a heurística SDH pode resolver problemas de otimização que não têm modelagem matemática e essa característica torna a heurística SDH, assim como as metaheurísticas, uma técnica de otimização relativamente distante da lógica de otimização usada na otimização clássica. Para uma adequada apresentação do funcionamento da heurística SDH devemos definir alguns conceitos abaixo.

Deve-se observar que, enquanto na otimização clássica a decisão mais importante é a escolha adequada das variáveis de decisão, na heurística SDH é fundamental a escolha de uma proposta de codificação eficiente. Uma proposta de solução está *adequadamente codificada* quando, a partir dessa informação, é possível encontrar o

valor da função objetivo (ou seu equivalente) do problema e determinar se a proposta é factível ou infactível. Uma proposta de solução está *eficientemente codificada* para sua utilização em uma heurística SDH ou em uma metaheurística quando os operadores desse algoritmo podem ser eficientemente implementados. Desta forma, a proposta de codificação deve permitir identificar de maneira única um elemento do espaço de busca, permitindo encontrar o valor da função objetivo ou equivalente e sendo possível verificar se a proposta de solução é factível ou infactível. Adicionalmente, a proposta de codificação deve permitir definir e implementar de forma adequada as estruturas de vizinhança e/ou implementar de forma eficiente os operadores existentes na metaheurística.

O **espaço de busca** está estreitamente relacionado com a proposta de codificação. Os elementos do espaço de busca são todas as propostas de solução que podem ser encontrados implementando as estruturas de vizinhança ou os operadores existentes na heurística SDH ou nas metaheurísticas. O espaço de busca pode compreender apenas uma parcela da região factível, exatamente toda a região factível ou parcelas da região factível e infactível do problema que está sendo resolvido. Os elementos do espaço de busca estão implicitamente especificados quando escolhemos a codificação de uma proposta de solução e a forma de realizar as transições a partir de uma solução inicial (ou conjunto de soluções iniciais). Assim, as estruturas de vizinhança ou operadores da metaheurística junto com a proposta de codificação definem os elementos do espaço de busca. A heurística SDH realiza a busca através de vizinhança, isto é, a partir da solução corrente, passa-se para a melhor solução vizinha. Assim, deve-se definir a vizinhança da solução corrente.

Assim sendo, a SDH assume a seguinte forma:

1. Passo preliminar: Montar os dados do problema. Escolher uma forma de codificação de uma proposta de solução denominada de p. Identificar uma forma de avaliar a qualidade da função objetivo ou equivalente e denominada $f(p)$. Definir a estrutura de vizinhança a ser usada, o que caracteriza o espaço de busca.

2. Encontrar uma solução inicial p_o que se transforma na solução corrente p_c.

3. Identificar e avaliar todas as soluções vizinhas da solução corrente p_c e identificar a melhor solução vizinha p^{best}.

4. Se $f(p^{best}) < f(p)$ então a melhor solução vizinha é melhor que a solução corrente e, portanto, fazer $p_c = p^{best}$ e voltar ao passo 3. Em caso contrário pare o processo e a solução encontrada pela heurística SDH é p_c (geralmente um ótimo local).

Em resumo, a heurística SDH gera um conjunto de soluções factíveis a partir de uma solução inicial, realizando transições através de soluções factíveis, percorrendo o espaço de busca e sempre passando para uma solução de melhor qualidade. O processo de busca termina quando todas as propostas de solução vizinhas são de pior qualidade. Essa estratégia é muito parecida com a estratégia fundamental do algoritmo primal simplex em programação linear. Uma heurística SDH geralmente é muito mais complexa, sofisticada e muito mais demorada que um AHC. Portanto, espera-se que a heurística SDH encontre soluções de melhor qualidade que o AHC na tentativa de resolver um problema complexo.

4.2 Metaheurísticas

A metaheurística é considerada como a evolução dos algoritmos heurísticos. As metaheurísticas representam um conjunto de técnicas de otimização que estão adaptadas para lidar com problemas complexos que geralmente apresentam a característica de explosão combinatória. Considerando um problema com um conjunto de soluções absurdamente grande, a ideia fundamental da metaheurística é visitar apenas um conjunto reduzido dentro do espaço de busca. Uma metaheurística é uma estratégia que especifica a forma em que deve ser realizada uma busca inteligente de uma solução quase ótima global. Ao contrário dos métodos exatos, que garantem a solução ótima, por meio das metaheurísticas é possível resolver problemas complexos e encontrar soluções de qualidade razoável em tempo aceitável para as necessidades da aplicação.

Assim, de acordo com [33], a diferença entre as metaheurísticas é a estratégia usada por cada uma delas.

Com o fim de executar o processo de busca, toda metaheurística deve especificar:

1. Como vai ser representada uma solução do problema ou um elemento do espaço de busca;
2. A forma de avaliar a função objetivo;
3. A vizinhança para uma solução ou elemento;
4. Como vai ser feita a transição de uma proposta de solução para outra;
5. Se no processo de busca, as soluções infactíveis podem ser inclusas ou não, com o objetivo de atingir a solução factível mais próxima possível da solução global.

Alternativamente, pode-se definir uma metaheurística como sendo um processo de otimização representado por uma generalização e/ou integração do algoritmo

heurístico construtivo do tipo guloso e a heurística de busca através de vizinhança de forma que, seja possível, encontrar soluções de qualidade percorrendo de forma eficiente o espaço de busca, em outras palavras, uma metaheurística pode ser interpretada como uma generalização e/ou integração do AHC e da heurística SDH mostrados anteriormente.

A estratégia que cada metaheurística utiliza na busca da solução é o que marca a diferença de uma para outra. Assim, qualquer pesquisador pode desenvolver uma metaheurística desde que formule uma estratégia eficiente e consistente.

Abaixo listamos algumas propriedades fundamentais para caracterizá-las:

- Metaheurísticas são estratégias para guiar o processo de pesquisa;
- Elas exploram o espaço de soluções de maneira eficiente a fim de encontrar soluções próximas do ótimo, ou algumas vezes o ótimo;
- As metaheurísticas são constituídas por ampla variedade de técnicas;
- Seus conceitos básicos permitem descrevê-las de maneira abstrata;
- Elas não são direcionadas a problemas específicos;
- Os algoritmos metaheurísticos são aproximativos e, tipicamente, não determinísticos;
- Podem incorporar mecanismos para evitar o confinamento em áreas do espaço de soluções e alguma forma de memória para orientar o processo de pesquisa.

A diferença fundamental entre as metaheurísticas está nesta última característica.

De todas as características mencionadas anteriormente, a mais importante é a escolha de uma forma de representação de um elemento do espaço de soluções. Essa proposta condiciona as outras características do problema e a qualidade da estratégia geral de otimização. A forma de representação de uma proposta de solução deve permitir encontrar o valor da função objetivo, ou seu equivalente, desta proposta de solução, assim como verificar se a proposta e factível ou infactível. Também a forma de representação de uma proposta de solução permite especificar a estrutura de vizinhança para realizar as transições através do espaço de soluções do problema.

Resumidamente, as metaheurísticas são técnicas de alto nível que conduzem outros métodos na exploração de problemas complexos. De acordo com BLUM e ROLI [35], o uso de metaheurísticas permite escapar de extremos (mínimo ou máximo) locais e seguir a exploração no espaço de soluções à procura de soluções ainda melhores devido a dois critérios fundamentais, porém contraditórios: **diversificação** e **intensificação**. A diversificação se refere à exploração de todo o espaço de soluções a

fim de evitar o confinamento em certas áreas, enquanto que intensificação, se refere à busca focada ou em profundidade em áreas promissoras do espaço de soluções, isto é, aquelas que apresentam boas soluções. Se por um lado, a diversificação é importante para identificar rapidamente as áreas com soluções de alta qualidade, por outro, a intensificação evita perda de tempo em áreas que já foram exploradas ou que não fornecem soluções de qualidade. Em resumo podemos dizer que a noção de diversificação se refere a estratégias de curto prazo, frequentemente, vinculadas à aleatoriedade, enquanto que intensificação são estratégias de médio e longo prazos com base no uso de algum mecanismo de memória.

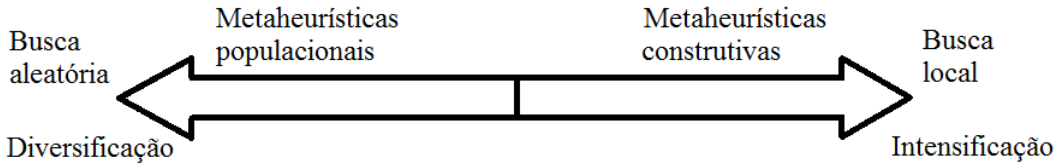


Figura 4.1: Critérios de diversificação e intensificação.

Fonte: Adaptado de BLUM e ROLI [35].

4.2.1 Simulated Annealing

Simulated Annealing (SA) é uma das primeiras metaheurísticas utilizada com muito sucesso na otimização de problemas complexos na pesquisa operacional. SA foi desenvolvido após verificar que existiam muitas semelhanças entre a técnica de construção de cristais perfeitos usando a técnica de *annealing* e a otimização de um problema complexo no campo da pesquisa operacional. O método empregado no SA constitui uma analogia à termodinâmica, especialmente, no modo como são realizados o congelamento e a cristalização de líquidos ou o resfriamento e o recozimento de metais. Isto é, uma vez expostas a altas temperaturas, as moléculas movem-se livremente umas em direção as outras. Se o resfriamento é lento, essa mobilidade termal é perdida, os átomos são alinhados e formam um cristal homogêneo, que representa o estado de energia mínima do sistema. Caso contrário, se o resfriamento ocorre rapidamente, o estado de cristalização não é atingido, mas sim um estado denominado metaestável, no qual a energia é ligeiramente mais elevada.

O algoritmo de SA pode ser considerado como um processo de transição em um espaço de busca de um problema complexo. As características fundamentais que apresenta o algoritmo são:

- A forma de escolha do vizinho mais interessante.
- O controle dos processos de transição.

A escolha do vizinho está baseada no processo de *annealing*. Assim, escolhe-se aleatoriamente um vizinho da topologia corrente, caso contrário, esse vizinho pode ser escolhido de forma probabilística baseado no processo de *annealing*.

A escolha de um vizinho de pior qualidade de forma probabilística está controlada por dois parâmetros:

- A temperatura.
- A variação da função objetivo.

Se a variação da função objetivo é pequena e/ou a temperatura for elevada a probabilidade de escolher um vizinho de pior qualidade é elevada. Nas fases iniciais são realizadas muitas transições para vizinhos de pior qualidade e essa probabilidade vai diminuindo e chegando praticamente a ser nula nas fases finais. Esta lógica permite ao *Simulated Annealing* sair dos ótimos locais. Desta forma a metaheurística SA parte de uma configuração inicial e depois gera um conjunto de configurações candidatas controladas pelo programa de esfriamento que é a estratégia geral do processo, até atingir a convergência.

A escolha adequada de alguns parâmetros dirige a eficiência de um algoritmo *Simulated Annealing*, entre eles pode-se dizer que está caracterizado pela escolha ou determinação do valor inicial de temperatura, número de tentativas de transição a cada nível de temperatura, taxa de diminuição de temperatura e critério de parada. Os altos valores da temperatura nos estágios iniciais aumentam a probabilidade de aceitação de uma solução de baixa qualidade. Além da temperatura inicial, é importante definir o número de iterações realizadas em uma mesma temperatura, bem como o seu arrefecimento ou taxa de decaimento.

Comparando a heurística SDH com ao algorítmo SA, verificam-se as diferenças fundamentais:

- No algoritmo SA, a solução vizinha é escolhida aleatoriamente e realiza a transição sem conhecer e sem avaliar as outras soluções vizinhas, enquanto heurística SDH avalia todas as soluções vizinhas para identificar a de melhor qualidade. Por isso, nas fases iniciais, SA realiza transições com maior intensidade que a heurística SDH, mas, nas fases finais ambas técnicas de otimização apresentam comportamento similares;
- O algoritmo SA pode sair do ótimo local aceitando a transição (de forma probabilística) para soluções vizinhas de pior qualidade, sendo esta a principal diferença em relação à heurística SDH;
- O algoritmo SA apresenta maior tempo de processamento e deve encontrar soluções de melhor qualidade em relação à heurística SDH.

4.2.2 Tabu Search - Busca Tabu

A teoria desta metaheurística foi desenvolvida por Fred Glover [36], a qual se baseia em conceitos que pertencem ao domínio da inteligência artificial e segundo Glover pode-se guiar um procedimento de busca local para explorar o espaço de soluções além do ótimo local. Como na busca local, a busca tabu seleciona de modo agressivo o melhor dos movimentos possíveis em cada passo, independentemente de esse movimento representar uma melhora em relação à solução corrente. Deste modo, evita-se ficar preso nos ótimos locais e pode continuar a busca em outras regiões, ou seja, o método consiste em mostrar que é possível inventar uma estratégia de busca inteligente, percorrendo o espaço de soluções de forma eficiente e seletiva. Nesse processo é fundamental integrar o processo de busca a estratégias de intensificação e de diversificação. Nesse contexto, intensificar significa realizar uma busca mais intensa em torno da solução corrente, por exemplo, aumentando o tamanho da vizinhança ou melhorando a qualidade da vizinhança. Por outro lado, diversificar significa sair de uma região do espaço de soluções e, de forma proposital, atingir uma região distante para novamente realizar algum processo de intensificação.

A implementação do TS segue os seguintes passos:

1. Montar os dados do problema; escolher a proposta de solução p, identificar como avaliar a função objetivo $f(p)$, definir a estrutura de vizinhança, identificar os atributos proibidos e o critério de aspiração, escolher os parâmetros do algoritmo (dimensão da lista tabu), escolher o critério de parada;

2. Encontrar a solução inicial p_0, que se transforma em solução corrente p_c;

3. Identificar e avaliar todas as soluções vizinhas da solução corrente, fazer o ordenamento das soluções vizinhas com o critério de qualidade, iniciando pela solução vizinha de melhor qualidade;

4. Realizar a transição para a solução vizinha melhor classificada e que não tenha atributo proibido ou, se tem o atributo proibido, deve satisfazer o critério de aspiração;

5. Atualizar a incumbente e a lista de atributo proibido e se o critério de parada for satisfeito, deve parar, caso contrário, deve retornar ao passo 3.

O algoritmo mais elementar é chamado algoritmo TS com memória de curto prazo. Este algoritmo usa uma lista de atributos proibidos e um critério de aspiração. A proibição através de atributos tenta evitar voltar a uma solução já visitada. Porém, muitas soluções vizinhas que compartilham atributos proibidos com soluções já visitadas podem ser de excelente qualidade e, eventualmente, a própria solução

ótima global pode estar proibida. Este problema é tratado usando uma função chamada critério de aspiração, assim, pode-se eliminar a proibição de uma configuração candidata. A ideia central do critério de aspiração é eliminar a proibição do atributo de uma solução vizinha de excelente qualidade porque compartilha o mesmo atributo de uma solução já visitada. Assim, se uma solução vizinha satisfaz um critério de qualidade que permite suspeitar que seja uma solução ainda não visitada então eliminamos a proibição e realizamos a transição para essa solução vizinha.

Em resumo, a diferença fundamental entre um TS básico e uma heurística SHD está na escolha do vizinho para se realizar a transição. O SHD termina o processo quando a melhor proposta de solução vizinha é pior que a solução corrente. Por outro lado, o TS realiza a transição para a melhor solução vizinha que não foi visitada anteriormente e portanto, deve-se especificar um critério de parada de forma explícita [32].

4.2.3 O Algoritmo Genético

O algoritmo genético (AG) é uma metaheurística inventada por Holland na década de 70 sendo que apenas na década de 80 teve sua aplicação de forma intensa para resolver problemas complexos no campo da pesquisa operacional. John Holland e colaboradores da University of Michigan estavam interessados em sistemas complexos artificiais, capazes de adaptarem-se a mudanças de condições ambientais [34]. No algoritmo genético, Holland encontrou semelhanças entre a forma de resolver um problema de otimização matemática e o processo de seleção natural e evolução das espécies. Sob este ponto de vista, a necessidade da estruturação de sistemas com mecanismos de autoadaptação é enfatizada. Uma população de indivíduos, para adaptar-se coletivamente em um ambiente, deve comportar-se como um sistema natural, onde a sobrevivência é promovida eliminando-se os comportamentos prejudiciais e recompensando-se os comportamentos úteis.

Um típico AG utiliza um conjunto de operadores para (i) selecionar indivíduos de uma população, com base em sua aptidão, (ii) recombinar e (iii) modificar esses indivíduos selecionados, para produzir novas gerações com indivíduos mais adaptados. Resumidamente, a ideia dos operadores é transformar uma população através de gerações sucessivas, mantendo as características de adaptação das gerações ascendentes.

O algoritmo deve cumprir as seguintes etapas:

1. Representar adequadamente uma configuração do problema. A mais comum é a representação em codificação binária, em que os operadores genéticos de recombinação e mutação são facilmente simulados;

2. Encontrar uma forma correta de avaliar a função objetivo ou seu equivalente (*fitness*) e identificar as configurações de melhor qualidade;

3. Desenvolver uma estratégia de seleção das configurações que atribua as configurações de melhor qualidade uma maior participação na formação das configurações da nova população;

4. Criar mecanismo que permita implementar o operador genético de recombinação;

5. Implementar o operador genético de mutação e terminar de gerar a nova população;

6. Quando o critério de parada for satisfeito, parar. Caso contrário, voltar ao passo 2.

4.2.4 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) é um processo iterativo que resulta da junção do método AHC do tipo guloso com o SDH. Se fundamenta nos tópicos conceituais já desenvolvidos para esses algoritmos de duas fases: construção e refinamento. Na fase de construção, uma solução viável é construída por um procedimento "guloso" aleatório. Na fase de refinamento, a solução construída é refinada por um método de busca local. Essas duas fases são aplicadas até que um critério de parada seja atingido, por exemplo, até que um número máximo de iterações sem melhora seja alcançado.

O algoritmo GRASP supera as limitações de um algoritmo construtivo clássico e gera um número elevado de propostas de soluções diferentes. Na fase de construção, caracteriza-se por permitir a escolha dos próximos componentes da solução dentro de uma *Lista Restrita de Candidatas* denominada RCL. Um algoritmo heurístico construtivo escolheria o primeiro elemento dentro desta lista RCL. O número de elementos da lista RCL deve ser variável e levar em conta a qualidade dos componentes candidatos a adição.

A fase construtiva do algoritmo GRASP apresenta os seguintes passos:

1. Escolher a solução inicial que pode ser vazia, isto é, sem adição de componentes, que se transforma na solução em construção;

2. Para a solução em construção (com alguns elementos já adicionados) e usando um indicador de sensibilidade, elaborar uma lista RCL com os k componentes mais atrativos;

3. Escolher um elemento (componente) dos k elementos existentes na lista RCL e atualizar a solução em construção corrente com a adição da componente escolhida;

4. Se a solução em construção corrente representa uma solução factível ou foi satisfeito o critério de parada (sem encontrar uma solução factível) terminar com a fase construtiva. Caso contrário voltar ao passo 1.

Depois de sair da fase construtiva, entra-se na fase final do algoritmo. A fase de pós-processamento ou busca local. Inicialmente esta fase era feita por uma heurística de busca através de vizinhança. O propósito era achar uma solução ótima local na vizinhança da solução encontrada nas fases anteriores. Através do tempo o GRASP foi sofisticando a proposta de otimização na fase de busca local, após a fase construtiva. Assim, nas propostas iniciais a ideia era usar uma heurística de busca local como a heurística SDH ou como o processo de intensificação no algoritmo *tabu search*. Em geral, a fase de busca local pode ser um processo muito simples ou muito complexo. Assim, no caso mais simples pode ser implementado uma heurística tipo SDH. No caso mais sofisticado pode ser outra metaheurística tais como *simulated annealing*, o algoritmo genético, *tabu search*, VNS, etc. Propostas intermediárias podem incorporar estratégias de otimização que fazem parte de metaheurísticas sofisticadas como, por exemplo, a estratégia de *path relinking* que foi inventada como sendo parte do *tabu search*.

Mais informações e outras aplicações sobre a metaheurística GRASP podem ser encontradas em RESENDE e RIBEIRO [37], bem com a sua integração com outras metaheurísticas.

Capítulo 5

Busca em Vizinhança Variável

A Busca em vizinhança variável (*Variable Neighbourhood Search* VNS) é uma metaheurística proposta por Nenad Mladenovic e Pierre Hansen [38] que tem por base a ideia de uma mudança sistemática de vizinhanças para explorar o espaço de soluções, alternando entre a aplicação de um método de busca local e um método de perturbação para sair do vale correspondente. Na primeira versão de VNS, denominada Descida em Vizinhança Variável (VND), a mudança de vizinhança é efetuada a cada iteração do método de pesquisa local subordinado. Esta mudança segue uma determinada sequência definida a priori. A metaheurística termina quando não é possível encontrar uma solução vizinha com custo inferior para todas as vizinhanças. Esta metaheurística foi originalmente projetada para solução aproximada de problemas de otimização combinatória, sendo estendida para problemas inteiros mistos, problemas não-lineares e problemas não lineares inteiros mistos. Além disso, é considerada uma importante ferramenta para teoria dos grafos. A maioria das metaheurísticas aceita a degradação da solução corrente (ou do conjunto de soluções correntes) como uma estratégia para sair de uma solução ótima local, nesse aspecto fundamental, VNS é significativamente diferente de outras metaheurísticas. O algoritmo VNS não aceita essa possibilidade, realizando uma mudança de vizinhança como uma forma de sair de soluções ótimas locais.

5.1 Fundamentos da Busca em Vizinhança Variável

As metaheurísticas oferecem um modo de melhorar consideravelmente o desempenho de procedimentos heurísticos simples. Quando são baseadas em busca local, elas encontram como principal obstáculo na busca por soluções de problemas complexos o fato de possuírem muitos ótimos locais, esta característica faz com que acabem convergindo para soluções de baixa qualidade.

A metaheurística VNS, ao contrário de muitas metaheurísticas baseadas em busca local, não permite degradação na função objetivo para realizar um movimento.

Ela explora, a partir de uma solução inicial, uma sequência de vizinhanças e aplica um movimento somente se a solução gerada for melhor que a solução incumbente.

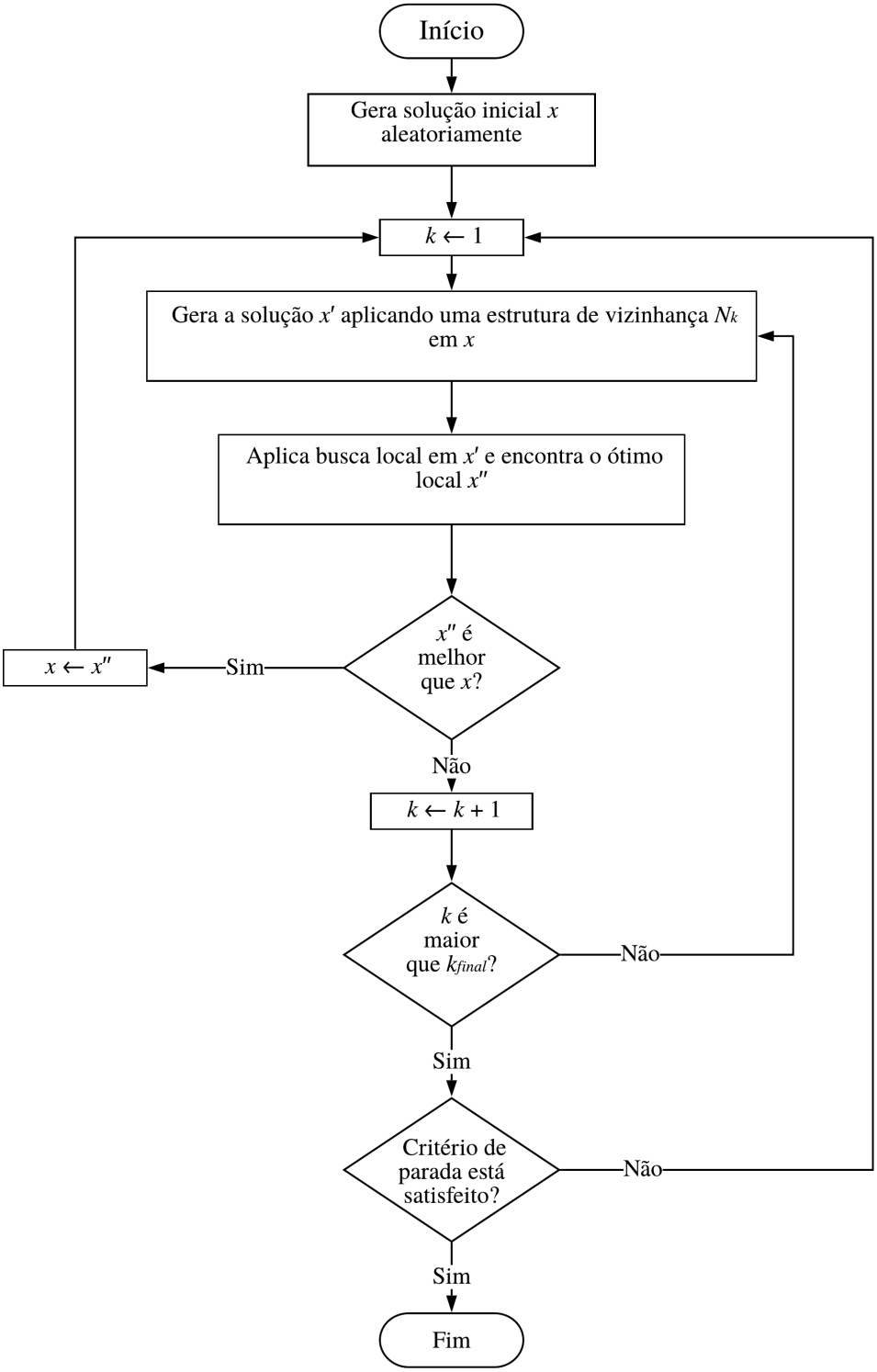


Figura 5.1: Fluxograma do algoritmo Busca em Vizinhança Variável Básica.

Fonte: Adaptado de Mladenovic e Hansen (1997)

De acordo com os autores MLADENOVIĆ e HANSEN [38], a estratégia do algoritmo VNS está inspirada em três fatos importantes:

Fato 1. Um ótimo local para uma determinada vizinhança não é necessariamente um ótimo local para uma vizinhança diferente; ou seja, VNS está constantemente modificando as bacias de atração como uma forma de sair de soluções ótimas locais;

Fato 2. Um ótimo global é também um ótimo local para qualquer uma das estruturas de vizinhança;

Fato 3. Para muitos problemas um ótimo local com respeito a uma estrutura de vizinhança compartilha características comuns com um ótimo local que corresponde a outra estrutura de vizinhança. Em geral, os mínimos locais para várias estruturas de vizinhança também compartilham características comuns, sendo possível passar de um ótimo local de uma estrutura de vizinhança para o ótimo local de outra estrutura de vizinhança.

De acordo com HANSEN *et al.* [39], os **Fatos 1** e **2** sugerem o uso de várias estruturas de vizinhança nas buscas locais para abordar um problema de otimização, ou seja, a ideia é definir um conjunto de estruturas de vizinhanças que possam ser utilizadas de forma determinística, estocástica ou ambas. Essas formas de utilizar as estruturas de vizinhanças produzem algoritmos VNS de desempenhos diferentes. Em geral, a grande maiorias das heurísticas baseadas em VNS utilizam apenas duas ou três vizinhanças somente, uma vez que, um grande número de diferentes vizinhanças equivaleria a reinícios aleatórios, levando a uma demora na convergência para uma solução ótima. O **Fato 3**, que é de caráter empírico, implica que em muitos problemas discretos verifica-se que ótimos locais para uma ou várias noções de vizinhança estão relativamente "próximos"entre si. Desta forma sabemos que uma solução ótima local fornece informações importantes em relação ao ótimo global, especialmente, se a solução ótima local for de excelente qualidade. Assim, se for encontrado um ótimo local na região em que se encontra o ótimo global, então uma metaheurística tipo VNS que realize uma busca fundamentada no processo de intensificação (exploração mais intensa de uma pequena parcela do espaço de busca, ou seja, nas proximidades da solução corrente) tem grande chance de encontrar esse ótimo global. Por outro lado, se o ótimo global se encontra em outra região, então a única possibilidade de encontrar o ótimo global consiste em implementar um processo de diversificação (exploração de novas regiões, diferente da atual). Por esse motivo um equilíbrio entre intensificação e diversificação no processo de busca pode ser de grande importância nessa metaheurística. Uma grande vantagem dessa abordagem, é que, ao contrário de muitas outras metaheurísticas, os esquemas básicos de VNS e

suas extensões são simples e exigem pouco, e às vezes nenhum parâmetro, podendo assim ser utilizado em um número maior de problemas e fornecendo soluções com qualidade aceitável.

5.1.1 Ingredientes da Busca em Vizinhança Variável

De acordo com HANSEN *et al.* [40], os ingredientes de uma metaheurística de busca de vizinhança variável incluem uma fase de melhoria usada para possivelmente melhorar uma determinada solução e a chamada fase de agitação (ou perturbação) usada para resolver esperançosamente as armadilhas dos mínimos locais. A fase de melhoria e o procedimento de agitação, juntamente com a etapa de mudança da vizinhança, são executados alternadamente até cumprir um critério de parada predefinido. Em outras palavras, as três etapas a seguir são repetidas até que algum critério de parada seja cumprido:

(1) Procedimento de agitação.

(2) Procedimento de melhoria da solução.

(3) Mudança de vizinhança.

O objetivo de um procedimento de agitação na heurística do VNS é, esperançosamente, tentar escapar das armadilhas dos mínimos locais. Nesse processo, a solução corrente também é a incumbente o que não acontece nas outras metaheurísticas. Assim, pode-se afirmar que o algoritmo VNS realiza um conjunto de transições no espaço de busca de um problema e em cada passo a transição é realizada para a nova incumbente. Se o processo encontra um ótimo local, então o algoritmo VNS muda de vizinhança para sair desse ótimo local e passar para a nova incumbente. Seja $N = \{N_1, ..., N_{k_{max}}\}$ seja um conjunto de operadores, de modo que cada operador N_k, $1 \leq k \leq N_{k_{max}}$ mapeie uma determinada solução x para uma estrutura de vizinhança predefinida $N_k(x)$.

Em particular, a escolha de vizinhanças com crescente cardinalidade gera uma progressiva diversificação. De fato, à medida que a perturbação aumenta, outras regiões do espaço de soluções são exploradas.

Observe que a ordem dos operadores no conjunto N também definirá a ordem de examinar estruturas de vizinhança de uma dada solução x. O procedimento simples de agitação consiste na seleção de uma solução aleatória da k-ésima estrutura da vizinhança, $N_k(x)$. Para alguns casos problemáticos, um salto completamente aleatório na k-ésima vizinhança é muito diversificado. Por isso, às vezes é preferível fazer a chamada agitação intensificada, que leva em consideração a sensibilidade da função objetivo a pequenas mudanças (agitação) da solução. No entanto, por

uma questão de simplicidade, assumiremos que cada variante do VNS apresentada a seguir usa um procedimento simples de agitação com base na seleção de uma solução aleatória da k-ésima estrutura da vizinhança.

O procedimento de melhoria da solução utiliza uma heurística de busca local que é baseada na exploração de uma estrutura de vizinhança $N(x)$ de uma solução atual x existente a cada iteração. Começando com uma solução inicial x, a cada iteração, ele seleciona uma solução melhor que x (se houver) da estrutura de vizinhança predefinida $N(x)$ e a define como a nova solução atual x. Uma heurística de busca local termina seu trabalho, alcançando uma solução incumbente x, de modo que ele já é o ótimo local em relação à sua estrutura de vizinhança $N(x)$.

5.2 Variantes de Algoritmos VNS

Existem várias propostas de algoritmos VNS que podem ser usadas de forma independentes ou integradas em estruturas VNS mais complexas. Podemos ver em detalhes várias dessas propostas em HANSEN e MLADENOVIĆ [41], como base nesse artigo, vamos detalhar abaixo alguns tipos de algoritmos VNS: (1) VND, (2) RVNS, (3)BVNS e (4) GVNS entre outros.

5.2.1 Algoritmo VND

Existem várias propostas de algoritmos VNS que podem ser usadas de forma independentes ou integradas em estruturas VNS mais complexas. A forma mais simples de um algoritmo tipo VNS é o algoritmo *Variável Neighborhood Descent* (VND). O algoritmo VND está inspirado no **Fato 1**, mencionado anteriormente, isto é, um mínimo local para um tipo de vizinhança não necessariamente é o mínimo local para outro tipo de vizinhança. Assim, o ótimo local x' de x na vizinhança $N_1(x)$ não necessariamente é igual ao ótimo local x" de x para a vizinhança $N_2(x)$. O algoritmo VND assume a seguinte estrutura:

De acordo com [41], algoritmo VND representa a forma mais simples de formular um algoritmo VNS que prioriza a lógica de intensificação. Para determinadas aplicações pode ser muito eficiente. Adicionalmente, esse tipo de algoritmo básico pode ser integrado em uma estrutura mais complexa de algoritmo VNS. Deve-se observar que o algoritmo VND representa uma generalização da heurística de busca através de vizinhança SDH (*Steepest Descent Heuristic*).

Em relação ao algoritmo VND, deve-se fazer as seguintes observações:

- O tamanho da vizinhança é um assunto crítico. Se o tamanho das vizinhanças for muito grande, então o algoritmo VND pode se tornar muito lento. Desta

Algoritmo 1: Algoritmo VND

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, $k = 1..., k_{max}$, que será utilizado durante o processo; Encontre uma solução inicial x.

Repetir a sequência até que nenhuma melhora seja obtida:

(1) Faça $k \leftarrow 1$;

(2) Repetir até que $k = k_{max}$:

(a) Exploração da vizinhança. Encontre o melhor vizinho x' de x, $(x' \in N_k(x))$;

(b) Mover ou não. Se a solução obtida é melhor que x, faça $x \leftarrow x'$ e $k \leftarrow 1$; Caso contrário, $k \leftarrow k + 1$.

maneira, se o algoritmo VND for usado de forma independente, então podem ser aceitáveis vizinhanças de cardinalidade elevada, isto é, aceitar um número elevado de vizinhos. Por outro lado, se o algoritmo VND for usado em uma estrutura VNS mais complexa (ver, por exemplo, o algoritmo GVNS), então as vizinhanças devem ter cardinalidade não muito grande. Em qualquer contexto, pode ser recomendável usar técnicas de redução de vizinhança da mesma forma que as utilizadas no algoritmo *tabu search*.

- Em relação ao item anterior, podemos modificar o algoritmo VND para que não seja muito lento. Observamos que no passo 2(a), todos os vizinhos são avaliados para identificar o melhor. Assim, apenas o melhor vizinho é comparado com a solução corrente x. Esse passo pode ser modificado de forma que os vizinhos sejam ordenados e assim que for identificado um vizinho melhor que a solução corrente então devemos realizar a transição. Nesse contexto, o tamanho da vizinhança não é crítico.

- Na lógica do algoritmo VND está implícito que as vizinhanças mudam de tamanho de forma crescente, isto é, a vizinhança N_1 tem menos elementos que a vizinhança N_2 e assim sucessivamente. Assim, a estrutura do algoritmo VND prioriza a vizinhança de menor tamanho, isto é, sempre que for encontrada uma nova incumbente e após executada a transição, o algoritmo VND retorna para a vizinhança N_1. Essa proposta prioriza claramente a lógica da intensificação. Uma outra alternativa pode ser avaliar as vizinhanças em sequência, isto é, em cada passo devemos analisar uma estrutura de vizinhança. Essa proposta pode tornar o VND mais lento, mas enriquece o processo com maior diversificação.

- As estruturas de vizinhanças podem não ser suficientes para encontrar o ótimo

global. Observamos que as estruturas de vizinhanças crescem em complexidade (cada nova estrutura tem um número muito maior de elementos ou vizinhos). Assim, se as estruturas de vizinhanças não estão adequadamente projetadas pode não ser possível encontrar as soluções ótimas de problemas complexos. Portanto, definir de forma adequada as estruturas de vizinhança é crucial em algoritmos tipo VNS.

- Assim como na heurística SDH e nas metaheurísticas tais como *tabu search* e *simulated annealing*, o algoritmo VND (e todos os algoritmos tipo VNS) precisam de uma proposta de solução inicial x para iniciar o processo de otimização. No caso do VND, recomenda-se que essa proposta de solução inicial seja de boa qualidade). Portanto, essa proposta de solução inicial pode ser encontrada usando um algoritmo heurístico construtivo (AHC) de bom desempenho.

- O algoritmo VND termina quando não existe proposta de solução vizinha de melhor qualidade que a solução corrente no último nível de vizinhança $N_{k_{max}}$. Portanto, o algoritmo VND não precisa da especificação de um critério de parada, mas pode ser especificada uma estratégia de parada alternativa.

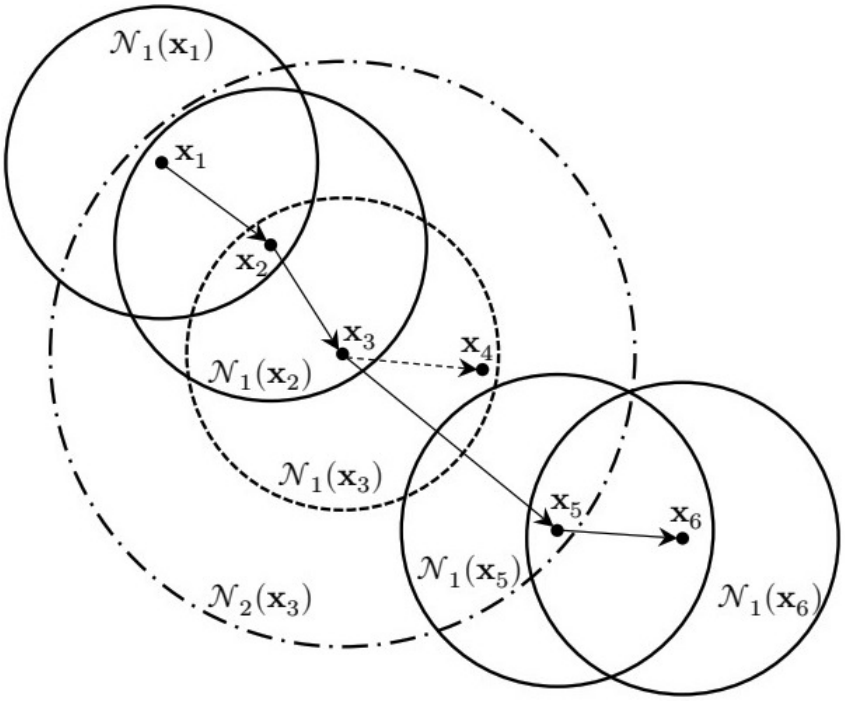


Figura 5.2: Processo de busca do algoritmo VND.

Fonte: POSSAGNOLO [42].

5.2.2 Algoritmo RVNS

O segundo tipo de algoritmo VNS é o chamado *Reduced Variable Neighborhood Search* (RVNS). Este tipo de algoritmo está inspirado em dois aspectos fundamentais no processo de busca relacionados com a intensificação e a diversificação. Por um lado, o **Fato 3** afirma que na região de um ótimo local normalmente existem outras soluções ótimas locais que podem ser encontradas a partir de um ótimo local inicial e, portanto, deve-se montar uma estratégia de intensificação para tentar encontrar esses ótimos locais. Por outro lado, sair de um ótimo local de qualidade para encontrar um ótimo local de uma região mais distante exige uma estratégia que implique em uma mudança mais radical na caracterização das vizinhanças, especialmente em problemas onde um ótimo local tem uma composição complexa. Assim, uma busca que contemple ambos os aspectos (intensificação e diversificação) pode permitir encontrar ótimos locais de uma mesma região e pode permitir ao processo de busca sair para ótimo locais de regiões mais distantes do ponto de ótimo local corrente. Esse tipo de estratégia, isto é, uma composição entre intensificação e diversificação foi incorporada no algoritmo RVNS. O algoritmo RVNS assume a seguinte estrutura:

Algoritmo 2: Algoritmo VNS reduzida - RVNS

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, para $k = 1..., k_{max}$, que será utilizado durante o processo; Encontre uma solução inicial x; Estabeleça uma condição de parada;

Repetir a sequência até que condição de parada seja satisfeita:

(1) Faça $k \leftarrow 1$;

(2) Repetir os passos seguintes até que $k = k_{max}$:

- **(a) Exploração da vizinhança.** Encontre uma solução aleatória x' da k-ésima vizinhança de x, $(x' \in N_k(x))$;
- **(b) Mover ou não.** Se a solução obtida é melhor que x, faça $x \leftarrow x'$ e continue a busca em $N_1(k \leftarrow 1)$; Caso contrário, $k \leftarrow k + 1$.

Uma característica fundamental no algoritmo RVNS é que ele é capaz de produzir uma escolha de vizinhanças mais dinâmica escolhendo vizinhos de todas as estruturas de vizinhanças (diversificação) e priorizando a primeira estrutura de vizinhança (intensificação) sempre que for realizada a transição da proposta de solução corrente para uma proposta de solução vizinha de melhor qualidade. Entretanto, uma componente importante da estrutura RVNS é a capacidade de encontrar novas regiões promissoras a partir de um ótimo local. Adicionalmente, deve-se observar que o tamanho de cada estrutura de vizinhança não é um problema crítico no algoritmo RVNS como pode acontecer no algoritmo VND. O algoritmo

RVNS também pode ser usado de forma independente ou pode ser integrado em uma estrutura mais complexa de algoritmo VNS.

Em relação ao algoritmo RVNS, deve-se fazer as seguintes observações:

- No passo 2(a), é escolhida de forma aleatória uma solução vizinha da solução corrente para uma certa estrutura de vizinhança. Nesse contexto, a estratégia RVNS procura, de forma aleatória, uma solução vizinha de melhor qualidade usando as estruturas de vizinhança simples (intensificação) e também uma solução vizinha de melhor qualidade usando as estruturas de vizinhança mais complexas (diversificação). Desta forma é possível encontrar uma solução na vizinhança próxima da solução corrente ou uma solução correspondente a uma outra região de ótimo local distante da solução atual.

- Um fato muito importante é saber definir o quanto deve-se insistir na busca na vizinhança de um ótimo local. Por um lado, existe o Fato 3 que estipula que em muitos problemas um ótimo local geralmente se encontra muito próximo de outras soluções ótimas locais e localizado em pequenas regiões do espaço de busca. Portanto, quando for encontrado um ótimo local, acredita-se então que essa solução contém informação implícita das outras soluções ótimas locais e, provavelmente, da própria solução ótima global. Desta forma, é natural explorar inicialmente regiões nas proximidades de um ótimo local. Entretanto, se essa região se encontrar muito distante de outras regiões de ótimo local, então a estratégia de busca local atual pode não ser suficiente para sair dessa região de ótimo local. Portanto, é necessário adicionar uma estratégia mais dinâmica no processo de busca. Essa possibilidade é introduzida no algoritmo RVNS.

- No algoritmo RVNS escolhe-se um conjunto de estruturas de vizinhança N_k, $k = 1, ..., k_{max}$ caracterizadas em relação com a solução corrente x (que pode ou não ser um ótimo local). Geralmente essas vizinhanças estão aninhadas, isto é, os elementos da vizinhança N_1 também são elementos na vizinhança N_2 e assim sucessivamente.Em alguns problemas não existe esse tipo de relação. Nesse contexto, escolhemos aleatoriamente um vizinho na primeira vizinhança. Se esse vizinho escolhido é melhor que a solução corrente então esse vizinho a substitui e reiniciamos o processo de busca a partir da nova solução. Em caso contrário, passamos para o nível de vizinhança seguinte. Se for realizada a transição em um nível superior de vizinhança então a busca volta para o primeiro nível. Depois que todos os níveis de vizinhança são avaliados o processo volta para o primeiro nível e o processo de busca deve continuar até cumprir um critério de parada. Esse comportamento pode ver verificado na Figura 5.3.

- O tamanho das estruturas de vizinhança não é um problema crítico no algoritmo RVNS já que em cada nível de vizinhança é escolhida uma solução vizinha de forma aleatória para tentar realizar a transição. Por esse motivo o algoritmo RVNS tem uma componente estocástica e em cada corrida podem ser encontradas soluções ótimas locais diferentes. Deve-se observar que o algoritmo VND é integralmente determinístico, isto, uma vez escolhidas a proposta de solução inicial e as estruturas de vizinhança, o algoritmo encontra a mesma solução final em qualquer corrida do algoritmo.

- Como as vizinhanças são cada vez maiores em tamanho então o algoritmo RVNS explora com maior frequência vizinhanças de níveis elevados já que em cada nível de vizinhança ela não é avaliada inteira como acontece com o algoritmo VND. No algoritmo RVNS escolhemos apenas um vizinho e se esse vizinho for de melhor qualidade então voltamos para o primeiro nível de vizinhança e continuamos com o processo de intensificação. Em caso contrário, passa-se para o nível seguinte. Assim, a vizinhança no algoritmo RVNS pode ser maior em tamanho e complexidade quando comparado com o algoritmo VND.

- Assim como o algoritmo VND, o algoritmo RVNS precisa de essa proposta de solução inicial que seja de boa qualidade. Essa proposta de solução inicial pode ser encontrada usando um algoritmo heurístico construtivo (AHC) de bom desempenho.

- O algoritmo RVNS precisa definir um critério de parada claramente especificado

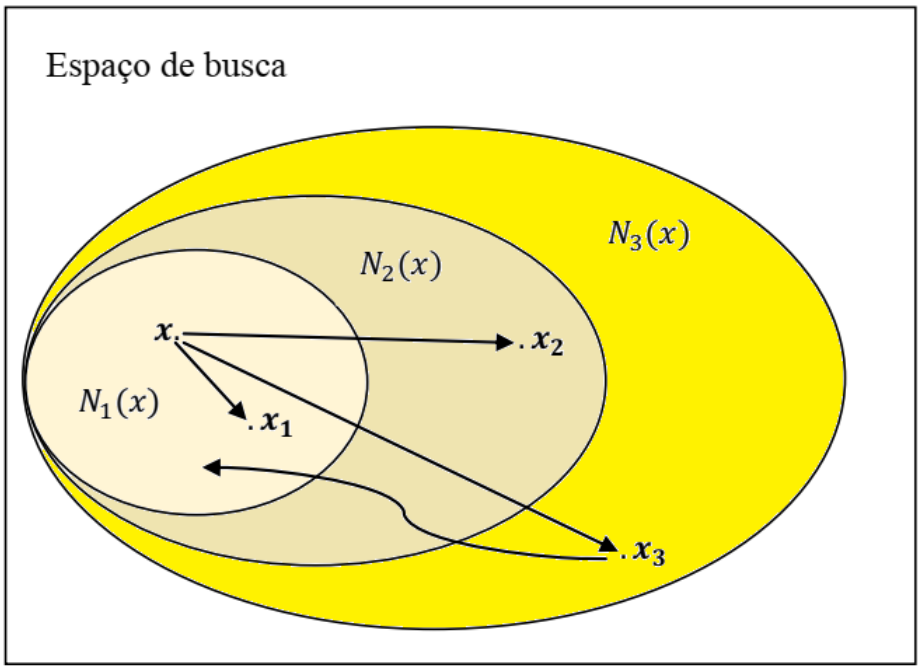


Figura 5.3: Comportamento do Algoritmo RVNS.

Fonte: Adaptado de SOUZA [43]

5.2.3 Algoritmo BVNS

Algoritmos VNS mais eficientes podem ser formulados integrando as características do algoritmo VND, que permite encontrar ótimos locais de qualidade, e do algoritmo RVNS que permite encontrar novas regiões promissoras a partir de um ótimo local. Assim, juntando as características dos dois algoritmos já apresentados podem ser formulados dois tipos de algoritmos VNS que geralmente apresentam excelente desempenho. Esses algoritmos são chamados de *Basic Variable Neighborhood Search* (BVNS) e *General Variable Neighborhood Search* (GVNS). O algoritmo BVNS combina a busca local com mudanças sistemáticas da vizinhança em torno de uma solução ótima local. A estrutura do algoritmo BVNS assume a seguinte forma:

O algoritmo BVNS combina a busca local com mudanças sistemáticas da vizinhança em torno do ótimo local encontrado. A lógica de trabalho do algoritmo BVNS é muito interessante. Inicialmente deve-se escolher as k estruturas de vizinhança. O processo de otimização é iniciado de uma solução x e na vizinhança $N_1(x)$. Na sequência escolhemos de forma aleatória um vizinho x' de x em $N_1(x)$. A partir de x' é iniciado um processo de busca local para encontrar um ótimo local x''.

Nesse contexto podem acontecer 3 casos:

1. Se $x'' = x$ significa que x já era o ótimo local dessa região e, portanto, devemos

Algoritmo 3: Algoritmo BVNS

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, para $k = 1..., k_{max}$, que será utilizado durante o processo; Encontre uma solução inicial x; Estabeleça uma condição de parada;

Repetir a sequência até que condição de parada seja satisfeita:

(1) Faça $k \leftarrow 1$;

(2) Repetir os passos seguintes até que $k = k_{max}$:

(a) Agitação. Gere aleatoriamente uma solução x' da k-ésima vizinhança de x, $(x' \in N_k(x))$;

(b) Busca local. Aplicar algum método de busca local a partir de x'. Seja x'' o ótimo local desse processo de busca.

(c) Mover ou não. Se a solução x'' obtida é melhor que x então faça $x \leftarrow x''$ e continue a busca em $N_1(k \leftarrow 1)$; Caso contrário, $k \leftarrow k + 1$.

mudar para outro nível de vizinhança ($N_2(x)$ neste caso);

2. Se x'' é de pior qualidade que x então foi encontrado um ótimo local de pior qualidade que a incumbente x e também devemos mudar de vizinhança;

3. Se x'' é de melhor qualidade que x então significa que foi encontrada uma solução melhor que a incumbente e, portanto, devemos atualiza-la e reiniciar a busca permanecendo na vizinhança $N_1(x)$. Se x'' é de melhor qualidade que x então significa que foi encontrada uma solução melhor que a incumbente e, portanto, devemos atualiza-la e reiniciar a busca permanecendo na vizinhança $N_1(x)$.

Se no processo de otimização atingimos o último nível de vizinhança sem encontrar uma solução melhor que a incumbente então o processo volta a busca no primeiro nível de vizinhança. O processo termina quando se cumpre um critério de parada. O critério de parada pode ser especificado de várias formas tais como por exemplo, o tempo máximo de processamento, ou número máximo de iterações, ou número máximo de iterações desde a última melhoria. Em relação ao algoritmo BVNS devemos fazer as seguintes observações:

- Em qualquer iteração do processo sempre que a busca local encontrar uma nova incumbente voltamos para a vizinhança $N_1(x)$ e sempre que a busca local encontra uma solução de igual ou pior qualidade que a incumbente então mudamos para uma vizinhança mais complexa. Essa estratégia e a escolha aleatória do vizinho da incumbente x evita ciclagem e permite encontrar ótimos locais distantes da incumbente corrente.

- No passo 2(a), é escolhida de forma aleatória uma solução vizinha da solução corrente para uma certa estrutura de vizinhança. Essa proposta de solução vizinha escolhida de forma aleatória é usada como solução inicial de um processo de otimização de busca local. Assim, essa escolha aleatória fornece um comportamento estocástico ao algoritmo BVNS.

- No passo 2(b), é implementado um processo de otimização chamado de busca local. Em princípio, pode-se implementar qualquer estratégia de busca local. Entretanto, fica evidente que na formulação inicial do algoritmo BVNS, essa estratégia de busca local seria a heurística SDH.

- O tamanho das estruturas de vizinhança não é um problema crítico no algoritmo BVNS já que em cada nível de vizinhança é escolhida uma solução vizinha de forma aleatória para iniciar o processo de otimização de busca local. Assim, o tempo de processamento do algoritmo BVNS depende mais fortemente da estratégia implementada na fase de busca local no passo 2(b) (da forma de definir a vizinhança se for usada a heurística SDH).

- Assim como o algoritmo VND e RVNS, o algoritmo BVNS precisa de uma proposta de solução inicial x para iniciar o processo de otimização. Assim, recomenda-se que essa proposta de solução inicial seja de boa qualidade. Portanto, essa proposta de solução inicial pode ser encontrada usando um AHC de bom desempenho.

- O algoritmo BVNS precisa definir um critério de parada claramente especificado.

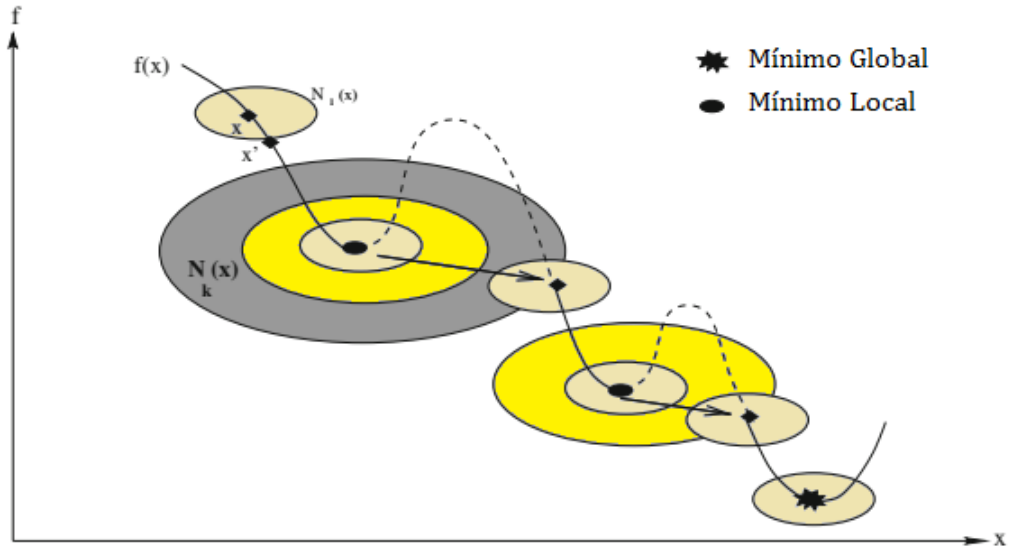


Figura 5.4: Comportamento do Algoritmo BVNS.

Fonte: Adaptado de HANSEN *et al.* [39]

5.2.4 Algoritmo GVNS

A Busca em Vizinhança Variável Geral, *General Variable Neighborhood Search* (GVNS) é um algoritmo generalizado a partir do algoritmo BVNS, obtido pela fusão dos algoritmos VND e RVNS, onde todas as observações realizadas para o algoritmo BVNS permanecem válidas no GVNS. Lembra-se que o algoritmo BVNS pode ser elaborado utilizando qualquer estratégia heurística, em particular usando as extensões da metaheurística VNS.

O algoritmo GVNS baseia-se nos critérios de busca local em torno de uma solução corrente e da análise de transição com intuito de explorar novas regiões, o que é análogo ao algoritmo BVNS, no entanto a mudança fundamental está na fase de melhoria da solução inicial usando o algoritmo RVNS. As peculiaridades dos algoritmos VND e RVNS destacam-se como aspectos promissores para aplicação do algoritmo GVNS, uma vez que o algoritmo VND realiza uma busca determinística de forma completa o que prioriza o melhoramento da solução corrente na fase de busca local e o algoritmo RVNS realiza uma busca estocástica o que é favorável na busca por outras regiões promissoras.

O algoritmo GVNS é estruturado da seguinte maneira:

Algoritmo 4: Algoritmo GVNS

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, para $k = 1..., k_{max}$, que será utilizado durante o processo de agitação; Selecione um conjunto de estruturas de vizinhanças N_p, $p = 1, ..., p_{max}$. que será usado durante a descida;Encontre uma solução inicial x; Estabeleça uma condição de parada;

Repetir a sequência até que condição de parada seja satisfeita:

(1) Faça $k \leftarrow 1$;

(2) Repetir os passos seguintes até que $k = k_{max}$:

- **(a) Agitação.** Gere aleatoriamente uma solução vizinha x' da k-ésima vizinhança de x, $(x' \in N_k(x))$;
- **(b) Busca local.** Aplicar a busca VND com as estruturas N'_p, para $p = 1, ..., p_{max}$, denote por x'' o mínimo obtido por essa busca;
- **(c) Mover ou não.** Se a solução x'' obtida é melhor que x então faça $x \leftarrow x''$ e continue a busca em $N_1(k \leftarrow 1)$; Caso contrário, $k \leftarrow k + 1$.

Na elaboração das estruturas das vizinhanças, segundo HANSEN e MLADENOVIĆ [44], SANTOS [45] deve-se priorizar e ter uma atenção especial para os seguintes aspectos:

i) Definir as formas ou esquemas para determinação das estruturas das vizinhanças

de modo que o conjunto total de vizinhança possa obter a maior região possível para exploração;

ii) Estabelecer as propriedades nas vizinhanças de modo fazer uma boa exploração a fim de encontrar vizinhos da incumbente de melhor qualidade;

iii) Estabelecer critérios coerentes de modo a obter uma dimensão (mensuração) para cada vizinhança.

Os dois primeiros itens chamam a atenção para o fato que a metaheurística VNS possa explorar as melhores regiões e executar a busca com mais eficiência e agilidade. Além disso, colocam em evidencia todos os cuidados para que a busca não seja bloqueada em uma determinada região, enquanto existir outra melhor. Por outro lado, o último critério atenta para a coerência na elaboração das estruturas das vizinhanças, de modo que, seja possível mensurar e criar níveis de vizinhança e, consequentemente ordená-las por grau de complexidade.

Em relação ao algoritmo GVNS, podemos fazer as seguintes observações:

- As estruturas de vizinhanças presentes no algoritmo GVNS devem ser adequadamente projetadas para que o algoritmo GVNS encontre a solução ótima global ou soluções quase-ótimas. Nesse projeto, notamos que a ideia fundamental consiste em encontrar as melhores regiões de ótimo local e o mais rapidamente possível. Na verdade a única forma de garantir que o processo não fique estacionado em um ótimo local é quando a união de todas as estruturas de vizinhança reproduzem o espaço de busca completo.

- Existe ainda a questão de como proceder se as estruturas de vizinhanças geram vizinhos infactíveis. Este incômodo aparece com muita frequência em alguns tipos de problemas. Uma alternativa simples é descartar as soluções vizinhas infactíveis e migrar para vizinhanças com grau de complexidade maior, isto é, migrando a busca em novas regiões. Outra possibilidade muito usada em outras metaheurísticas, é penalizar a função objetivo incorporando as infactibilidades na função objetivo.

5.3 Estruturação de Vizinhanças na Metaheurística VNS

Durante o processo da construção das vizinhanças para o algoritmo VNS, em qualquer de suas extensões, deve-se atentar intuitivamente na lógica empregada para que a união das vizinhanças de qualquer solução factível x contenha todo o conjunto X de soluções factíveis. Isto pode ser traduzido matematicamente da seguinte maneira:

$$X \subseteq N_1(x) \cup N_2(x) \cup N_3(x) \cup ... \cup N_{k_{max}} \forall x \in X,$$

onde X é o conjunto das soluções factíveis

Estes conjuntos de vizinhanças $N_i(x)$, podem cobrir X sem necessariamente particionar, o que é mais fácil de implementar quando utilizam-se, por exemplo, as estruturas de vizinhanças encaixantes (aninhadas) da seguinte forma:

$$N_1(x) \subset N_2(x) \subset N_3(x) \subset ... \subset N_{k_{max}} \forall x \in X.$$

Se estas propriedades não forem asseguradas, não se pode garantir que o conjunto de soluções viáveis, X, seja completamente percorrido. Ressalta-se ainda, que vizinhanças aninhadas podem ser facilmente obtidas em muitos problemas de otimização combinatória, por exemplo, definindo uma primeira vizinhança $N_1(x)$ por um tipo de movimento (por exemplo, 2-opt no problema do caixeiro viajante) e então iterando k para obter as vizinhanças $N_k(x)$, para $k = 2, ..., k_{max}$. Assim definidas, as vizinhanças têm a propriedade de que os seus tamanhos aumentam conforme o acréscimo de k. Além disso, ao percorrer várias vezes a sucessão inteira de vizinhanças, a primeira delas será explorada mais completamente que as demais, o que é fundamental, principalmente quando atentamos para a **Fato 3**, isto é, ótimos locais tendem a estarem próximos uns dos outros.

5.3.1 Extensões da VNS

Os autores HANSEN e MLADENOVIĆ [46], propuseram varias extensões da VNS para resolver problemas de grandes instâncias. Em seu artigo [41] encontramos em detalhes um tutorial sobre VNS, incluindo extensões, exemplos ilustrativos e questões em relação à implementação. Algumas evoluções da VNS são descritas nas próximas seções.

Na literatura aparecem diversas formas de estender a VNS, nas quais geralmente são acrescentadas algumas características adicionais ás quatro variações principais do VNS observadas anteriormente.

As primeiras extensões derivaram-se diretamente da VNS básica. Como já vimos da seção anterior a BVNS é um método do tipo descendente obtido através da primeira melhoria com aleatorização. Sem muito esforço adicional pode ser transformado em um método ascendente-descendente: basta fazer no passo (2c) $x \leftarrow x''$ com alguma probabilidade, ainda que a solução encontrada seja pior que a solução incumbente. Também pode ser transformado em uma busca do melhor movimento: aplicando um movimento na melhor vizinhança $k*$ entre todas as k_{max} vizinhanças.

Outras extensões da VNS consistem em encontrar a solução x' no passo (2a) como sendo a melhor entre b (um parâmetro) soluções geradas aleatoriamente na k-ésima vizinhança, ou introduzir k_{min} e k_{passo}, dois parâmetros que controlam o

processo de troca de vizinhança: no algoritmo anterior, em vez de $k \leftarrow 1$, fazer $k \leftarrow k_{min}$, e em vez de fazer $k \leftarrow k + 1$, fazer $k \leftarrow k + k_{passo}$.

5.3.1.1 Algoritmo SVNS

De uma forma geral, a VNS encontra soluções melhores ou de igual qualidade que as heurísticas de partidas múltiplas, e soluções muito melhores em casos de problemas com muitos ótimos locais. Isso ocorre devido ao **Fato 3** mencionado anteriormente que afirma que muitos problemas agrupam ótimos locais, pode acontecer que, em alguns casos, os ótimos locais estejam dispersos em várias regiões do espaço de busca, e possivelmente longe dos vales que contêm as soluções quase ótimas.

Uma vez que se tenha encontrado a melhor solução em uma região grande, é necessário muito esforço para que se consiga obter uma solução melhor. Ao se considerar vizinhanças maiores, as soluções geradas aleatoriamente podem diferenciar-se substancialmente da solução atual; e por isso a VNS se degenera em uma heurística de partidas múltiplas (no sentido de que se realizam, iterativamente, descidas com soluções geradas aleatoriamente). Além disso, se a solução incumbente atual não está no vale mais profundo esta informação é, em parte irrelevante. Seria então interessante modificar o esquema da VNS para explorar melhor aqueles vales que estão mais longe da solução incumbente. Isto pode ser feito aceitando realizar um movimento para uma solução de valor próximo ao da incumbente atual, mas não necessariamente melhor, desde que esta solução esteja longe da incumbente.

O esquema VNS modificado para esta variante chamada *Skewed Variable Neighborhood Search* (SVNS) é apresentado abaixo. Portanto, a SVNS incorpora uma compensação da distância entre a solução gerada e a solução atual para evitar este inconveniente.

Evidentemente, poderiam ser consideradas fórmulas mais complicadas para sair de um vale. E algumas perguntas devem ser respondidas quando se aplica a SVNS:

(i) O problema considerado apresenta função objetivo quase convexa, ou existem vários vales profundos e separados?

(ii) Como α deveria ser escolhido?

Estas perguntas podem ser respondidas, até certo ponto, utilizando inicialmente uma versão VNS de partidas múltiplas, iniciando a VNS em vários pontos aleatórios e em um período pequeno. Depois considerar aquela posição que se encontra o melhor ótimo local e observar se eles estão agrupados ou dispersos. Então se pode calcular os valores em função de distancia do ótimo local correspondente para a melhor solução conhecida e escolher α como uma fração da inclinação média.

Algoritmo 5: Algoritmo SVNS

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, para $k = 1..., k_{max}$, que será utilizado durante a busca; Encontre uma solução inicial x e seu valor $f(x)$; fazer $n^* \leftarrow x$, $f^* \leftarrow f(x)$ Estabeleça uma condição de parada e um parâmento α;

Repetir a sequência até que condição de parada seja satisfeita:

(1) Faça $k \leftarrow 1$;

(2) Repetir os passos seguintes até que $k = k_{max}$:

- **(a) Agitação.** Gere aleatoriamente uma solução vizinha x' da k-ésima vizinhança de x, $(x' \in N_k(x))$;
- **(b) Busca local.** Aplicar o método de busca local utilizando x' como solução inicial; denote por x'' o ótimo local encontrado;
- **(c) Melhorar ou não.** Se $f(x'') < f^*$ fazer $f^* \leftarrow f(x'')$ e $n^* \leftarrow x''$
- **(d) Mover ou não.** Se $f(x'') - \alpha\rho(x, x'') < f(x)$ fazer $x \leftarrow x''$ e $k \leftarrow 1$; Caso contrário, fazer $k \leftarrow k + 1$.

5.3.1.2 Algoritmo VNDS

A busca através de vizinhança variável com decomposição, *Variable Neighborhood Decomposition Search* (VNDS) apresentada em HANSEN e MLADENOVIĆ [46] estende a VNS em um esquema de vizinhança variável em dois níveis baseados na decomposição do problema. Seus passos são apresentados no algoritmo abaixo.

Note que a única diferença entre a VNS básica e a VNDS está no passo (2b): em vez de aplicar algum método de busca local no espaço completo X (começando em $x' \in N_k(x)$), na VNDS se resolve, em cada iteração, um subproblema em um subespaço com $X_k \subseteq N_k(x)$ com $x' \in X_k$. Quando a busca local utilizada neste passo é também a VNS, aparece então um esquema VNS em dois níveis. A VNDS pode ser vista com a inclusão do esquema de aproximações sucessivas, que historicamente tem sido muito utilizada em otimização combinatória, na estrutura da VNS.

Algoritmo 6: Algoritmo VNSD

Inicialização: Selecione um conjunto de estruturas de vizinhanças N_k, para $k = 1..., k_{max}$, que será utilizado durante o processo de descida; Encontre uma solução inicial x; Estabeleça uma condição de parada;

Repetir a sequência até que condição de parada seja satisfeita:

(1) Faça $k \leftarrow 1$;

(2) Repetir os passos seguintes até que $k = k_{max}$:

(a) Agitação. Gere aleatoriamente uma solução x' da k-ésima vizinhança de x, $(x' \in N_k(x))$; denote-se por y o conjunto de atributos da solução presente em x', mas não em x, ou seja, $y = x' - x$;

(b) Busca local. Determinar o ótimo local no espaço de y por inspeção, ou por alguma técnica heurística; denote por y' a melhor solução encontrada e por x'' a solução correspondente no espaço completo X $(x'' = (x' \setminus y) \cup y')$;

(c) Mover ou não. Se a solução x'' obtida é melhor que x então faça $x \leftarrow x''$ e continue a busca em $N_1(k \leftarrow 1)$; Caso contrário, $k \leftarrow k + 1$.

5.3.1.3 VNS Híbrida

O VNS é baseado na exploração de um modelo dinâmico de vizinhança. Ao contrário de outras metaheurísticas baseadas em métodos de busca local, o VNS permite alterações na estrutura da vizinhança ao longo da pesquisa explore mais e mais vizinhanças da melhor solução atual encontrada x.

Considerando que a troca sistemática de estruturas de vizinhanças é uma ferramenta muito simples e robusta, outra forma de estender a VNS é incorpora-la a outras metaheurísticas. Segundo HANSEN *et al.* [40] VNS também foi hibridizado com outras metaheurísticas (*Tabu Search, Path Relinking, Simulated Annealing, Greedy Randomized Adaptive Search Procedures* (GRASP), *Genetic Algorithm, Particle Swarm Optimization*, etc)

Na busca tabu (*Tabu Search*, TS) [36];[47] geralmente utiliza-se uma estrutura de vizinhança com relação a qual realiza movimentos de descida e subida explorando diferentes tipos de memórias. A princípio, há duas maneiras de se fazer hibridismo com VNS e TS: A primeira consiste em usar algum tipo de memória para orientar a busca dentro da VNS, ou usar a VNS dentro da TS. Em (BURKE *et al.*

[48]; KOVAČEVIĆ-VUJČIĆ *et al.* [49]; BRÄYSY [50]) é proposto hibridismo do primeiro tipo, e em (BRIMBERG *et al.* [51]; DAVIDOVIC *et al.* [52] ;HANSEN e MLADENOVIĆ [46]), do segundo tipo.

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) (FEO e RESENDE [53]) apresenta duas fases: na primeira fase se constroem soluções utilizando um procedimento guloso aleatório, e na segunda, as soluções são melhoradas por alguma busca local ou um método enumerativo. Uma forma natural de fazer hibridismo com VNS e GRASP é utilizar a VNS na segunda fase de GRASP.

Uma primeira tentativa no sentido de integrar a VNS ao GRASP foi realizada por Martins et al. [54] A fase de construção de sua heurística híbrida para o problema de Steiner nos grafos segue a estratégia aleatória gulosa do GRASP, enquanto a fase de busca local utiliza duas estruturas de vizinhança diferentes como estratégia VND. Sua heurística foi posteriormente aprimorada por Ribeiro, Uchoa e Werneck [55], um dos principais componentes do novo algoritmo sendo outra estratégia para a exploração de diferentes vizinhanças. Festa et al. [56]) estudaram diferentes variantes e combinações de GRASP e VNS para o problema MAX-CUT, encontrando e melhorando algumas das soluções que na época eram as soluções mais conhecidas para algumas instâncias abertas da literatura. Outros autores estudaram várias hibridizações do GRASP, incluindo o VNS, podem ser vistos em ANDREATTA e RIBEIRO [57]; CANUTO *et al.* [58]; OCHI *et al.* [59];

A busca com partidas múltiplas (*Multi Start Search*, MS) (CORBERÁN *et al.* [60]) é uma metaheurística clássica que, para evitar a estagnação de um procedimento de descida em um ótimo local, simplesmente reinicia a busca através de outra solução. Em (BELACEL *et al.* [61]) é proposta e analisada uma heurística híbrida entre a VNS e a MS, consistindo em reiniciar a VNS a partir de uma solução gerada aleatoriamente no espaço X, quando a busca converge em um mínimo local, por não encontrar nenhuma melhora através das vizinhanças $N_1(x), N_2(x), ..., N_k(x)$ da solução x.

5.3.2 VNS em Problemas de Otimização

A VNS foi aplicada com grande êxito em muitos problemas de otimização combinatória, entre os quais se encontram os problemas mais relevantes de planejamento logístico: o problema de **empacotamento**, de localização e de rotas. Adicionalmente, também foram abordados problemas onde as soluções são certos conjuntos de arestas, como as arvores, e alguns problemas de otimização contınua. Os problemas de empacotamento constituem uma das classes de problemas importantes no contexto de logística e distribuição. Em FLESZAR e HINDI [62] foi utilizada a VNS básica em uma das versões básicas: o problema de empacotamento unidimensional

(*bin-packing problem, BPP*), em que se tem de empacotar um conjunto de objetos de diferentes tamanhos no menor número de caixas ou segmentos de capacidade fixa.

Os problemas de **localização**, muito relevantes no planejamento logístico, e os problemas de **agrupamento** (*clustering*) têm características comuns, e por isso permitem que sejam tratados de forma similar em buscas metaheurísticas. O problema das p-medianas é o problema mais extensamente estudado em Teoria de Localização. Consiste em determinar p locais, entre um conjunto de m localizações possíveis para um serviço, de forma que se minimize a soma das distâncias de n-usuários a seu posto de serviço mais próximo. Este problema foi abordado em MLADENOVIĆ e HANSEN [38], em WHITAKER [63] e em HANSEN e MLADENOVIĆ [46] utilizando a VNS básica, a VNS reduzida e a VNDS, respectivamente. Diversas variantes da VNS foram aplicadas em BRIMBERG *et al.* [51] ao problema de janelas múltiplas, que é uma versão contınua do problema das p-medianas, onde as p localizações podem ser escolhidas em todo o plano, onde também estão localizados os usuários.

Outro tipo de problema de otimização combinatória importante, em relação a todo contexto de planejamento logístico, são os problemas de rotas. Tanto as versões clássicas do problema do caixeiro viajante (*Traveling Salesman Problem*, TSP), do problema de roteamento de veículos (*Vehicle Routing Problem*, VRP), como algumas de suas extensões foram abordadas com a VNS. O TSP consiste em dadas n cidades com as distancias e custos entre elas, encontrar uma rota de custo mínimo. Em MLADENOVIĆ e HANSEN [38] são aplicados procedimentos do tipo VNS com distintos tipos de movimentos para o TSP. Em OCHI *et al.* [59] é utilizado a VNS para resolver uma extensão do TSP denominada problema do comprador, em que dada uma partição do conjunto de clientes, deve-se visitar pelo menos um dos conjuntos de cada partição.

Em SILVA *et al.* [64] são apresentados algoritmos baseados em GRASP e VNS para resolver generalizações do TSP, onde apenas um conjunto de cidades é visitado, e os autores concluem que a combinação destas metaheurísticas apresenta resultados bem expressivos.

O problema de roteamento de veículos (*Velhice Routing Problem*, VRP) consiste em determinar rotas a partir de um depósito para visitar um conjunto de clientes com um custo mínimo. Os clientes têm demandas conhecidas e os veículos possuem capacidade limitada. Em BRÄYSY [50] foram aplicados a VNS e VND na resolução de uma variante deste problema, onde cada cliente deve ser visitado em certo intervalo de tempo, que se denomina VRPTW (*Velhice Routing Problem Time Windows*).

Outros problemas com múltiplas aplicações importantes cujas soluções são **árvores**, como o problema de *Steiner*, onde se deve determinar a árvore de menor

distância que conecta todos os n terminais em um grafo. Foram testadas várias versões da VND para estes problemas em MARTINS *et al.* [54] RIBEIRO *et al.* [55]. Em outras versões em que o grau de cada vértice no subgrafo está incluído, também foram testados a VNS com sucesso em RIBEIRO e SOUZA [65].

Em CANUTO *et al.* [58] foi abordado o problema de determinar a subárvore que minimiza a soma das distâncias das arestas incluídas no grafo e o peso dos vértices conectados. Em ANDREATTA e RIBEIRO [57] foi aplicada a VND ao problema da arvore filogenética (*the Phylogeny Problem*). A efetividade da VNS foi provada em FESTA *et al.* [56] para resolver problemas de corte máximo, onde o objetivo é dividir o grafo em duas partes de forma que a soma dos pesos das arestas que os unem seja máxima.

Em QU e BURKE [66] é apresentada uma versão de hiperheurística com VNS aplicada ao problema de tabela de horários de exames (*Exam Timetabling Problem*) com comparações entre TS e *Iterated Local Search*.

Em BRIMBERG *et al.* [67] foi realizada uma análise das propriedades de convergência da metaheurísticas VNS em condições gerais. Também é apresentada uma versão híbrida da VNS e de *Random Multistart Local Search* (MLS) aplicada ao conhecido problema de alocação contínua, *Multisource Weber Problem*, é uma explicação do fato de a técnica VNS obter resultados superiores aos obtidos com MLS.

5.4 VNS entre as Metaheurísticas

As heurísticas tradicionais em otimização realizam buscas locais de descida, e por isso encontram dificuldades de sair de ótimos locais encontrados. Entretanto, as metaheurísticas proporcionam métodos, determinísticos ou aleatórios, para escapar de ótimos locais de baixa qualidade. Como o valor de tais ótimos locais frequentemente diferem consideravelmente do valor do ótimo global, especialmente se existem muitos ótimos locais, o impacto das metaheurísticas tem sido considerável. Para compreender melhor estas questões, em HANSEN *et al.* [68] foram utilizadas três idéias para comparar outras metaheurísticas de busca com a VNS:

(i) O estudo da topografia dos ótimos locais;

(ii) Análise das trajetórias seguidas pelos processos de busca;

(iii) Fases de melhoria ou deterioração das soluções.

Cada uma dessas caraterísticas, como visto anteriormente, é tratada de forma particular nas diversas variações do VNS. Um fato que deve-se considerar sempre é quais seriam as propriedades desejáveis e que se quer encontrar na escolha de uma

metaheurísticas adequada. Algumas dessas qualidades caraterísticas garantem interesse prático e teórico do seu uso como podemos ver em HANSEN *et al.* [69], nesse artigo, são que analisadas algumas propriedades caraterísticas do VNS em relação a outras metaheurísticas. Na lista de tais propriedades incluem-se as seguintes:

1. **Simplicidade:** a metaheurística deve estar baseada num princípio simples e claro, que deve ser amplamente aplicável;

2. **Precisão:** os passos da metaheurística devem ser formulados em termos matemáticos precisos, independentemente de possíveis analogias físicas ou biológicas que possam ter sido a fonte inicial de inspiração;

3. **Coerência:** todas as etapas da heurística desenvolvidas para solucionar um determinado problema devem seguir naturalmente os princípios metaheurísticos;

4. **Eficácia:** heurísticas para problemas específicos devem fornecer soluções ótimas ou quase ótimas para todas as instâncias conhecidas ou pelo menos as mais realistas. Preferencialmente, eles devem encontrar soluções ótimas para a maioria dos problemas de referência para os quais tais soluções são conhecidas;

5. **Eficiência:** heurísticas para problemas específicos devem levar um tempo de computação moderado para fornecer soluções ótimas ou quase ótimas, ou soluções comparáveis ou melhores do que o estado-da-arte;

6. **Robustez:** o desempenho da metaheurística deve ser de qualidade garantida em vários exemplos e em varias instâncias;

7. **Facilidade:** deve se expressar claramente, ser fácil de entender e de usar;

8. **Inovação:** os princípios das metaheurísticas, e/ou eficiência e eficácia das heurísticas derivadas delas, devem levar a novos tipos de aplicação.

9. **Generalidade:** a metaheurística deve levar a bons resultados para uma ampla variedade de problemas;

10. **Interatividade:** a metaheurística deve permitir ao usuário incorporar seu conhecimento para melhorar o processo de resolução;

11. **Multiplicidade:** a metaheurística deve ser capaz de produzir várias soluções quase ótimas das quais o usuário pode escolher.

Capítulo 6

Uma abordagem metaheurística

Anteriormente, apresentamos uma formulação exata para a resolução do PRB, testes preliminares mostraram que o seu uso é pouco viável para instâncias maiores. O alto custo computacional e tempo elevado de execução nos levam a desenvolver como alternativa para esse problema uma proposta baseada na metaheurística VNS.

O VNS prossegue por um método de descida até o mínimo local e depois explora, sistemática ou aleatoriamente, vizinhanças $N_k(f)$ cada vez maiores em torno da solução corrente. O VNS é um método que não segue a trajetória como *Simulated Annealing* e não especifica movimentos proibidos como acontece na *busca tabu*, apesar de sua simplicidade, mostra-se muito eficaz.

O **loop** principal da metaheurística do VNS contém três etapas: perturbação, busca local e mudança de vizinhança.

Inicialmente, o raio k da vizinhança da solução inicial f é atualizado como k_{min}. Geram-se perturbações para tentar escapar dos vales que os mínimos locais pertencem. Na etapa de agitação, uma solução f' é encontrada dentro da vizinhança $N_k(f)$. Ocorre 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 f' para encontrar uma solução f''.

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

Repetem-se esses passos até que um critério de parada seja satisfeito. São exemplos de critérios de parada um número máximo de iterações consecutivas executadas sem melhoria da solução ou um tempo máximo de processamento.

Nesta tese, utiliza-se o tempo máximo de processamento como critério de parada.

6.1 Inicialização

Na primeira etapa da metaheurística VNS vamos construir uma solução inicial das numerações dos vértices de $G = (V, E)$. Para isso, constrói-se a estrutura de nível enraizada de um vértice inicial aleatório v. A renumeração é realizada em cada nível da estrutura de nível enraizada $\mathscr{L}(v)$. A renumeração em cada nível $L_i(v)$ é iniciada por um vértice aleatório. Quando todos os vértices do nível $L_i(v)$ estiverem numerados, renumera-se o nível $L_{i+1}(v)$ e assim sucessivamente até $L_{\ell_{(v)}}(v)$. Considere, para essa etapa, que o grafo $G = (V, E)$ é representado por uma lista de adjacências $E = \{E(1), ..., E(|V|)\}$, em que $E(i) = \{e(i, 1), ..., e(i, |E(i)|)\}$ representa os vértices adjacentes ao vértice i. A heurística inicia por um vértice v aleatório. O vértice v recebe o rótulo 1, ou seja, $f(v) = 1$, e constrói-se a estrutura de nível enraizada $\mathscr{L}(v)$. Renumeram-se os vértices por nível da estrutura de nível enraizada, iniciando-se por um vértice aleatório em cada nível $L_i(v)$, para $1 \leq i \leq L_{\ell_{(v)}}(v)$. Esse procedimento é chamado de busca em largura aleatória.

Para um nível $L_i(v)$ com vértices u_j; com $j = 1, ...; |L_i(v)|$, escolhe-se aleatoriamente um índice j^* no intervalo $[1, |L_i(v)|]$, para que u_j^* seja o primeiro vértice do nível corrente 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, $f(u) \leftarrow 2$ (porque o vértice $v \in L_0(v)$ recebeu a renumeração 1). Repete-se esse passo para cada nível da estrutura de nível enraizada. Como a construção da estrutura de nível enraizada é pela busca em largura, essa sub-rotina tem complexidade $O(|V| + |E|)$.

Observe no Algoritmo 7, a sub-rotina para a construção da solução inicial.

Algoritmo 7: Solução inicial

Input: grafo $G = (V, E)$; rotulagem f;

Output: nova rotulagem f;

```
1  início
2      foreach (v ∈ V) do
3          v.visitou ← false
4      end
5      v ← RadomVert(V);
6      q(1) ← v;
7      rot ← 1;
8      uniCor ← 1;
9      v.visitou ← true;
10     while (rot < |V|) do
11         uniProx ← 0;
12         j* ← RadomInt(1, uni);
13         for (i ← 1; i ≤ uniCor; i ← i + 1) do
14             foreach (vértice w ∈ Adj(G, q(i)) do
15                 if w.visitou = false then
16                     uniProx ← uniProx + 1;
17                     w.visitou ← true;
18                     r(uniProx) ← w
19                 end
20             end
21             f(q(j*)) ← rot;
22             rot ← rot + 1;
23             j* ← j* + 1
24             if (j* > uniCor) then
25                 j* ← 1;
26             end
27         end
28         for (j ← 1; j ≤ uniProx; j ← j + 1) do
29             q(j) ← r(j);
30             uniCor ← uniProx;
31         end
32     end
33     return f;
34 fim
```

Inicializam-se as variáveis v, o vetor q, a variável *rot* e a variável *vniCor*. O vetor q é o conjunto de vértices no nível corrente da estrutura de nível. A variável *vniCor* é o número de vértices no nível corrente da estrutura de nível enraizada. v é o vértice inicial e a variável *rot* é a rotulagem que será atribuída aos vértices. Considere que, inicialmente, $(\forall v \in V)\ v.visitou \leftarrow false$. Na estrutura de repetição para cada nas linhas 14 a 20, visitam-se todos os vértices w adjacentes a u. Marcam-se como visitados aqueles que não foram visitados, armazenando-os em r, que é o vetor dos vértices do próximo nível da estrutura de nível. Ainda, incrementa-se a variável *uniProx* em uma unidade para cada vértice w visitado. Os passos das linhas 14 a 20 são repetidos para todos os vértices pertencentes a q. Desse modo, são visitados todos os vértices adjacentes aos vértices de q, que ainda não foram visitados. Para iniciar a numeração do vetor q, na linha 12, escolhe-se, aleatoriamente, um índice j^* no intervalo $[1, |L_i(v)|]$.

Na estrutura de repetição das linhas 13 a 27, numeram-se os vértices do nível corrente. Especificamente, nas linhas 21 a 26, numeram-se os vértices do nível corrente da estrutura de nível, com a variável *rot*. Na linha 30, copia-se o conteúdo da variável *uniProx* para *uniCor*. Dessa forma, o conteúdo do vetor q estará atualizado para que o próximo nível da estrutura de nível seja numerado, na próxima iteração. O Algoritmo termina sua execução quando rótulo exceder $|V|$. O Algoritmo retorna a nova rotulagem de f.

6.2 Busca local

Em 2001, MARTÍ *et al.* [70] propuseram um Algoritmo para o PRB utilizando a metaheurística *Tabu Search* com vizinhança de troca reduzida. Uma estratégia especial que lista os candidatos à troca foi usada para aumentar a velocidade da seleção de movimentos. Esse procedimento, chamado de *Hill Climbing*, serviu de base para vários trabalhos posteriores [71, 72].

O Algoritmo *Hill Climbing* busca melhorar a qualidade da solução através das trocas dos rótulos dos vértices. Em [70], considera-se como melhora na qualidade da solução tanto a redução da largura de banda quanto a redução do número de arestas críticas desde que não implique no aumento da largura de banda.

Apresenta-se o *Hill Climbing* no Algoritmo 8.

Algoritmo 8: Hill Climbing

```
1  while Can_Improve do
2      Can_Improve = false
3      for v = 1 to |V| do
4          if B_f(v) = B_f(G) then
5              forall u such that u ∈ N(v, f) do
6                  swap(f(v), f(u)) and update (B_f(w), B_f(G)) for all
                    w ∈ (N(v) ∪ N(u))
7                  if number of critical edges reduced then
8                      Can_Improve = true
9                      break
10                 end
11                 swap(f(v), f(u)) and update (B_f(w), B_f(G)) for all
                    w ∈ (N(v) ∪ N(u))
12             end
13         end
14     end
15 end
```

Na etapa de busca local, devemos selecionar um conjunto de vértices críticos do grafo para que suas numerações sejam trocadas. O conjunto E_C de arestas críticas é definido por $E_C(f) = \{(u, v) \in E : |f(u) - f(v)| = B_f(G)\}$. O conjunto de vértices críticos V_C de $G = (V, E)$ é formado por todos os vértices de arestas de E_C.

Na formulação proposta por MARTÍ *et al.* [70], o rótulo máximo e o rótulo mínimo para o vértice que se conectam ao vértice v são definidos como:

$$f_{\min}(v) = \min\{f(u)|(u, v) \in E\} \tag{6.1}$$

$$f_{\max}(v) = \max\{f(u)|(u, v) \in E\} \tag{6.2}$$

O rótulo que minimiza a banda de v na rotulagem atual é

$$mid(v) = \lfloor \frac{f_{\max}(v) + f_{\min}(v)}{2} \rfloor \tag{6.3}$$

Para a aplicação da busca local, define-se a vizinhança reduzida para a troca de numeração de v por

$$N(v, f) = \{u : |mid(v) - f(u)| < |mid(v) - f(v)|, (u, v) \in E_C(f)\} \tag{6.4}$$

O conjunto $N(v, f)$ é formado pelos vértices cuja troca isolada de rótulos com v pode reduzir a banda de v.

Como a etapa da busca local por *Hill Climbing* acontece apenas depois da etapa de ajuste da numeração, a solução f é considerada solução corrente. Com $N(f)$, forma-se um conjunto de vértices apropriados para a troca. A busca local é aplicada no conjunto $N(f)$. Troca-se a numeração do vértice crítico v com cada numeração dos vértices $u \in N(v, f) \subseteq N(f)$ até que uma melhora em relação à solução corrente f seja constatada. Com isso, gera-se a solução f' verifica-se se uma melhora é realizada quando um vértice crítico se torna um vértice não crítico.

Nesse algoritmo, as atualizações do $B_f(v)$ e de $B_f(G)$ requer, no pior caso, $O(|V|)$ operações, pois a largura de banda de todos os vértices conectados a u e v precisa ser atualizada toda vez que é realizada uma troca.

Como o número máximo possível de elementos em $N(v, f)$ é $|V|$, atualizar $B_f(v)$ requer tempo $O(|V|)$. A complexidade pode chegar a ordem $O(|V|^3)$ para cada iteração.

Neste procedimento, verificar se uma troca melhora a solução tem custo elevado, pois a largura de banda para os vértices trocados deve ser recalculada cada vez que uma troca é executada. Como uma alternativa para a redução da complexidade da checagem de viabilidade da troca de rótulos, os autores LIM *et al.* [73] propuseram um algoritmo genético e uma modificação do Algoritmo *Hill Climbing* denominado *Improved Hill Climbing*. Basicamente, essa modificação ocorre pela substituição da linha 7 (loop mais interno) no Algoritmo Hill Climbing.

Os autores do Algoritmo *Improved Hill Climbing* enunciam e fazem uso do teorema abaixo.

Teorema 6.1 *Dada uma rotulagem f, vértices u e v, a troca de $f(u)$ por $f(v)$ reduz o número de arestas críticas sem aumentar a largura de banda se, e somente se,*

$$\begin{cases} C'(u) \leq C(u) \\ C'(v) \leq C(v) \\ C'(u) + C'(v) < C(u) + C(v) \end{cases}$$

onde $C'(u)$ e $C'(v)$ são os valores da criticalidade de u e v após a troca.

Segundo LIM *et al.* [73], é conveniente definir a função de criticalidade de um vértice como sendo

$$C(v) = \begin{cases} 0, & \text{quando } B_f(v) < B_f(G) \\ 1, & \text{quando } B_f(v) = B_f(G) \end{cases} \tag{6.5}$$

Os mesmos autores, no ano seguinte em LIM *et al.* [74], reformularam o teorema 6.1, o enunciando da seguinte forma:

Teorema 6.2 *Dada uma rotulagem f, vértices u e v, na troca de $f(u)$ por $f(v)$, a largura de banda $B_f(G)$ é reduzida ou o número de arestas críticas é reduzido sem alterar $B_f(G)$ se, e somente se,*

$$\begin{cases} C'(u) \leq C(u) \\ C'(v) \leq C(v) \\ C'(u) + C'(v) < C(u) + C(v) \end{cases}$$

onde $C'(u)$ e $C'(v)$ são os valores da criticalidade de u e v após a troca.

Apresenta-se o *Improved Hill Climbing* no Algoritmo 9.

Algoritmo 9: Improved Hill Climbing

```
1  while Can_Improve do
2      Can_Improve = false
3      for v = 1 to |V| do
4          if C(u) = 1 then
5              forall u such that u ∈ N(v, f) do
6                  if ((C'(u) ≤ C(u)) ∧ ((C'(v) ≤ C(v)) ∧ ((C'(u) + C'(v) <
                     (C(u) + C(v)) then
7                      swap(f(v), f(u))
8                      Can_Improve = true
9                      break
10                 end
11             end
12         end
13     end
14 end
```

Da mesma forma que MARTÍ *et al.* [70], os autores LIM *et al.* [73] afirmaram que uma solução melhora, não apenas quando a banda é reduzida, mas também quando o número de arestas críticas é reduzido.

6.2.1 Nova proposta de busca local

Nesta tese, apresenta-se uma proposta VNS com uma nova busca local. A principal motivação para nossa nova busca local é a comprovação que a condição do teorema 6.1 pode ser violada, uma vez que, existem trocas de u por v que valem a pena e não passam pelo critério definido pelo Algoritmo *Improved Hill Climbing*.

Considere o seguinte contra-exemplo:

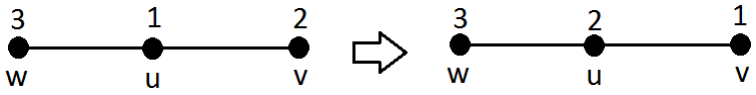


Figura 6.1: Exemplo arestas críticas.

Observe que, no grafo acima, temos $C(u) = C(w) = 1$ e $C(v) = 0$ e largura de banda $B_f(G) = 2$.

Sugerimos a troca dos rótulos entre u e v, após a troca, $C'(u) = 1$ $C'(v) = 1$, largura de banda é reduzida de 2 para 1.

$$\begin{cases} C'(u) = 1 \leq C(u) = 1 & \text{(I)} \\ C'(v) = 1 \leq C(v) = 0 & \text{(II)} \\ C'(u) + C'(v) = 1 + 1 < C(u) + C(v) = 1 + 0 & \text{(III)} \end{cases}$$

Desta forma conseguimos reduzir a banda mesmo quando duas desigualdades do teorema 6.1 são violadas.

Embora o contra-exemplo apresentado seja bem simples e um caso particular, mostra-se que é possível realizar uma troca onde o número de arestas críticas aumenta e, mesmo assim, a largura de banda diminui.

Nesta tese, propõe-se a correção do Algoritmo *Improved Hill Climbing*, sem prejuízo da complexidade, pela substituição do teste expresso na **linha 6** pela seguinte desigualdade

$$[(|f(v) - f_{\min}(u)| < B_f(G)) \wedge (|f(v) - f_{\max}(u)| < B_f(G))] \tag{6.6}$$

Esta desigualdade baseia-se no fato de que a redução de banda de um vértice crítico v sem aumento da banda total só ocorre quando u pertence ao conjunto $N(v, f)$ e, simultaneamente, temos a distância entre $f(v)$ e os vizinhos extremos de u menor que a banda corrente. Para que a manutenção da complexidade, basta utilizarmos uma lista dos vizinhos extremos de cada vértice. Esta lista só é atualizada quando as trocas são efetuadas e não tem relação com a efetividade da troca na melhora da solução.

Principais características da nova busca local

- Não prejudica a eficiência do algoritmo;
- A condição imposta apresenta uma busca local mais completa;

- Como $u \in N(v, f)$, da perspectiva de v, sempre vale a pena trocar v por u, uma vez que, a troca reduz a banda de v;
- A condição também garante que após a troca, a banda de u pode até piorar, porém, sua banda nunca será pior $B_f(G)$.

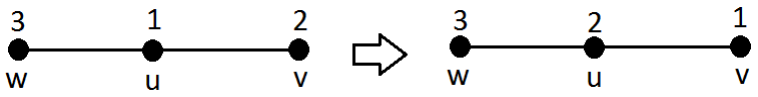
Sendo assim, a nossa proposta de nova busca local tem a seguinte forma:

Algoritmo 10: Nova busca Local

```
1  while Can_Improve do
2      Can_Improve = false
3      for v = 1 to |V| do
4          if C(u) = 1 then
5              forall u such that u ∈ N(v, f) do
6                  if [(|f(v) − f_min(u)| < B_f(G)) ∧ (|f(v) − f_max(u)| < B_f(G))]
                    then
7                      swap(f(v), f(u))
8                      Can_Improve = true
9                      break
10                 end
11             end
12         end
13     end
14 end
```

Verificando desigualdade proposta em nosso contra-exemplo:

$$[(|f(v) - f_{\min}(u)| < B_f(G)) \wedge (|f(v) - f_{\max}(u)| < B_f(G))]$$



Como $f(v) = 2$, $f_{\min}(u) = 2$, $f_{\max}(u) = 3$ e $B_f(G) = 2$, temos

$$[(|2 - 2| < 2) \wedge (|2 - 3| < 2)]$$
$$[(0 < 2) \wedge (1 < 2)]$$

Desta forma, pode-se observar que, a nova desigualdade proposta em 6.6 aceita a troca efetuada no contra-exemplo.

6.3 Perturbação

Na heurística VNS, a perturbação consiste em realizar uma troca das numerações dos vértices. Essa etapa é dividida em três procedimentos.

O primeiro processo de perturbação, mostrado na subseção 6.3.1, é um método que seleciona previamente alguns vértices que possuem as maiores larguras de banda e trocam-se as suas respectivas numerações. No segundo processo de perturbação, mostrado na subseção 6.3.2, é realizada a rotação das numerações de alguns vértices.

Finalmente, a terceira parte, mostrada na subseção 6.3.3, é um procedimento que dependendo das circunstâncias, seleciona qual dos dois procedimentos anteriores será utilizado para realizar a perturbação.

Para melhor entendimento desta etapa, é necessário compreender o conceito de diferença (ou distância) entre as soluções. A distância k entre duas soluções f e f' é dada pelo número de vértices com numeração diferente entre f e f' e decrementa-se um dessa diferença. Uma solução f pertence à vizinhança $N_k(f)$ se a solução f' difere de f em $k+1$ numerações, ou seja, $\rho(f, f') = k \Leftrightarrow f' \in N_k(f)$, em que ρ é a distância entre f e f'. A distancia ρ pode ser definida como uma distância de Hamming [75]. Essa distância é o número de posições em que duas *strings* de mesmo comprimento diferem entre si. A distância ρ entre as soluções f e f' é definida por:

$$\rho(f, f') = \left(\sum_{i=1}^{|V|} \eta(i) \right) - 1 \tag{6.7}$$

em que

$$\eta(i) = \begin{cases} 1, & \text{se } f(i) \neq f'(i) \\ 0, & \text{caso contrário} \end{cases} \tag{6.8}$$

e $f'(i)$ corresponde ao rótulo do vértice i.

6.3.1 Primeiro processo da perturbação

No primeiro processo da perturbação, inicialmente, seleciona-se um conjunto de vértices $Y \subseteq V$ com larguras de banda maior que um valor B', em que $Y = \{v : B_f(v) \geq B'\}$ e $|Y| \geq k$ para que suas numerações sejam trocadas. Desse modo, selecionamos os vértices com as maiores larguras de banda, em seguida, seleciona-se um vértice aleatório $u \in Y$ e seu vértice crítico v.

Nosso próximo passo é encontrar um vértice w, tal que $f_{min}(u) \leq f(w) \leq f_{max}(u)$. Isso significa que a numeração do vértice w não pode aumentar $B_f(u)$.

Além disso, é preciso analisar também as adjacências de w de forma que ao colocar v no lugar de w não altere $B_f(v)$ na nova posição de v. Para isso, verifica-se o valor de $\max\{|f(v) - f_{min}(w)|, |f_{max}(w) - f(v)|\}$. Desta forma, pretendemos atribuir a v uma nova numeração de forma que $B_f(v)$ seja a menor possível.

O Algoritmo 11 apresenta a primeira sub-rotina de perturbação.

Algoritmo 11: Perturbação-1

Input: grafo $G = (V, E)$; raio da vizinhança k; rotulagem f; limiar B'
Output: nova rotulagem f'

```
1  início
2      Let Y = {v|B_f(v) ≥ B'} where B' is selected such that |Y| ≥ k.
3      for i= 1 to k do
4          u ← RadomInt (1, |Y|);
5          v ← such that |f(v) − f(u)| = B_f(u);
6          if (u, v) ∈ E then
7              w ← argmin_w{max{f_max(w) − f(v), f(v) − f_min(w)}|f_min(u) <
                 f(w) < f_max(u)};
8              swap(f(v), f(w))
9          end
10     end
11     return f';
12 fim
```

O Algoritmo recebe o grafo $G = (V, E)$, o raio k da solução corrente, a numeração f corrente e o limiar B'. O valor de B', passado para o Algoritmo 11, é de forma que $|Y| \geq k$. Na estrutura de repetição **for** nas linhas 3 a 9, seleciona-se o vértice $u \in Y$, o vértice crítico v de u e o vértice w para trocar com v. Repete-se esse processo k vezes. A sub-rotina retorna a renumeração $f' \in N_k(f)$.

6.3.2 Segundo processo da perturbação

De acordo com MLADENOVIC *et al.* [72], apenas o Algoritmo 11 não é suficiente para escapar de um mínimo local em algumas situações. Para contornar esse problema utilizamos uma segunda sub-rotina de perturbação, proposta por RODRIGUEZ-TELLO *et al.* [76] que é baseada na rotação entre duas vizinhanças. Na sub-rotina de rotação de vizinhanças, na etapa de perturbação, troca-se a numeração de todos os vértices que pertencem a um intervalo preestabelecido.

Inicialmente, para delimitar o intervalo de troca, escolhem-se dois índices b e e com um terceiro índice m entre eles, ou seja, $1 \leq b < m < e \leq |V|$. Trocam-se as numerações de todos os vértices no intervalo $[b, e]$, deslocando-os m posições, ou

seja, a numeração $f(b)$ passará a ser a posição $f(b+m)$, se $b+m \leq e$. Para um índice i, em que em $i+m > e$, a numeração $f(i)$ será deslocada para a posição $f(b+|e-(i+m)|)$.

A sub-rotina de rotação é mostrada no Algoritmo 12.

Algoritmo 12: Perturbação-2

Input: grafo $G=(V,E)$; raio da vizinhança k; rotulagem f;
Output: nova rotulagem f';

```
1  início
2      f' ← f;
3      for (i ← 1; i ≤ k; i ← i + 1) do
4          b ← RadomInt(1, |V|);
5          e ← b + RadomInt(2, min(20, B_f(G)/2));
6          if (e > |V|) then
7              e ← |V|;
8          end
9          m ← RadomInt(b + 1, e − 1);
10         for (j ← b; j ≤ e; j ← j + 1) do
11             if (j ≥ b + m) then
12                 f'(j) ← j − m;
13             else
14                 f'(j) ← j + e − b − m + 1;
15             end
16         end
17     end
18     return f';
19 fim
```

O Algoritmo recebe um grafo $G=(V,E)$, o raio k da solução corrente e a rotulagem f corrente. Dentro da estrutura de repetição para nas linhas 3 a 17, primeiramente, escolhem-se, aleatoriamente, os índices b e e do intervalo de troca. Estabelecem-se, aleatoriamente, os índices b e e, em que, na função min, utilizam-se os parâmetros 20 e $B_f(G)/2$. Esses parâmetros foram sugeridos por [72], pois gerou bons resultados nos testes realizados, embora eles possam ser adaptados de acordo com as características das novas instâncias testadas. Na linha 9, estabelece-se m, também aleatório, no intervalo $[b,e]$. Na estrutura de repetição para nas linhas 10 a 15, realiza-se a rotação no intervalo $[b,e]$. A sub-rotina retorna a numeração $f' \in N_k(f)$.

6.3.3 Escolha do processo da perturbação

A partir desse momento é necessário uma sub-rotina de escolha para decidir qual sub-rotina de perturbação devemos utilizar. O Algoritmo 13 serve para definir quando utilizar o Algoritmo de perturbação 11 ou 12. No Algoritmo 13, o valor k'_{max} é definido de acordo com a necessidade de perturbação para se escapar de um vale.

Se $k \leq k'_{max}$, utiliza-se o Algoritmo 11; caso contrário, utiliza-se o algoritmo 12.

Segundo [72], verificou-se que o valor de k, quando passado ao algoritmo 12, gerava perturbações desnecessárias, pois necessita-se de $k = \lfloor \frac{k-k_{min}}{k_{step}} \rfloor$, em que k_{min} é o raio inicial para a busca na vizinhança e k_{step} é o incremento do raio a cada passo. Nesta proposta, quando não consegue escapar de um mínimo local, o raio da vizinhança $N_k(f)$ é incrementado por k_{step}, possibilitando encontrar novas soluções.

Algoritmo 13: Perturbação

Input: grafo $G = (V, E)$; raio da vizinhança k; raio inicial k_{min}; raio de incremento k_{step}; rotulagem f; raio máximo k'_{max}; limiar B'

Output: nova rotulagem f';

```
1 início
2 |   if (k ≤ k'_max) then
3 |   |   f' ← Perturbacao1(G, k, f, B');
4 |   else
5 |   |   f' ← Perturbacao2(G, ⌊(k − k_min)/k_step⌋, f);
6 |   end
7 |   return f';
8 fim
```

O Algoritmo recebe o raio k da solução corrente, k_{min} como o raio inicial, k_{step} como o incremento para aumentar o raio de busca, k'_{max} que é o raio máximo, a numeração f corrente e o limiar B'.

6.4 Mudança de vizinhança

A mudança de vizinhança é uma etapa muito importante do VNS. Definir estratégias eficientes de escolha entre as vizinhanças podem acelerar o processo de convergência do método HANSEN e MLADENOVIĆ [77]. Para executar esse movimento, precisamos responder as seguintes questões:

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

(i) O mais simples é baseado nos valores da função objetivo. Verifica-se qual solução possui a menor largura de banda. Se $B_f(G) > B_{f'}(G)$, então, f' passa a ser a solução corrente, ou seja, $f \leftarrow f'$. Caso contrário, f continua como a solução corrente.

(ii) Se $B_f(G) = B_{f'}(G)$, então, avalia-se o número de vértices críticos de cada solução. Se $|V_C(f)| > |V_C(f')|$, então, f' passa a ser a solução corrente. Caso contrário, f continua como a solução corrente, em que $|V_C(f)|$ é o conjunto de vértices críticos da solução f.

(iii) Se $B_f(G) = B_{f'}(G))$ e $|V_C(f)| = |V_C(f')|$, nesse caso, avalia-se a diferença (distância) entre as soluções f e f'. Troca-se a solução f por f' se $\rho(f, f') > \alpha$, em que α é um parâmetro constante que, segundo [72] nos testes preliminares, gerou os melhores resultados para $\alpha = 10$. Esse terceiro critério baseia-se em uma variação da metaheurística VNS proposta por [44]. A ideia é aplicar um movimento que aceita uma solução pior do que a solução corrente, se o ótimo estiver muito longe dela. Essa técnica também é conhecida como SVNS (ou VNS inclinada). Porém, para o problema de redução de largura de banda, se f e f' possuírem a mesma largura de banda e o mesmo número de vértices críticos, permuta-se de f para f' apenas se $\rho(f, f') > \alpha$. A grande vantagem dessa troca é a possibilidade de se explorar novas vizinhanças que ainda não foram visitadas.

2. Qual a próxima vizinhança: A segunda questão é descobrir qual será a nova vizinhança no caso de ocorrer a permutação da solução corrente e como alterar a vizinhança corrente quando não houver permutação.

   Utilizam-se parâmetros k_{min}, k_{step} e k_{max} para definir se o raio de busca na vizinhança será incrementado, se voltará ao raio inicial ou se excedeu o limite. Temos que k_{min} é o raio inicial para a busca na vizinhança. Se a solução corrente f é trocada para f', então, atualiza-se o raio da busca de $N_k(f')$ para k_{min}, ou seja, utiliza-se $N_{k_{min}}(f')$.

   Se após a perturbação e a busca local não houver melhorias em relação à solução corrente f, então, estabelece-se $k \leftarrow k_{step}$. Com isso, se $k < k_{max}$, repetem-se os passos de busca. Lembrando que k_{max} é o raio máximo em que é realizada a busca.

Algoritmo 14: Troca

Input: rotulagem f; rotulagem f'; parâmetro α;

Output: $true$ ou $false$;

```
1 início
2 |  //permuta-se ou não a solução corrente
3 |  move ← false
4 |  if (B_f(G) > B_f'(G))  or
5 |  (B_f(G) = B_f'(G) and |V_C(f)| > |V_C(f')|) or
6 |  (B_f(G) = B_f'(G) and |V_C(f)| = |V_C(f')| and ρ(f, f') > α) then
7 |  |  move ← true
8 |  end
9 fim
```

Mostra-se no Algoritmo 14 a sub-rotina que verifica se haverá ou não a permutação da solução corrente f pela solução f'. A sub-rotina recebe a solução corrente f, a solução f' e o parâmetro $\alpha = 10$. Verificamos se é viável realizar a permutação da solução corrente f pela solução f' ou não. Se for viável, o retorno do Algoritmo é verdadeiro; em caso contrário, o Algoritmo retorna falso.

6.5 Formulação Metaheurística VNS

A metaheurística VNS mostrada no Algoritmo 15 utilizou o tempo de execução como critério de parada. O algoritmo recebe o raio k da vizinhança, raio inicial k_{min}, o incremento k_{step} do raio, o raio máximo k_{max}, o raio pré-definido k'_{max} , o limiar B', a lista de adjacências E, o parâmetro α, a numeração corrente f e o tempo máximo t_{max}.

Nesta tese, em todas as instâncias, foram utilizados os seguintes valores nos parâmetros: $k_{min} = k_{step} = 5$; $k'_{max} = 25$; $k_{max} = 50$ $\alpha = 10$ e $t_{max} = 60s$.

O Algoritmo 15 mostra o funcionamento da metaheurística VNS.

Algoritmo 15: Heurística VNS

Input: grafo $G = (V, E)$; raio da vizinhança k; raio inicial k_{min}; raio de incremento k_{step}; raio máximo k_{max}; raio pre-definido k'_{max}; limiar B'; parâmetro α; rotulagem f; tempo máximo t_{max};

Output: rotulagem final f^*;

```
1  início
2      B_min ← +∞;
3      t ← 0;
4      i_max ← ⌊(k_max − k_min)/k_step⌋;
5      while (t < t_max) do
6          f ← SolucaoInicial(f);
7          f ← BuscaLocal(f);
8          f* ← f;
9          i ← 0;
10         k ← k_min;
11         while (i ≤ i_max) do
12             f' ← Perturbacao(G, k, k_min, k_step, k'_max, f, B');
13             f'' ← BuscaLocal(f')
14             if (Troca(f, f', α) = true) then
15                 f ← f'';
16                 k ← k_min;
17                 i ← 0;
18             else
19                 k ← k + k_step;
20                 i ← i + 1;
21             end
22         end
23         if (B_f(G) < B_min) then
24             B_min ← B_f(G);
25             f* ← f;
26         end
27         t ← TempoCPu()
28     end
29     return f*;
30 fim
```

Nas linhas 2 a 4, as variáveis B_{min}, t e i_{max} são inicializadas, respectivamente. B_{min} é a variável com a menor largura de banda encontrada durante a execução, t é

o tempo inicial e i_{max} recebe $\lfloor (k_{max} - k_{min})/k_{step} \rfloor$. Na linha 6, dentro da estrutura de repetição *while* externa, constrói-se então a solução inicial. Após a construção da solução inicial pelo Algoritmo 7, aplica-se a busca local nessa solução e inicializam-se as variáveis i e k. Na estrutura de repetição *while* interna nas linhas 11 a 21, aplica-se a perturbação na solução corrente f, gerando-se f'. Em seguida, gera-se f'', aplicando-se a busca local em f'. Ainda dentro do *while* interno, na estrutura condicional, verifica-se se haverá troca da solução corrente f pela solução f''. Caso troque, a solução corrente passa a ser f'' e atualiza se o raio k para k_{min}, caso contrário, incrementa-se o raio k com k_{step}. Na estrutura condicional nas linhas 23 a 26, verifica-se se a largura de banda da solução corrente é menor do que a menor largura de banda encontrada até o momento. Caso seja, guarda-se a largura de banda em B_{min} e a numeração correspondente em f^*. A sub-rotina $TempoCPU$ na linha 27 retorna o tempo decorrido. O Algoritmo para quando o tempo de processamento excede o limite t_{max}. O Algoritmo retorna a renumeração f^* do grafo.

Capítulo 7

Experimentos computacionais

Nesta seção, apresenta-se os resultados computacionais utilizando o modelo exato e os resultados obtidos pela metaheurística VNS. A metaheurística VNS foi implementada em linguagem Python 3 devido a facilidade de codificação. Esta escolha, no entanto, apresenta como efeito colateral a perda de performance por ser uma linguagem interpretada. Sendo assim, não nos atemos aos tempos de execução.

Compara-se a qualidade dos nossos resultados com os da metaheurística clássica denominada Reverse Cuthill-McKee (RCM) [19]. Esse método é uma variação do Algoritmo Culthill-Mckee proposto em [18], sendo considerado o método padrão para a solução do problema de redução de largura de banda de matrizes simétricas esparsas devido à sua rapidez e simplicidade.

O modelo exato proposto é de aplicação restrita, dada a complexidade decorrente da otimalidade de suas soluções. Por isso, restringimos nossos experimentos apenas a validação do modelo. Para tanto, consideramos a matriz DWT59 formada por 60 colunas, 267 elementos não-nulos e obtida em `https://sparse.tamu.edu/HB/dwt_59`. A Figura 7.1 mostra a configuração original e os resultados obtidos utilizando o modelo exato e os Algoritmos VNS e RCM. Embora a instância seja bem pequena, o modelo exato apresentou resultados melhores que os resultados obtidos com os métodos heurísticos.

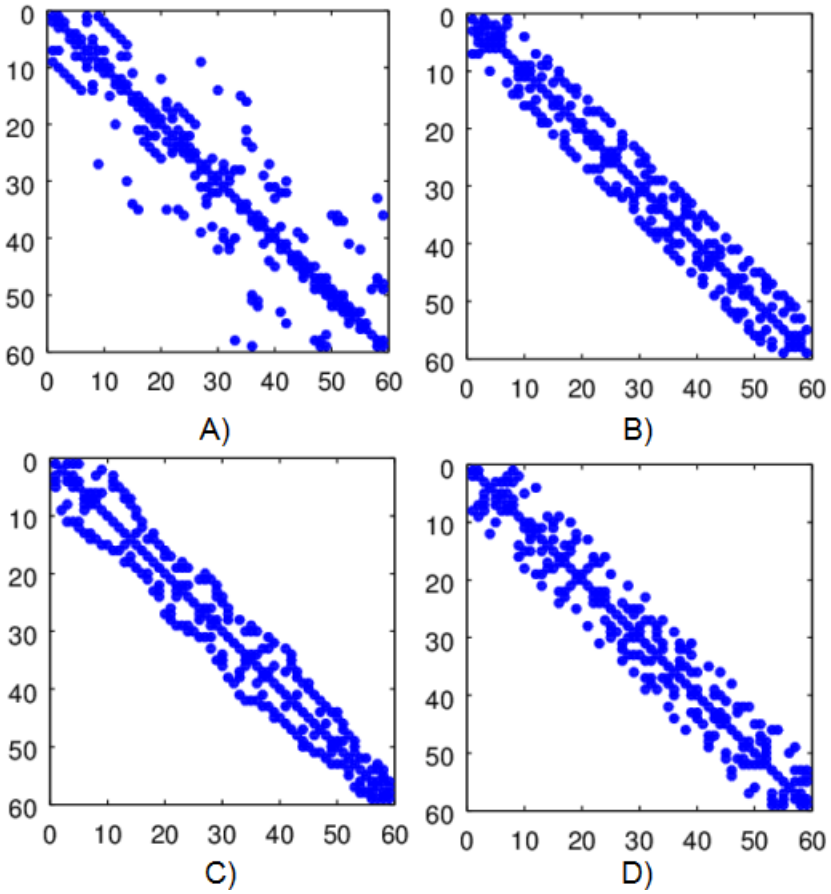


Figura 7.1: A) Banda da Matriz original = 25 B) Banda do Modelo Exato = 6 C) Banda do Método RCM = 8 D) Banda do Método VNS = 8.

As Tabelas 7.1, 7.2 e 7.3 apresentam o número de linhas (n) das matrizes utilizadas, e largura de banda obtida pela proposta VNS e pelo método Reverse Cuthill-McKee (RCM). Apresenta-se uma coluna com os resultados VNS-RCM. Valores positivos demonstram que a proposta VNS foi mais eficiente. Todas as instâncias testadas foram obtidas em `https://sparse.tamu.edu`. A Tabela 7.1 apresenta as matrizes até 10^3 linhas, a Tabela 7.2 apresenta as matrizes com número de linhas entre 10^3 e 10^4 e a Tabela 7.3 as matrizes até 10^6 linhas.

Embora o artigo MLADENOVIC *et al.* [72] adote o tempo máximo de 500 segundos, na revisão sistemática apresentada por DE OLIVEIRA *et al.* [22], o VNS proposto por [72] apresentou performance melhor que outras heurísticas mesmo com tempo máximo limitado a apenas 8 segundos.

Durante a execução do VNS, ocorrem vários restarts, os resultados apresentados em nossos experimentos foram obtidos por uma única execução do VNS proposto, onde limitamos o tempo de execução em 60 segundos e até 20 restarts.

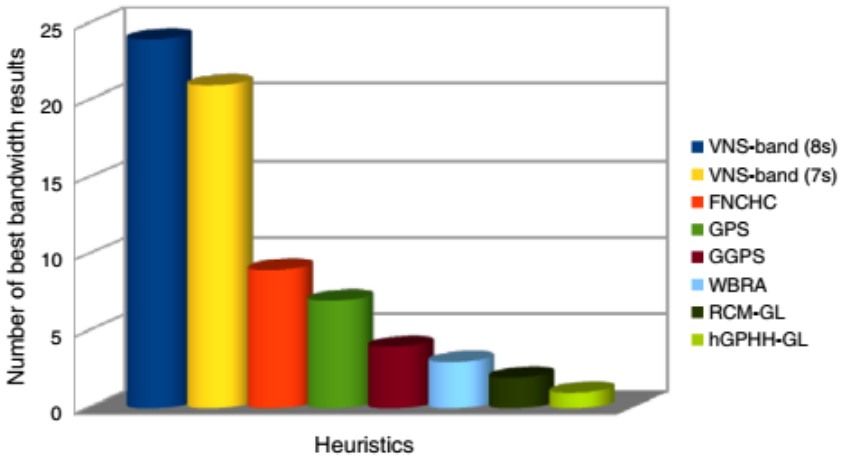


Figura 7.2: VNS versus outras Metaheurísticas.

Fonte: DE OLIVEIRA *et al.* [22]

A Figura 7.2, mostra os principais resultados obtidos pela revisão sistemática apresentada em [22] entre diversas metaheurísticas diferentes utilizadas para a redução de banda de matrizes esparsas. Observe que mesmo com tempos de execução de 8 e 7 segundos, a metaheurística VNS proposta por [72] apresenta resultados superiores em relação as demais metaheurísticas.

Tabela 7.1: Resultados para matrizes até 10^3 linhas.

	Instância	**n**	**Banda RCM**	**Banda VNS**	**RCM-VNS**
1	**494_bus**	494	68	48	20
2	**662_bus**	662	134	64	70
3	**685_bus**	685	82	49	33
4	**bcsstk01**	48	27	16	11
5	**bcsstk02**	66	65	65	0
6	**bcsstk04**	132	54	38	16
7	**bcsstk05**	153	24	20	4
8	**bcsstk06**	420	50	47	3
9	**bcsstk07**	420	50	47	3
10	**bcsstk19**	817	21	23	-2
11	**bcsstk34**	588	124	95	29
12	**bcsstm07**	420	56	47	9
13	**ex5**	27	8	8	0
14	**lund_a**	147	23	23	0
15	**lund_b**	147	23	23	0
16	**mesh1e1**	48	48	12	36
17	**mesh1em1**	48	15	12	3
18	**mesh1em6**	48	15	12	3
19	**mesh2e1**	306	55	31	24
20	**mesh2em5**	289	55	31	24
21	**mesh3e1**	289	19	20	-1
22	**mesh3em5**	726	19	20	-1
23	**nos3**	960	79	43	36
24	**nos4**	100	16	11	5
25	**nos5**	468	131	69	62
26	**nos6**	675	30	23	7
27	**nos7**	729	65	75	-10
28	**plat362**	362	52	39	13

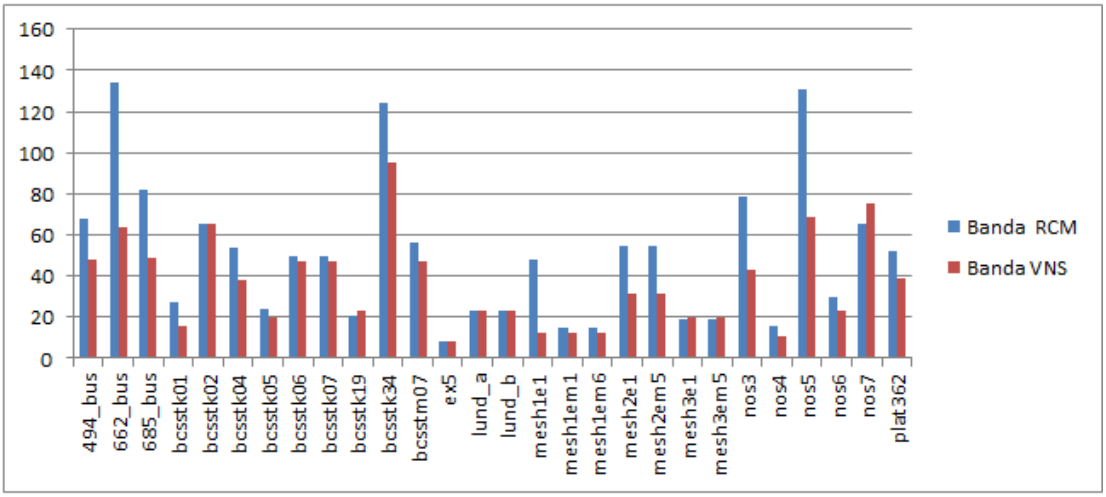


Figura 7.3: Comparação entre larguras de banda RCM e VNS.

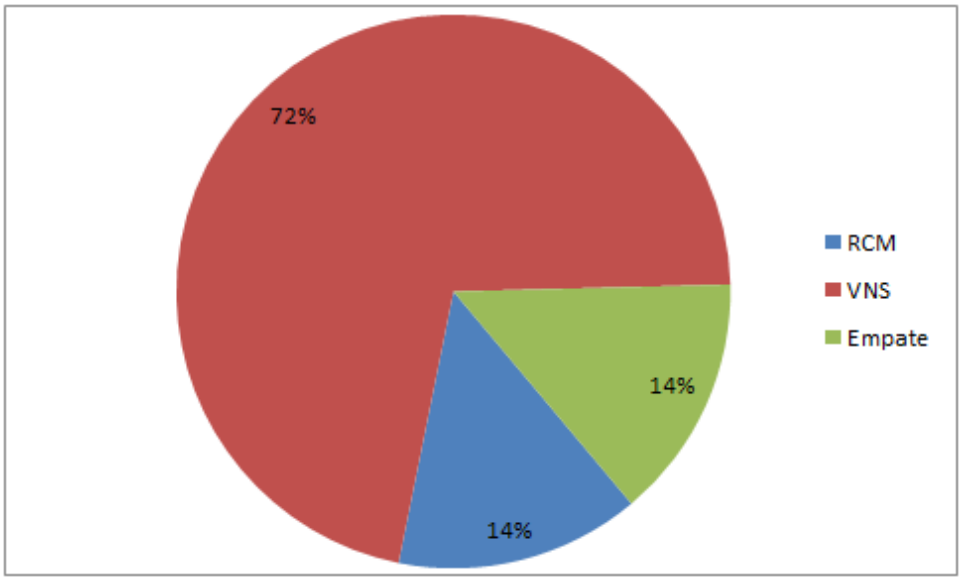


Figura 7.4: Percentuais de eficiência do VNS em relação ao RCM matrizes apresentadas na Tabela 7.1.

A Figura 7.4 mostra que a posposta VNS foi mais eficiente em 72%, empatou em 14% e perdeu em 14% dos casos testados.

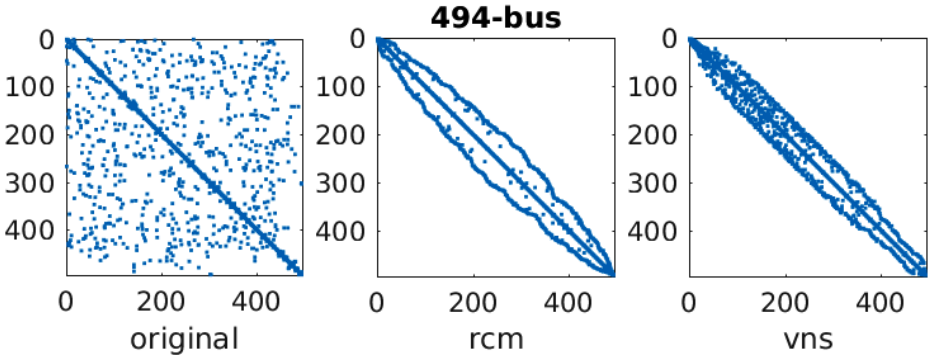


Figura 7.5: Banda Original=428, Banda RCM=68, Banda VNS=48

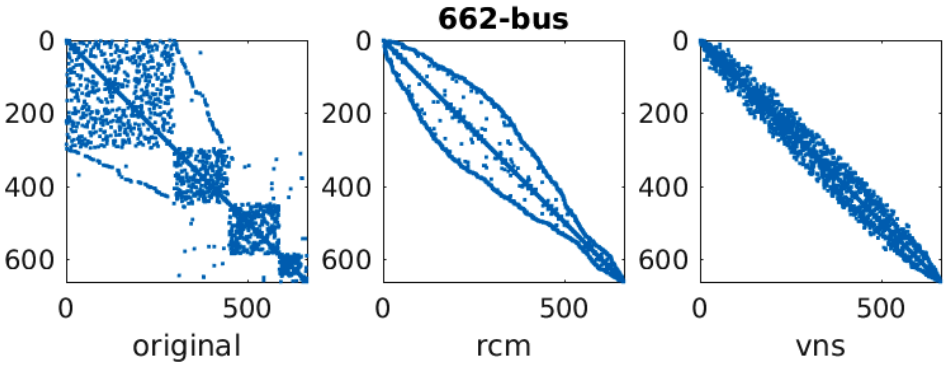


Figura 7.6: Banda Original=335, Banda RCM=134, Banda VNS=64

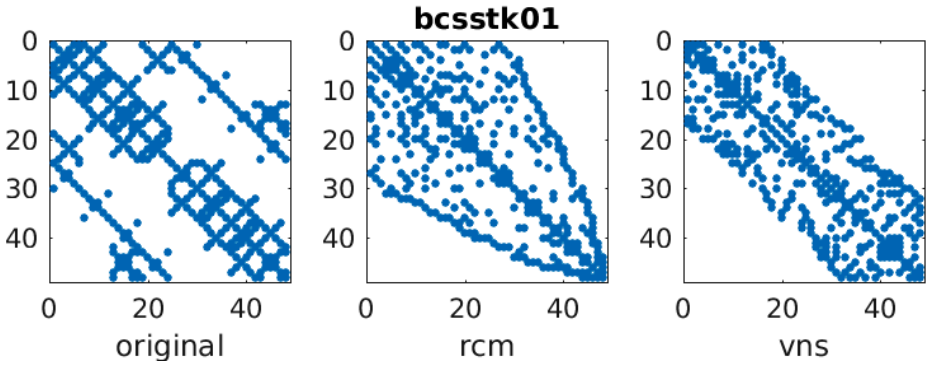


Figura 7.7: Banda Original=35, Banda RCM=27, Banda VNS=16

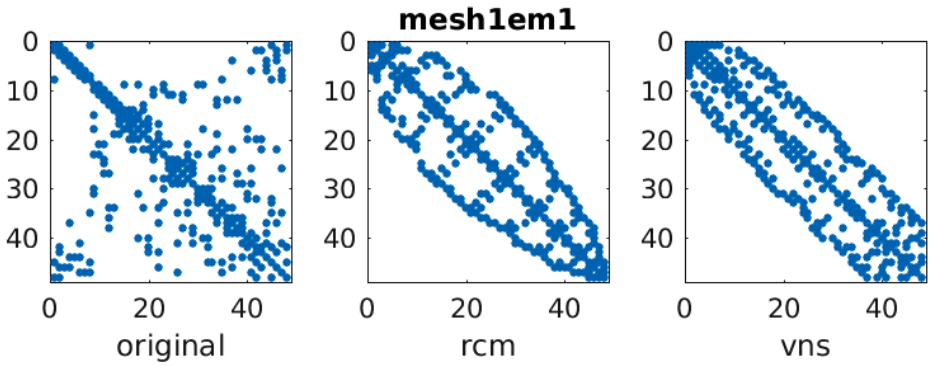


Figura 7.8: Banda Original=47 Banda RCM= 15, Banda VNS=12

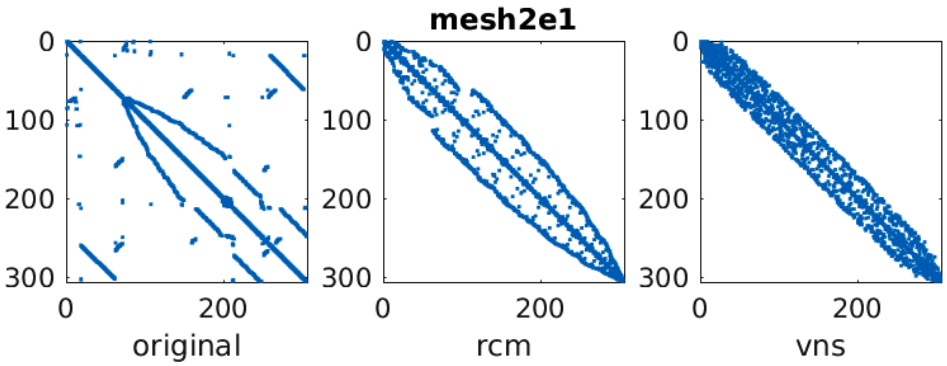


Figura 7.9: Banda Original=285, Banda RCM=55, Banda VNS=31

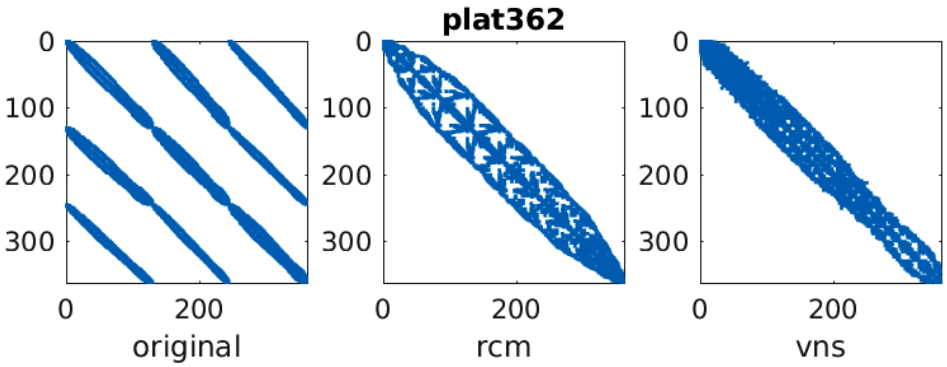


Figura 7.10: Banda Original=249, Banda RCM=52, Banda VNS=39

As Figuras 7.5, 7.6,7.7, 7.8, 7.9, 7.10 apresentam os nomes de algumas matrizes testadas na Tabela 7.1, sua largura de banda original e larguras de banda reduzidas obtidas pelos métodos RCM e VNS.

Tabela 7.2: Resultados para matrizes até 10^4 linhas.

	Instância	n	Banda RCM	Banda VNS	RCM-VNS
1	**1138_bus**	1138	148	96	52
2	**aft01**	8205	247	154	93
3	**bcsstk09**	1083	113	62	51
4	**bcsstk13**	2003	562	315	247
5	**bcsstk23**	3134	391	382	9
6	**bcsstk24**	3562	305	260	45
7	**bcsstk26**	1922	242	180	62
8	**bcsstk27**	1244	66	48	18
9	**bcsstk28**	4410	494	320	174
10	**bcsstk38**	8032	609	608	1
11	**dwt_2680**	2680	131	103	28
12	**ex13**	2568	223	133	90
13	**ex3**	1821	181	107	74
14	**ex9**	3363	147	85	62
15	**fv1**	9604	195	99	96
16	**fv2**	9801	197	100	97
17	**fv3**	9801	197	100	97
18	**Kuu**	7102	673	664	9
19	**msc00726**	1050	350	188	162
20	**msc01050**	1824	414	227	187
21	**msc01440**	1440	162	109	53
22	**nasa1824**	2910	263	172	91
23	**nasa2146**	2146	148	81	67
24	**nasa2910**	2910	871	489	382
25	**nasa4704**	4704	419	423	-4
26	**plat1919**	1919	80	94	-14
27	**s1rmq4m1**	5489	359	191	168
28	**s1rmt3m1**	5489	186	191	-5
29	**s2rmq4m1**	5489	359	191	168
30	**s2rmt3m1**	5489	186	191	-5
31	**s3rmq4m1**	5489	359	191	168
32	**s3rmt3m1**	5489	186	191	-5
33	**s3rmt3m3**	5357	212	220	-8
34	**sts4098**	4098	1365	1262	103

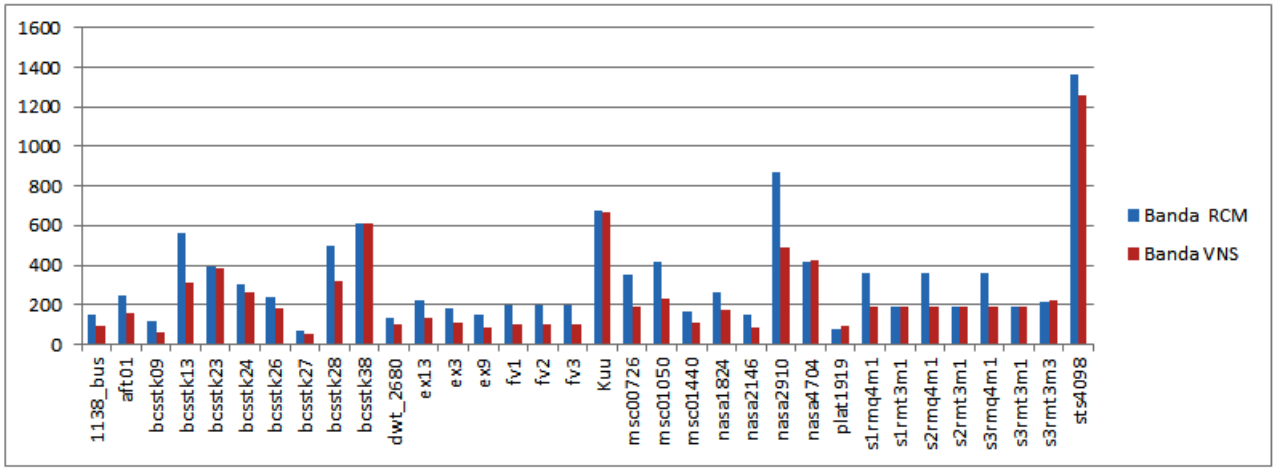


Figura 7.11: Comparação entre larguras de banda RCM e VNS.

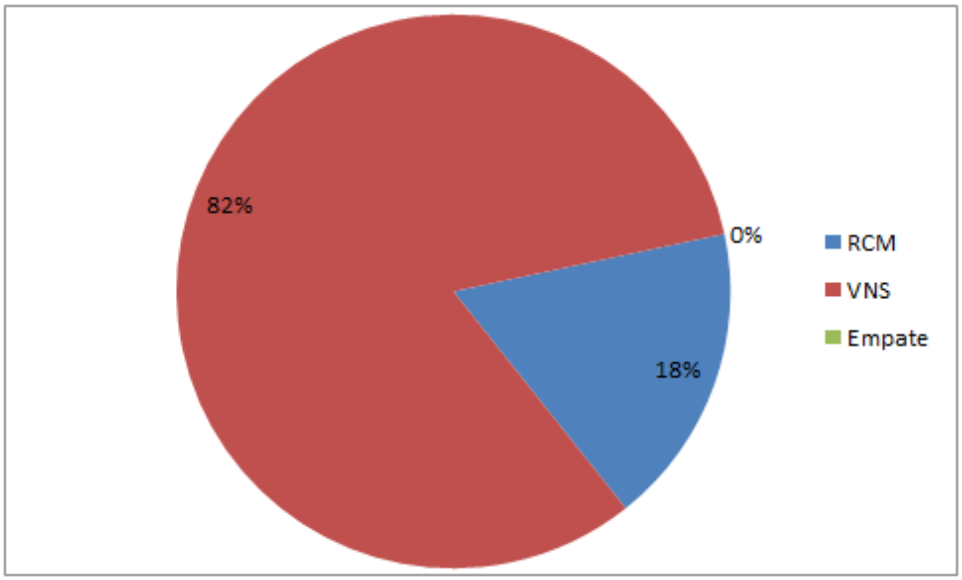


Figura 7.12: Percentuais de eficiência do VNS em relação ao RCM matrizes apresentadas na Tabela 7.2.

A Figura 7.12 mostra que a posposta VNS foi mais eficiente em 82%, empatou em 0% e perdeu em 18% dos casos testados.

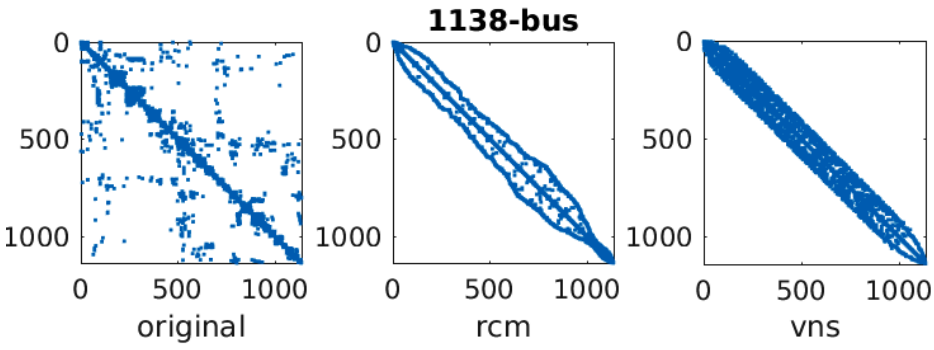


Figura 7.13: Banda Original=1030, Banda RCM=148, Banda VNS=96

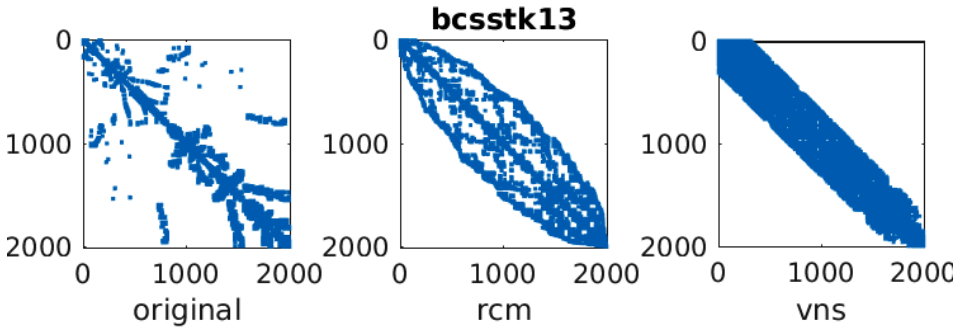


Figura 7.14: Banda Original=1250, Banda RCM=562, Banda VNS=315

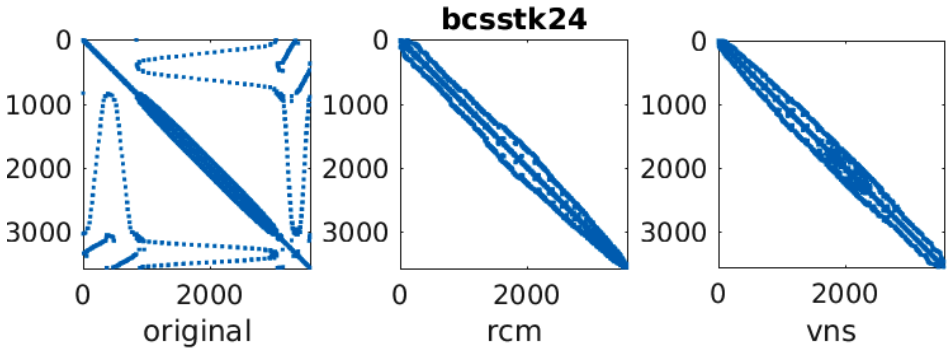


Figura 7.15: Banda Original=3333, Banda RCM=305, Banda VNS=260

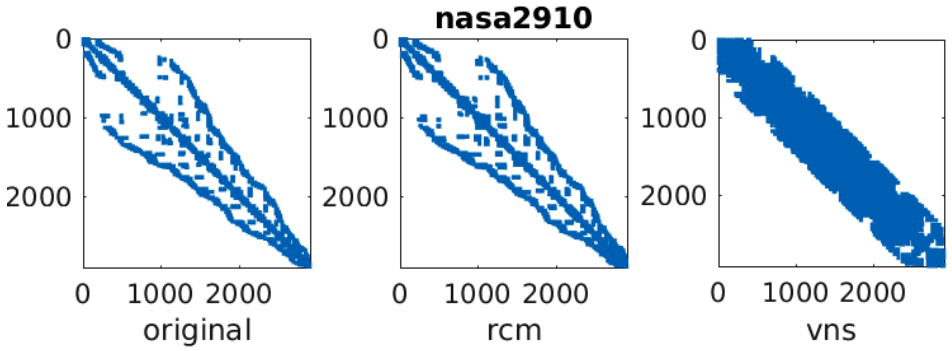


Figura 7.16: Banda Original=859, Banda RCM=871, Banda VNS=489

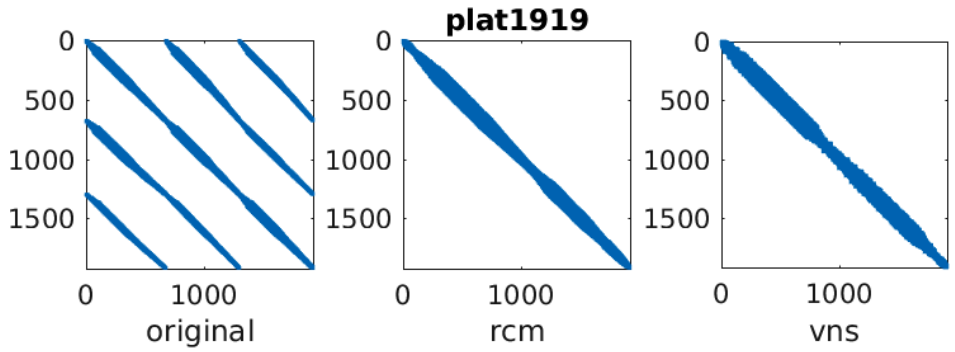


Figura 7.17: Banda Original=1297, Banda RCM=80, Banda VNS=94

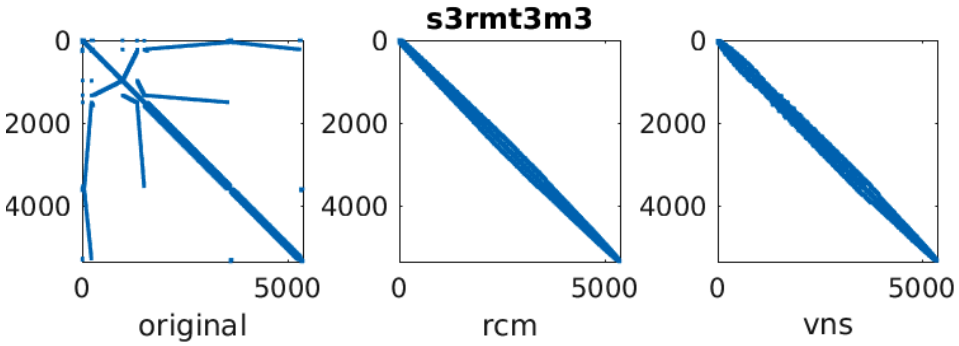


Figura 7.18: Banda Original=5302, Banda RCM=212, Banda VNS=220

As Figuras 7.13, 7.14,7.15, 7.16, 7.17, 7.18 apresentam os nomes de algumas matrizes testadas na Tabela 7.2, sua largura de banda original e larguras de banda reduzidas obtidas pelos métodos RCM e VNS.

Tabela 7.3: Resultados para matrizes até 10^6 linhas.

	Instância	n	Banda RCM	Banda VNS	RCM-VNS
1	**bundle1**	10581	5426	4946	480
2	**bodyy4**	17546	402	253	149
3	**bodyy6**	19406	432	367	65
4	**denormal**	89400	597	596	1
5	**Dubcova1**	16524	500	440	60
6	**Dubcova2**	65025	1012	954	58
7	**ecology2**	999999	1000	1000	0
8	**finan512**	74752	1317	1266	51
9	**G2_circuit**	150102	2149	2080	69
10	**gridgena**	48962	670	405	265
11	**gyro**	17361	1703	1527	176
12	**gyro_k**	17361	1703	1527	176
13	**jnlbrng1**	40000	200	200	0
14	**msc10848**	10848	2168	1685	483
15	**msc23052**	23052	1526	1556	-30
16	**olafu**	16146	533	584	-51
17	**qa8fm**	66127	1968	1048	920
18	**raefsky4**	19779	1052	1228	-176
19	**s3dkt3m2**	90449	608	614	-6
20	**shallow_water1**	81920	322	351	-29
21	**shallow_water2**	81920	322	351	-29
22	**t2dah_e**	11445	232	214	18
23	**wathen100**	30401	601	304	297
24	**wathen120**	36441	601	304	297

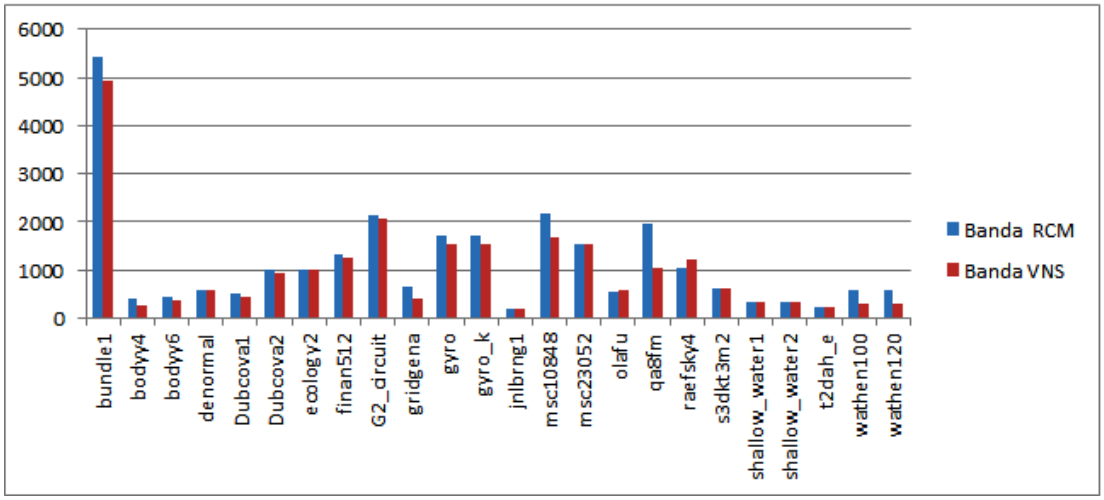


Figura 7.19: Comparação entre larguras de banda RCM e VNS.

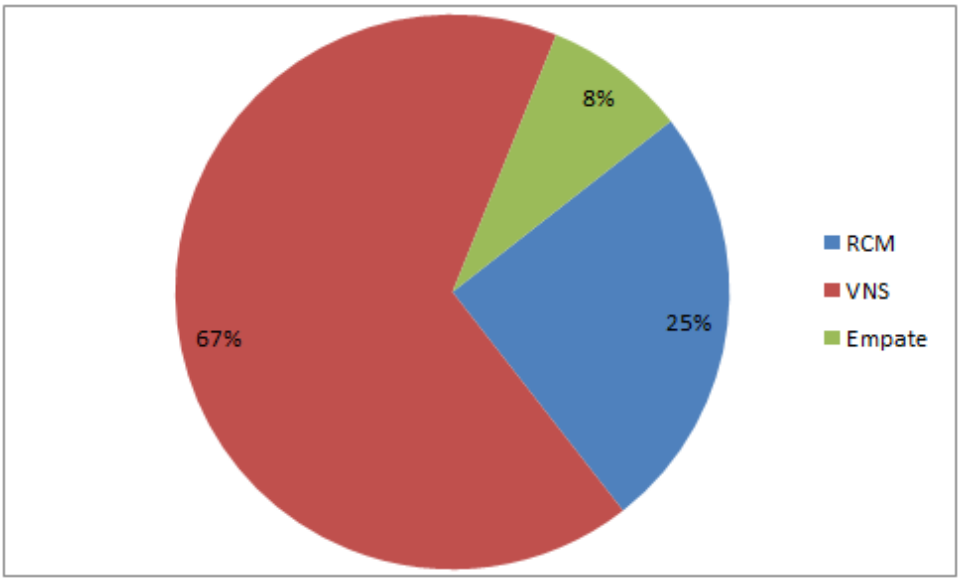


Figura 7.20: Percentuais de eficiência do VNS em relação ao RCM matrizes apresentadas na Tabela 7.3.

A Figura 7.20 mostra que a posposta VNS foi mais eficiente em 67%, empatou em 8% e perdeu em 25% dos casos testados.

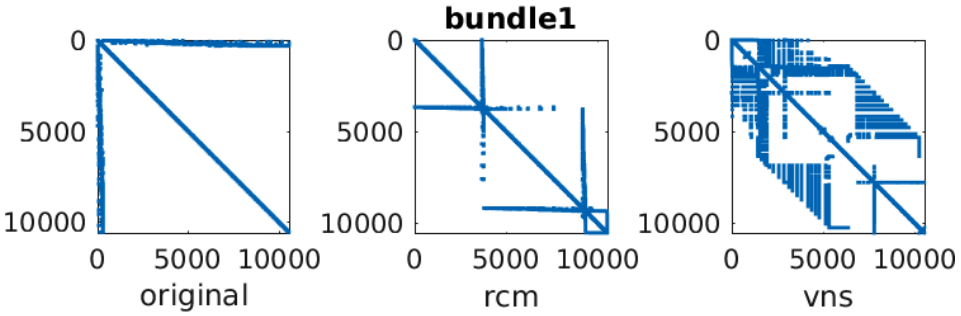


Figura 7.21: Banda Original=10461, Banda RCM=5426, Banda VNS=4946

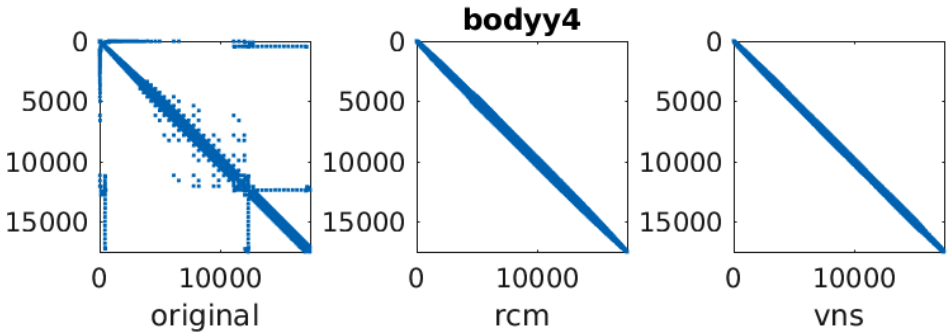


Figura 7.22: Banda Original=16818, Banda RCM=402, Banda VNS=253

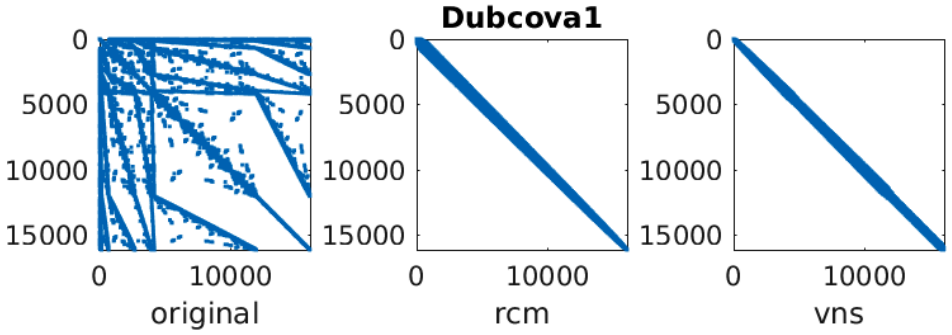


Figura 7.23: Banda Original= 16052, Banda RCM=500, Banda VNS=440

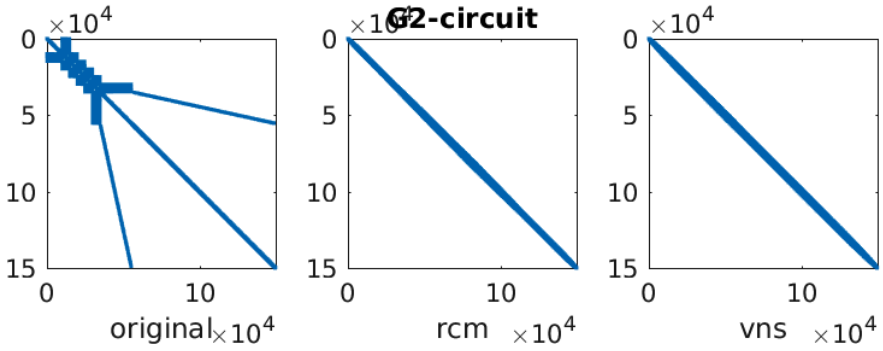


Figura 7.24: Banda Original= 93719, Banda RCM=2149, Banda VNS=2080

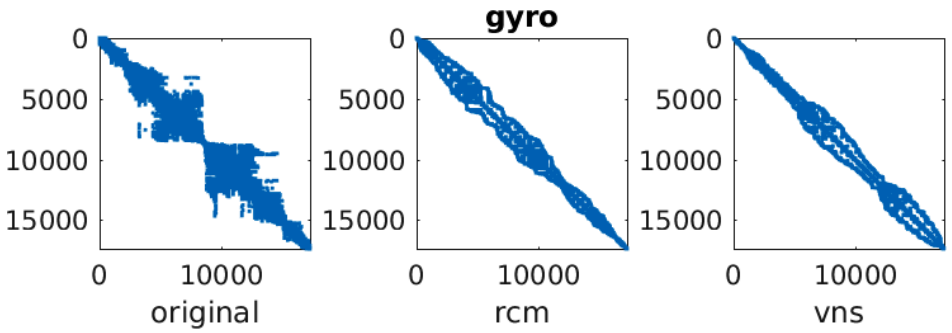


Figura 7.25: Banda Original= 5144, Banda RCM=1703, Banda VNS=1527

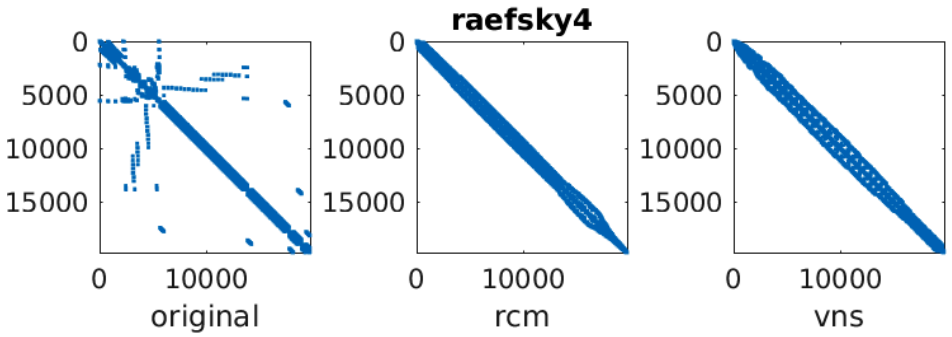


Figura 7.26: Banda Original= 11786, Banda RCM=1052, Banda VNS=1228

As Figuras 7.21,7.22,7.23,7.24,7.25,7.26, apresentam os nomes de algumas matrizes testadas na Tabela 7.3, sua largura de banda original e larguras de banda reduzidas obtidas pelos métodos RCM e VNS.

Capítulo 8

Conclusões e Sugestões de Trabalhos Futuros

Nesta tese, são apresentadas duas alternativas para resolver o problema de redução de banda (PRB). Uma destas propostas foi um modelo exato de programação inteira e a outra alternativa foi uma modificação pontual na etapa de busca local da metaheurística VNS apresentada em [72]. Esta modificação incorpora uma correção do algoritmo *Improved Hill-Climbing* proposto em [73].

O modelo exato, embora encontre a solução ótima, tem aplicação restrita a matrizes de pequeno porte. Uma possível utilidade prática deste modelo é a geração de limites para a qualificação dos resultados utilizando metaheurísticas. Como trabalho futuro, podemos expandir a aplicação do modelo exato incorporando novos limitantes ou considerando versões relaxadas para a geração de soluções iniciais em métodos híbridos como aplicado em [78].

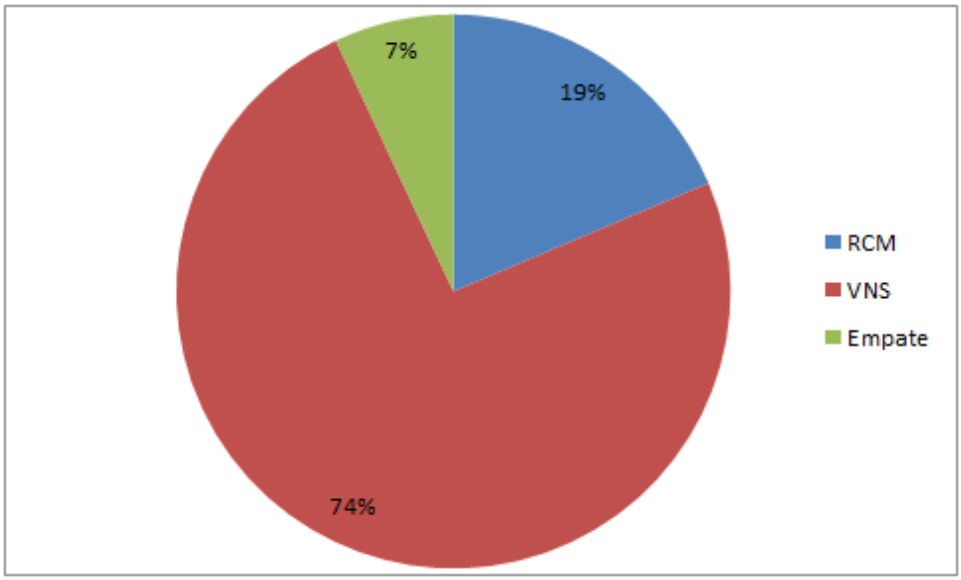


Figura 8.1: Percentuais gerais de eficiência do VNS em relação ao RCM.

O método VNS proposto mostrou-se computacionalmente mais caro que a heurística padrão RCM nos testes preliminares, contudo a qualidade das soluções obtidas foi melhor em 74% dos testes realizados. Por simplicidade, sua versão preliminar foi implementada em linguagem Python e como trabalho futuro desejamos implementar versões em linguagem C para verificar a viabilidade de utilização desta proposta em aplicações reais. Outra possível linha a ser abordada em trabalhos futuros é a combinação de diferentes metaheurísticas.

Das 86 instancias testadas o VNS apresentou resultados melhores que o RCM em 64 instâncias, o VNS apresentou resultados iguais ao RCM em 6 instâncias, e apresentou resultados piores que o RCM em 16 instâncias.

Em termos de eficiência, o nosso Algoritmo apresenta a mesma complexidade dos propostos em [72], [73] e [74] que utilizam o teorema 6.1. No entanto, corrigimos o teorema 6.1 e introduzindo a condição 6.6, apresentando assim uma proposta de busca local mais correta sem prejudicar a eficiência do algoritmo.

Com base no que foi apresentando, propõe-se como trabalhos futuros para o Modelo Exato:

Estudar o poliedro do problema para encontrar cortes válidos e desenvolver uma algoritmo de Branch-and-cut baseado em CAPRARA *et al.* [79] e NEMHAUSER e WOLSEY [80].

Encontrar limites inferiores tendo com base o trabalho proposto por CAPRARA e SALAZAR-GONZÁLEZ [25], sendo esta uma formulação considerada clássica.

Buscar novas desigualdades válidas para as formulações matemáticas propostas com base nas desigualdades válidas sugeridas nos trabalhos SEITZ [81] e FERREIRA [82].

Com base no que foi apresentando, propõe-se como trabalhos futuros para o Método Metaheurístico VNS:

Desenvolver novas vizinhanças válidas.

Construir uma metaheurística hibrida combinando VNS com outras metaheurísticas clássicas como GRASP, *Simulated Annealing*, *Tabu Search* e Algoritmo Genético.

Construir uma metaheurística hibrida combinando VNS com modelos exatos.

Desenvolver uma metaheurística VNS apresentada utilizando paralelismo.

Referências Bibliográficas

[1] KAVEH, A., SHARAFI, P. "Nodal ordering for bandwidth reduction using ant system algorithm", *Engineering Computations*, v. 26, n. 3, pp. 313–323, 2009.

[2] BENZI, M. "Preconditioning techniques for large linear systems: a survey", *Journal of computational Physics*, v. 182, n. 2, pp. 418–477, 2002.

[3] CARMO, F. C. D. *Análise da influência de algoritmos de reordenação de matrizes esparsas no desempenho do método CCCG (n)*. Dissertação de mestrado, Universidade Federal de Minas Gerais, 2005.

[4] BERMAN, A. "Nonnegative Matrices in the Mathematical Sciences", *Classics Appl. Math.*, 1994. Disponível em: <`https://ci.nii.ac.jp/naid/10021857718/en/`>.

[5] CARVALHO, L., MACULAN, N., LAGO, R., et al. "Algebra Linear Numérica e Computacional-Métodos de Krylov para Soluçao de Sistemas Lineares", *Editora Ciência Moderna*, 2010.

[6] BERN, M., GILBERT, J. R., HENDRICKSON, B., et al. "Support-graph preconditioners", *SIAM Journal on Matrix Analysis and Applications*, v. 27, n. 4, pp. 930–951, 2006.

[7] GEORGE, A., LIU, J. W. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981. ISBN: 0131652745.

[8] DUFF, I. S., ERISMAN, A. M., REID, J. K. *Direct Methods for Sparse Matrices*. New York, NY, USA, Oxford University Press, Inc., 1986. ISBN: 0-198-53408-6.

[9] TARJAN, R. E. *Graph theory and Gaussian elimination*. Computer Science Department, School of Humanities and Sciences, Stanford University, 1975.

[10] DAVIS, T. A. *Direct methods for sparse linear systems*, v. 2. Siam, 2006.

[11] WATKINS, D. S. *Fundamentals of matrix computations*, v. 64. John Wiley & Sons, 2004.

[12] VARGA, R. S. *Matrix iterative analysis*, v. 27. Springer Science & Business Media, 2009.

[13] VARGA, R. S. "Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics (Eugene L. Wachspress)", *SIAM Review*, v. 9, n. 4, pp. 756–758, 1967.

[14] YOUNG, D. M. *Iterative solution of large linear systems*. Elsevier, 2014.

[15] GIBBS, N. E. *A Survey of Bandwidth and Profile Reduction Algorithms.* Relatório técnico, DTIC Document, 1980.

[16] PAPADIMITRIOU, C. H. "The NP-completeness of the bandwidth minimization problem", *Computing*, v. 16, n. 3, pp. 263–270, 1976.

[17] DUFF, I. S., MEURANT, G. A. "The effect of ordering on preconditioned conjugate gradients", *BIT Numerical Mathematics*, v. 29, n. 4, pp. 635–657, 1989.

[18] CUTHILL, E., MCKEE, J. "Reducing the Bandwidth of Sparse Symmetric Matrices". In: *Proceedings of the 1969 24th National Conference*, ACM '69, pp. 157–172, New York, NY, USA, 1969. ACM. doi: 10.1145/800195.805928. Disponível em: <http://doi.acm.org/10.1145/800195.805928>.

[19] GEORGE, J. A. *Computer implementation of the finite element method.* Relatório técnico, DTIC Document, 1971.

[20] DUFF, I. S. "Computer Solution of Large Sparse Positive Definite Systems (Alan George and Joseph W. Liu", *SIAM Review*, v. 26, n. 2, pp. 289–291, 1984.

[21] PARTER, S. "The use of linear graphs in Gauss elimination", *SIAM review*, v. 3, n. 2, pp. 119–130, 1961.

[22] DE OLIVEIRA, S. L. G., BERNARDES, J. A., CHAGAS, G. O. "An evaluation of low-cost heuristics for matrix bandwidth and profile reductions", *Computational and Applied Mathematics*, v. 37, n. 2, pp. 1412–1471, 2018.

[23] 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, pp. 531–546, 1984.

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

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

[26] NEMHAUSER, G. L., WOLSEY, L. A. *Integer programming and combinatorial optimization*, v. 20. Springer, 1988.

[27] LAND, A. H., DOIG, A. G. "An Automatic Method for Solving Discrete Programming Problems". In: Jünger, M., Liebling, T. M., Naddef, D., et al. (Eds.), *50 Years of Integer Programming 1958-2008*, Springer Berlin Heidelberg, cap. 5, pp. 105–132, Berlin, Heidelberg, 2010.

[28] LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., et al. "An Algorithm for the Traveling Salesman Problem", *Operations Research*, v. 11, n. 6, 1963.

[29] MURTY, K. G. *Operations Research: Deterministic Optimization Models*. Prentice-Hall, Inc., 1995.

[30] BLUM, A., KONJEVOD, G., RAVI, R., et al. "Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 100–105. ACM, 1998.

[31] DUMITRESCU, I., STÜTZLE, T. "Combinations of local search and exact algorithms". In: *Workshops on Applications of Evolutionary Computation*, pp. 211–223. Springer, 2003.

[32] ROMERO, R.; RIDER, L. M. "Introdução a metaheurísticas aplicadas a sistemas elétricos de potência", *Minicurso do Simposio Brasileiro de Sistemas Elétricos, SBSE, Goiania*, 2012.

[33] ROMERO, R., MANTOVANI, J. "Introdução a Metaheurísticas, Anais do 3º Congresso Temático de Dinâmica e Controle da SBMAC, UNESP, Campus de Ilha Solteira, Brasil", *Tec Art Editora*, 2004.

[34] GLOVER, F., KOCHENBERGER, G. *Handbook of metaheuristics*. Kluwer Academic Publishers, 2003.

[35] BLUM, C., ROLI, A. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", *ACM Computing Surveys (CSUR)*, v. 35, n. 3, pp. 268–308, 2003.

[36] GLOVER, F. "Tabu search-part I", *ORSA Journal on computing*, v. 1, n. 3, pp. 190–206, 1989.

[37] RESENDE, M. G., RIBEIRO, C. C. *Optimization by GRASP*. Springer, 2016.

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

[39] HANSEN, P., MLADENOVIĆ, N., PÉREZ, J. A. M. "Variable neighbourhood search: methods and applications", *Annals of Operations Research*, v. 175, n. 1, pp. 367–407, 2010.

[40] HANSEN, P., MLADENOVIĆ, N., TODOSIJEVIĆ, R., et al. "Variable neighborhood search: basics and variants", *EURO Journal on Computational Optimization*, v. 5, n. 3, pp. 423–454, 2017.

[41] HANSEN, P., MLADENOVIĆ, N. "A Tutorial on Variable Neighborhood Search". In: *LES CAHIERS DU GERAD, HEC MONTREAL AND GERAD*. Citeseer, 2003.

[42] POSSAGNOLO, L. H. F. M. *Reconfiguração de sistemas de distribuição operando em vários níveis de demanda através de uma meta-heurística de busca em vizinhança variável*. Dissertação de mestrado, Universidade Estadual Paulista (UNESP), 2015.

[43] SOUZA, R. F. F. *Planejamento da expansão de sistemas de distribuição usando a metaheurística de busca em vizinhança variável*. Dissertação de mestrado, Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2011.

[44] HANSEN, P., MLADENOVIĆ, N. "Variable Neighborhood Search", *Handbook of Metaheuristics*, pp. 145–184, 2003.

[45] SANTOS, D. B. *Planejamento da expansão de sistemas de transmissão usando a metaheurística de busca em vizinhança variável*. Tese de Doutorado, Universidade Estadual Paulista Júlio de Mesquita Filho, Faculdade de Engenharia de Ilha Solteira,, 2013.

[46] HANSEN, P., MLADENOVIĆ, N. "Variable neighborhood search: Principles and applications", *European journal of operational research*, v. 130, n. 3, pp. 449–467, 2001.

[47] GLOVER, F. "Tabu search—part II", *ORSA Journal on computing*, v. 2, n. 1, pp. 4–32, 1990.

[48] BURKE, E., DE CAUSMAECKER, P., PETROVIC, S., et al. "Variable neighborhood search for nurse rostering problems". In: *Metaheuristics: computer decision-making*, Springer, pp. 153–172, Boston, MA, 2003.

[49] KOVAČEVIĆ-VUJČIĆ, V., ČANGALOVIĆ, M., AŠIĆ, M., et al. "Tabu search methodology in global optimization", *Computers & Mathematics with Applications*, v. 37, n. 4-5, pp. 125–133, 1999.

[50] BRÄYSY, O. "A reactive variable neighborhood search for the vehicle-routing problem with time windows", *INFORMS Journal on Computing*, v. 15, n. 4, pp. 347–368, 2003.

[51] BRIMBERG, J., HANSEN, P., MLADENOVIĆ, N., et al. "Improvements and comparison of heuristics for solving the uncapacitated multisource Weber problem", *Operations research*, v. 48, n. 3, pp. 444–460, 2000.

[52] DAVIDOVIC, T., HANSEN, P., MLADENOVIC, N. "Variable neighborhood search for multiprocessor scheduling problem with communication delays". In: *Proc. MIC*, v. 4, pp. 737–741, 2001.

[53] FEO, T. A., RESENDE, M. G. "Greedy randomized adaptive search procedures", *Journal of global optimization*, v. 6, n. 2, pp. 109–133, 1995.

[54] MARTINS, S. L., RESENDE, M. G., RIBEIRO, C. C., et al. "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", *Journal of Global Optimization*, v. 17, n. 1-4, pp. 267–283, 2000.

[55] RIBEIRO, C. C., UCHOA, E., WERNECK, R. F. "A hybrid GRASP with perturbations for the Steiner problem in graphs", *INFORMS Journal on Computing*, v. 14, n. 3, pp. 228–246, 2002.

[56] FESTA, P., PARDALOS, P., RESENDE, M., et al. "GRASP and VNS for Max-Cut". In: *Extended Abstracts of the Fourth Metaheuristics International Conference*, pp. 371–376, 2001.

[57] ANDREATTA, A. A., RIBEIRO, C. C. "Heuristics for the phylogeny problem", *Journal of Heuristics*, v. 8, n. 4, pp. 429–447, 2002.

[58] CANUTO, S. A., RESENDE, M. G., RIBEIRO, C. C. "Local search with perturbations for the prize-collecting Steiner tree problem in graphs", *Networks: An International Journal*, v. 38, n. 1, pp. 50–58, 2001.

[59] OCHI, L. S., SILVA, M. B., DRUMMOND, L. "Metaheuristics based on GRASP and VNS for solving traveling purchaser problem". In: *Metaheuristics International Conference*, pp. 489–494, 2001.

[60] CORBERÁN, A., FERNÁNDEZ, E., LAGUNA, M., et al. "Heuristic solutions to the problem of routing school buses with multiple objectives", *Journal of the operational research society*, v. 53, n. 4, pp. 427–435, 2002.

[61] BELACEL, N., HANSEN, P., MLADENOVIC, N. "Fuzzy J-means: a new heuristic for fuzzy clustering", *Pattern Recognition*, v. 35, n. 10, pp. 2193–2200, 2002.

[62] FLESZAR, K., HINDI, K. S. "New heuristics for one-dimensional bin-packing", *Computers & operations research*, v. 29, n. 7, pp. 821–839, 2002.

[63] WHITAKER, R. "A fast algorithm for the greedy interchange for large-scale clustering and median location problems", *INFOR: Information Systems and Operational Research*, v. 21, n. 2, pp. 95–108, 1983.

[64] SILVA, M., DRUMMOND, L., OCHI, L. "Metaheurısticas grasp+ vns para a soluçao de problemas de otimizaçao combinatória". In: *Congresso brasileiro de pesquisa operacional*, v. 32, 2000.

[65] RIBEIRO, C. C., SOUZA, M. C. "Variable neighborhood search for the degree-constrained minimum spanning tree problem", *Discrete Applied Mathematics*, v. 118, n. 1-2, pp. 43–54, 2002.

[66] QU, R., BURKE, E. "Hybrid variable neighborhood hyperheuristics for exam timetabling problems", *Citeseer*, 2005.

[67] BRIMBERG, J., HANSEN, P., MLADENOVIĆ, N. "Convergence of variable neighborhood search", *Cahiers du GERAD*, 2002.

[68] HANSEN, P., JAUMARD, B., MLADENOVIĆ, N., et al. "Variable neighbourhood search for maximum weight satisfiability problem", *Cahiers du GERAD*, 2000.

[69] HANSEN, P., MLADENOVIĆ, N., BRIMBERG, J., et al. "Variable neighborhood search". In: *Handbook of metaheuristics*, pp. 57–97. Springer, 2019.

[70] MARTÍ, R., LAGUNA, M., GLOVER, F., et al. "Reducing the bandwidth of a sparse matrix with tabu search", *European Journal of Operational Research*, v. 135, n. 2, pp. 450–459, 2001.

[71] PINANA, E., PLANA, I., CAMPOS, V., et al. "GRASP and path relinking for the matrix bandwidth minimization", *European Journal of Operational Research*, v. 153, n. 1, pp. 200–210, 2004.

[72] MLADENOVIC, N., UROSEVIC, D., PÉREZ-BRITO, D., et al. "Variable neighbourhood search for bandwidth reduction", *European Journal of Operational Research*, v. 200, n. 1, pp. 14–27, 2010.

[73] LIM, A., RODRIGUES, B., XIAO, F. "Heuristics for matrix bandwidth reduction", *European Journal of Operational Research*, v. 174, n. 1, pp. 69–91, 2006.

[74] LIM, A., RODRIGUES, B., XIAO, F. "A fast algorithm for bandwidth minimization", *International Journal on Artificial Intelligence Tools*, v. 16, n. 03, pp. 537–544, 2007.

[75] HAMMING, R. W. "Error detecting and error correcting codes", *The Bell system technical journal*, v. 29, n. 2, pp. 147–160, 1950.

[76] RODRIGUEZ-TELLO, E., HAO, J.-K., TORRES-JIMENEZ, J. "An improved simulated annealing algorithm for bandwidth minimization", *European Journal of Operational Research*, v. 185, n. 3, pp. 1319–1335, 2008.

[77] HANSEN, P., MLADENOVIĆ, N. "Variable Neighborhood Search Methods". In: *Encyclopedia of Optimization*, pp. 3975–3989, Boston, MA, Springer US, 2009. ISBN: 978-0-387-74759-0. doi: 10.1007/978-0-387-74759-0_694. Disponível em: <https://doi.org/10.1007/978-0-387-74759-0_694>.

[78] VO, T. K. *Exact and Heuristic Solutions to the Bandwidth Minimization Problem.* Tese de Doutorado, Universität Heidelberg, 2011.

[79] CAPRARA, A., LETCHFORD, A. N., SALAZAR-GONZÁLEZ, J.-J. "Decorous Lower Bounds for Minimum Linear Arrangement", *INFORMS Journal on Computing*, v. 23, n. 1, pp. 26–40, 2011.

[80] NEMHAUSER, G. L., WOLSEY, L. A. "Integer programming and combinatorial optimization", *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, v. 20, pp. 8–12, 1988.

[81] SEITZ, H. *Contributions to the minimum linear arrangement problem.* Tese de Doutorado, Heidelberg University, 2010.

[82] FERREIRA, M. D. S. *Problema do arranjo linear mínimo.* Dissertação de mestrado, Universidade Federal do Ceará, 2016.