UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

GABRIEL LOPES LIMA

ALGORITMOS PARA RESOLUÇÃO DO PROBLEMA DO CORTE MÁXIMO:

abordagem exata e meta-heurísticas

Recife

2022

GABRIEL LOPES LIMA

ALGORITMOS PARA RESOLUÇÃO DO PROBLEMA DO CORTE MÁXIMO:

abordagem exata e meta-heurísticas

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, como requisito parcial para obtenção do título de mestre em Engenharia de Produção.

Área de concentração: Pesquisa Operacional.

Orientadora: Profª. Drª. Isis Didier Lins.

Recife

2022

Catalogação na fonte
Bibliotecária Margareth Malta, CRB-4 / 1198

L732a Lima, Gabriel Lopes.
Algoritmos para resolução do problema do corte máximo: abordagem exata e meta-heurísticas / Gabriel Lopes Lima. - 2022.
87 folhas, il., gráfs., tabs.

Orientadora: Profa. Dra. Isis Didier Lins.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia de Produção, 2022.
Inclui Referências e Apêndices.

1. Engenharia de Produção. 2. Problema do Max-Cut. 3. Abordagem exata. 4. Árvore de busca binária. 5. Meta-heurísticas. I. Lins, Isis Didier (Orientadora). II. Título.

UFPE

658.5 CDD (22. ed.) BCTG/2022-280

GABRIEL LOPES LIMA

ALGORITMOS PARA RESOLUÇÃO DO PROBLEMA DO CORTE MÁXIMO:

abordagem exata e meta-heurísticas

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, como requisito parcial para obtenção do título de mestre em Engenharia de Produção.

Área de concentração: Pesquisa Operacional.

Aprovada em: 03/03/2022.

BANCA EXAMINADORA

_____

Profª. Drª. Isis Didier Lins (Orientadora)
Universidade Federal de Pernambuco

_____

Prof. Dr. Raphael Harry Frederico Ribeiro Kramer (Examinador Interno)
Universidade Federal de Pernambuco

_____

Prof. Dr. Reinaldo Morabito Neto (Examinador Externo)
Universidade Federal de São Carlos

Para os meus queridos pais Valmir e
Rosimeire e minha doce irmã Damaris.

AGRADECIMENTOS

Agradeço primeiramente ao meu Rei Jesus Cristo, que me concedeu muita saúde em meio a uma situação pandêmica e ainda me proporcionou oportunidades para um amadurecimento cientifico e intelectual, abrindo os meus olhos para compreender de maneira sistemática as realidades do mundo da vida.

Agradeço aos meus pais Valmir e Rosimeire e minha irmã Damaris que sempre foram um refúgio em meio as intempéries, meu porto seguro, as pessoas que eu sei que sempre poderei contar. Agradeço a minha namorada Larissa, que nunca mediu esforços para me ajudar em quaisquer que fossem as dificuldades encontradas. E também agradeço a minha avó Dona Maria, que sempre esteve com seus joelhos dobrados nas madrugadas orando pela minha vida.

Agradeço imensamente ao professor Sóstenes Lins, que me escolheu como seu último orientando na carreira cientifica, me concedendo a oportunidade de grandes desafios e aprendizados, principalmente por ter compartilhado comigo suas anotações de anos de estudo, que aprimoradas e implementadas formaram a abordagem apresentada por esta pesquisa. Tornando assim os escritos desse estudo, uma homenagem para sua última exploração cientifica antes de sua aposentadoria.

Agradeço a minha orientadora Isis Lins, que como professora foi responsável por apresentar a base dos ensinamentos da otimização combinatória e outras disciplinas que proporcionaram grandes desafios e aprendizados. Mas agradeço principalmente por assumir a orientação dessa pesquisa com a saída do professor Sóstenes, não medindo esforços para contribuir com o aprimoramento e evolução deste estudo.

Agradeço ao professor Raphael Kramer que fez com que seus ensinamentos em Tópicos Avançados em Pesquisa Operacional II fossem um divisor de águas na minha vida, abrindo os meus olhos para a grande importância dos algoritmos de otimização e me forçando a perder muitas noites de sono até aprender implementar algoritmos e programar com *python*. Na correção dessa pesquisa fez uma análise minuciosa, revelando lacunas que precisavam ser preenchidas.

Agradeço ao professor Reinaldo Morabito pela participação na banca e contribuição na melhoria dessa pesquisa.

Agradeço a cada um dos professores que eu tive no PPGEP-UFPE, cada um deles deixou sua marca especial no decorrer do mestrado. Cada dica, cada história e cada ensinamento foi de grande valia para vencer as barreiras e as dificuldades do programa.

Aos amigos do programa Augusto, Igor, Eugenio, Amanda, Anderson, Allan, Luan e Bruno que tornaram o processo de aprendizado mais dinâmico com companheirismo e respeito.

Agradeço imensamente ao Concelho Nacional de Desenvolvimento Científico e Tecnológico - CNPq. Que desde a graduação com os programas de iniciação científica e agora com a bolsa de pesquisa de mestrado foi fundamental para que eu conseguisse me dedicar exclusivamente aos estudos.

RESUMO

O problema do corte máximo é um dos problemas de natureza combinatória amplamente abordados na literatura. Neste problema dado um grafo não direcionado e ponderado $G = (V, A)$, busca-se particionar o conjunto de vértices em dois subconjuntos, tal que a soma dos pesos das arestas que conectam os dois subconjuntos seja maximizada. Apesar da simplicidade da sua definição, o problema do corte máximo é NP-Completo, sendo classificado também como NP- Difícil, ou seja, uma classe de problemas para a qual, até o momento, não foi definido um método para encontrar uma solução ótima em tempo polinomial de maneira determinística. Para resolver esses problemas com alta complexidade são usadas, na maioria das vezes, abordagens meta-heurísticas. Desta forma, diferente dos algoritmos já propostos na literatura e visando explorar a possibilidade de uma nova abordagem exata, capaz de explorar o espaço de soluções e encontrar a solução ótima para o problema, este estudo propõe um algoritmo de árvore de busca e desenvolve cinco algoritmos meta-heurísticos para solução do problema do *max-cut*. Esta pesquisa foi dividida em quatro fases: (i) desenvolvimento e implementação da abordagem exata para a solução do problema; (ii) implementação de cinco meta-heurísticas; (iii) execução de experimentos computacionais com diferentes tipos de instâncias de grafos; (iv) realização de análises estatísticas. Os resultados mostraram que para a solução do *max-cut* o método exato proposto é eficaz e preferível em casos de pequeno porte, já os métodos meta-heurísticos existentes, em casos de grande porte, apresentam um melhor *trade-off* entre encontrar uma boa solução e o tempo necessário para tal, pois, nessa primeira versão do algoritmo Árvore de Busca Binária (ABB), a complexidade ainda é muito elevada para solucionar instâncias de grafos mais densos. A fim de resolver esse desafio da complexidade são necessárias explorações mais aprofundadas no algoritmo para encontrar um novo critério de contradição, para assim ser viável encontrar a solução ótima do problema em tempos aceitáveis para qualquer grafo.

Palavras-chave: problema do *Max-Cut*; abordagem exata; árvore de busca binária; meta-heurísticas.

ABSTRACT

The maximum cut problem is one of the combinatorial problems widely discussed in the literature. In this problem, given an undirected weighed graph G = (V,A), we seek to partition the set of vertices into two subsets, such that the sum of the weights of the edges connecting the two subsets is maximized. Despite the simplicity of its definition, the maximum cut problem is NP-Hard, that is, a class of problems for which, so far, no method to find an optimal solution has been defined in polynomial time in a deterministic way. To solve these highly complex problems, meta-heuristic approaches are mostly used. In this way, different from the algorithms already proposed in the literature and aiming to explore the possibility of a new exact approach, capable of exploring the space of solutions and finding the optimal solution for the problem, this study proposes a search tree algorithm and develops five metaheuristic algorithms to solve the max-cut problem. This research was divided into four phases: (i) development and implementation of the exact approach to solve the problem; (ii) implementation of five meta-heuristics; (iii) execution of computational experiments with different types instances; (iv) performing statistical analyses. The results showed that for the max-cut solution the proposed exact method is effective and preferable in small cases, whereas the existing meta-heuristic methods, in large cases, present a better tradeoff between finding a good solution and the time required for that, because, in this first version of the Binary Search Tree (ABB) algorithm, the complexities are still high to solve instances of denser graphs. To solve this complexity challenge, deeper explorations are needed in the algorithm to find a new contradiction criterion, so that it is feasible to find the optimal solution to the problem in acceptable times for any graph.

Keywords: *Max-Cut* Problem; exact approach; binary search tree; metaheuristics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

A	Aresta ou Arco
BT	Busca Tabu
G	Grafo
GA	Algoritmos Genéticos
GRASP	Greedy Randomized Adaptive Search Procedure
IA	Inteligência Artificial
LRC	Lista Restrita de Candidatos
Max-Cut	Maximum Cut Problem ou Problema do Corte Máximo
MCD	Memória de Curta Duração ou Lista Tabu
MH	Metaheuristicas
NP	Nondeterministic polynomial time
NP-Completo	Nondeterministic polynomial time complete
PLM	Programa Linear Mestre
PO	Pesquisa Operacional
QUBO	Otimização Binária Irrestrita Quadrática
SMCP	Simple Max-Cut Problem ou Problema do Corte Máximo Simples
SSP	Subset Sum Problem ou Problema da Soma de Subconjuntos
unifs	Unificadores do caminho diagonal
V	Vértice ou Nó
VLSI	Circuitos Integrados De Grande Escala
VND	Variable Neighborhood Descent ou Descida em Vizinhança Variável
ORG_h	Grafo Orbital Radial h

SUMÁRIO

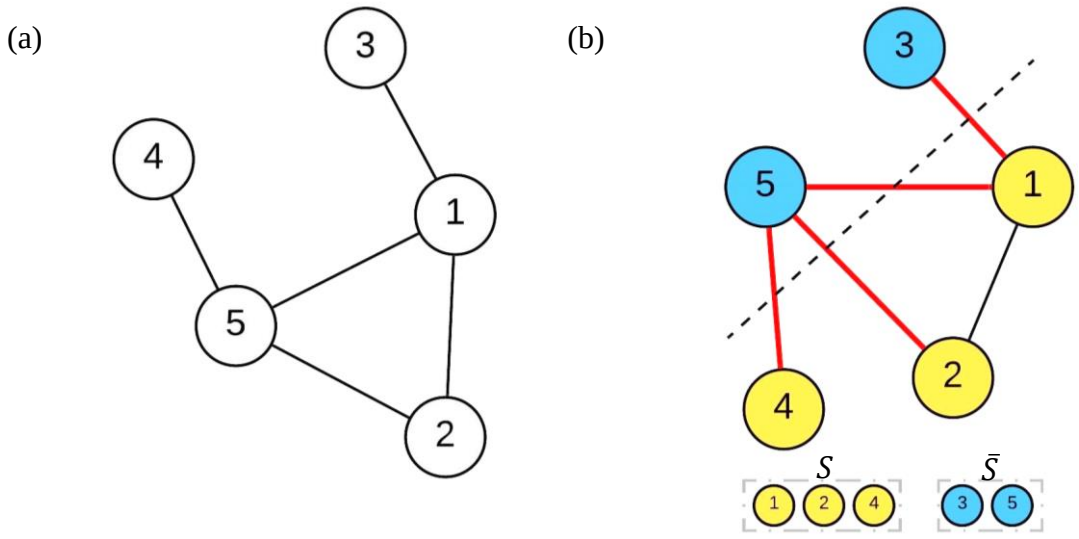
1 INTRODUÇÃO

Otimização combinatória é uma das áreas de estudo presente nas ciências exatas que busca, dentro de um conjunto de soluções, um subconjunto ótimo para um determinado problema, seja este de maximização ou de minimização. O espaço de soluções pode ser obtido por meio da combinação de diferentes valores para as variáveis de decisão, podendo ser custoso computacionalmente enumerar todo este conjunto. Sua aplicação prática está presente em contextos que buscam melhorar a utilização de recursos limitados, tornando sua aplicação importante para áreas como a Engenharia de Produção (BENGIO; LODI; PROUVOST, 2020).

1.1 DESCRIÇÃO DO PROBLEMA

Um dos problemas amplamente abordados na literatura, de natureza combinatória, se trata do problema do corte máximo (*maximum cut problem* ou *max-cut problem*). Dado um grafo não direcionado e ponderado $G = (V, A)$, sendo o conjunto de vértices V e o conjunto de arestas $A = \{(i, j): i, j \in V, i \neq j\}$, busca-se particionar o conjunto de vértices em dois subconjuntos, tal que a soma dos pesos das arestas que conectam os dois subconjuntos seja maximizada (KIM; YOON; GEEM, 2018). O problema pode ter ainda como instâncias grafos não ponderados, dessa forma adota-se o peso de valor 1 para as arestas. Assim o problema se reduz a determinar o corte de arestas de cardinalidade máxima, denominado na literatura de *simple max-cut problem* (GAREY; JOHNSON, 1979), um exemplo deste problema com os pesos omitidos, porém de valor igual a 1, é representado na Figura 1.

Figura 1 - (a) Grafo não direcionado com arestas de peso igual a 1 (b) Solução ótima do Simple Max-Cut Problem



Fonte: O autor (2022)

O *simple max-cut* faz parte dos problemas combinatórios clássicos da teoria dos grafos que têm sido amplamente estudados desde a década de 1960. Erdós (1967) mostrou que é sempre possível obter um corte de arestas com pelo menos metade das arestas do grafo. Na década seguinte, com a evolução da ciência da computação, foram definidos os conceitos de problemas de decisão NP-completos, sendo provado que o *simple max-cut* é classificado como NP-Completo (KARP, 1972; GAREY; JOHNSON, 1979).

Apesar da simplicidade da sua definição, o problema do corte máximo é classificado como NP- Difícil, ou seja, uma classe de problemas para os quais até o momento não foi encontrado um método para obter uma solução ótima em tempo polinomial de maneira determinística, pois dependendo do tamanho da instância do problema, atinge-se uma alta complexidade de operações que torna o tempo inviável (GUERRESCHI; MATSUURA, 2019). Assim, a resposta ótima para qualquer problema de corte máximo não pode ser encontrada pelos métodos já publicados em tempo polinomial.

Para resolver esses problemas com alta complexidade são usadas meta-heurísticas para otimização, buscando soluções ótimas para questões específicas. O processo de busca da solução pode ser realizado usando vários agentes que formam essencialmente um sistema de soluções em evolução usando um conjunto de regras ou equações matemáticas durante várias iterações. Essas iterações continuarão até que as soluções encontradas atendam a certos critérios predefinidos. A solução final, podendo ser ótima ou subótima, é considerada a solução do problema, com o sistema tendo atingido um estado de convergência (YANG; DEB, 2014).

Desta forma ao contrário do método exato, que gasta muito tempo computacional para encontrar a solução ótima, os métodos heurísticos podem encontrar rapidamente uma solução subótima. As técnicas meta-heurísticas são muito bem-sucedidas porque são suscetíveis de fornecer soluções a um custo computacional aceitável. Combinando um bom método heurístico com um método meta-heurístico clássico, muitos problemas práticos podem ser resolvidos e uma solução muito boa pode ser obtida (HUSSAIN et al., 2019).

Desta forma, diferente dos algoritmos já propostos na literatura, e visando explorar a possibilidade de uma nova abordagem exata, que seja capaz de explorar o espaço de soluções e encontrar a solução ótima para o problema, este estudo tem o seguinte questionamento: Como solucionar o problema do corte máximo com uma abordagem que encontre resultados ótimos com tempos de processamento computacional compatíveis aos requeridos pelos métodos meta-heurísticos?

1.2 JUSTIFICATIVA

O *max-cut* é um problema no campo da otimização combinatória que tem uma ampla gama de aplicação, como física, estatística, segmentação de imagens, circuitos integrados de grande escala (circuitos VLSI), entre outros, que vem atraindo o interesse científico de pesquisadores nas áreas de otimização matemática, matemática discreta e pesquisa operacional. A quantidade de problemas práticos e matemáticos que aparecem diretamente na forma do max-cut ou podem ser modelados desta forma torna cada vez mais necessário o desenvolvimento de algoritmos que alcancem bons resultados em um tempo viável (ARAÚJO, 2018).

Uma das importantes aplicações do problema de corte máximo se dá no agrupamento de dados (*Data Clustering*), problema este que busca formar um conjunto de elementos presentes na base de dados, considerando as suas devidas similaridades (OTTERBACH et al. 2017). Segundo Ding *et al.* (2001) utilizando o princípio do agrupamento *min-max* (*min-max clustering principle*), minimizando a similaridade entre os conjuntos e maximizando a similaridade nos conjuntos, o corte máximo pode ser utilizado para realizar a busca no grafo do par de conjuntos disjuntos que resultam na maior soma entre as distâncias (arestas ponderadas), encontrando assim um agrupamento nos dados.

Além do *data clustering*, o *max-cut* é utilizado em etapas de algoritmos de *machine learning* e inteligência artificial (IA) como o de reconhecimento de imagens, tornando assim ainda mais importante encontrar melhores soluções do *max-cut*, pois surge uma crescente motivação de pesquisadores de IA para investigarem e aplicarem novos algoritmos e técnicas para problemas difíceis de IA, há também contribuições em classe de algoritmos de Otimização Binária Irrestrita Quadrática (QUBO) que requerem hardware de mecânica quântica, chamados de algoritmos adiabáticos quânticos(POLJAK; TUZA, 1995; OTTERBACH et al. 2017).

O problema do *max-cut* é um exemplo da classe de problemas NP-completos, que são notoriamente difíceis de resolver. Outros problemas combinatórios que podem ser reduzidos ao *max-cut* são o problema de planejamento de layout ou problema de localização de instalações, que é de grande utilidade para as áreas de arquitetura, engenharia industrial e pesquisa operacional (ALIDAEE et al,1994). Há também ainda a aplicação de Krarup e Pruzan (1978) do problema de projetos auxiliado por computador (*Computer-aided layout design*), e problemas de gerenciamento de mensagens de tráfego (GALLO, 1980).

No entanto, existem algumas desvantagens em usar um algoritmo exato para o *max-cut*. Por se tratar de um problema NP-difícil (KARP, 1972), mesmo para instâncias de grafos com

apenas 20 vértices, é considerado um desafio computacional resolvê-lo de forma ótima (KRISLOCK et al., 2014). Por conseguinte, para instâncias desse tamanho ou maiores, faz-se necessário o uso de algoritmos que forneçam soluções próximas da solução ótima, tais como heurísticas, meta-heurísticas e algoritmos aproximativos (DUNNING et al., 2018; GOEMANS; WILLIAMSON, 1995).

A contribuição desse trabalho se dará na elaboração de um algoritmo exato e na implementação de cinco meta-heurísticas, que serão capazes de explorar o espaço de soluções e encontrar soluções ótimas e subótimas para o problema de forma robusta e eficiente, como também apresentar a implementação de um programa desenvolvido em linguagem de programação Python que retornará o corte máximo de um grafo, economizando tempo e trazendo um método inovador de solução para pesquisadores, indústrias e empresas.

1.3 OBJETIVOS

1.3.1 Objetivo geral

Propor uma nova abordagem exata com um algoritmo de árvore de busca binária e implementar cinco algoritmos meta-heurísticos para solução do *max-cut*.

1.3.2 Objetivos específicos

- Definir e descrever um algoritmo de árvore de busca para o método de solução exata;
- Implementar algoritmos meta-heurísticos para soluções subótimas;
- Implementar heurísticas de refinamento de busca para as soluções subótimas;
- Apresentar uma demonstração do funcionamento do algoritmo proposto com exemplos.
- Realizar experimento do algoritmo exato proposto em diferentes grafos aleatórios com diferentes densidades;
- Analisar o comportamento do algoritmo proposto nas diferentes classes de grafos;
- Realizar experimento comparativo entre o algoritmo exato proposto e as meta-heurísticas;
- Analisar a eficiência de solução e tempo computacional entre o método exato e as meta-heurísticas.

2 REFERENCIAL TEÓRICO

Neste tópico serão fundamentados temas como otimização combinatória, teoria dos grafos, problema do corte máximo e as cinco meta-heurísticas que serão implementadas, realizando assim a construção de uma base teórica para a realização do estudo.

2.1 OTIMIZAÇÃO COMBINATÓRIA

Otimização combinatória é o campo da pesquisa operacional, que busca métodos eficazes para projetar e operar um determinado sistema. Tomando uma decisão com base na melhor combinação do valor da variável do problema dentro de um tempo útil de execução computacional (HILLIER e LIEBERMAN, 2013).

Para Arenales et al.(2015) a pesquisa operacional (PO) é um ramo interdisciplinar da matemática aplicada que utiliza modelos estatísticos, matemáticos e algoritmos para auxiliar na tomada de decisão. É usado principalmente para analisar sistemas reais complexos e geralmente visa melhorar ou otimizar o desempenho. Os problemas determinísticos de PO podem ser divididos em várias categorias: programação linear, teoria dos jogos, programação dinâmica e programação não linear.

Em alguns casos, o modelo de PO é tão complexo que é impossível encontrar a melhor solução. Nesse caso, ainda é importante encontrar uma solução viável. Métodos heurísticos são frequentemente usados para encontrar essas soluções. Problemas clássicos de natureza combinatória como o *max-cut*, podem ser consultados nos trabalhos de Garey e Johnson (1979), Goldbarg e Luna (2005) e Arenales et al. (2015). Dentro da otimização combinatória é utilizada a teoria dos grafos para a representação das redes nos diversos problemas de diferentes áreas do conhecimento.

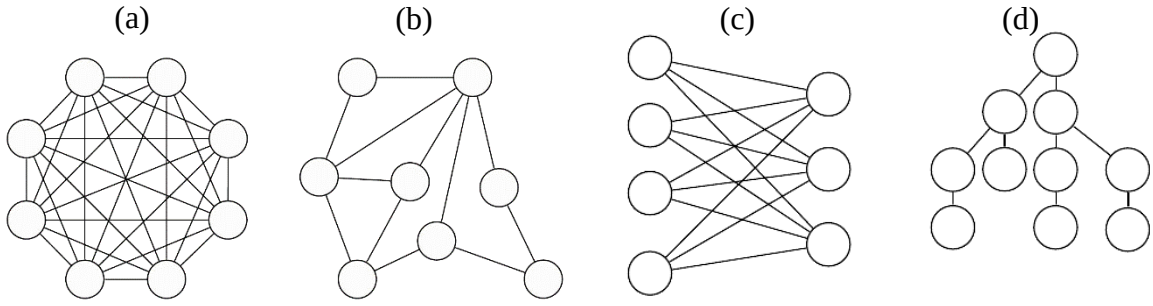
2.1.1 Teoria dos grafos

Segundo Netto e Jurkiewicz (2017), a teoria dos grafos é o campo da matemática que estuda uma estrutura que pode modelar vários problemas, representando matematicamente objetos classificados como vértices, podendo haver ou não relação entre eles, sendo chamadas de arestas ou arcos. O primeiro problema conhecido da teoria dos grafos foi modelado por Leonhard Euler em 1736, sendo chamado o problema das pontes, onde foi explorada a possibilidade de visitar a cidade alemã de Königsberg, saindo e retornando ao mesmo lugar de partida, passando por cada uma das sete pontes que cruzam um rio chamado Pregel exatamente uma vez, algo que foi constatado por Euler ser impossível (PRESTES, 2016).

A representação matemática de um grafo é *G = (V, A)*, que segundo Araújo (2018) consiste num conjunto finito não vazio *V* de elementos, chamados vértices, e um conjunto de pares não-ordenados de elementos distintos de *V*, chamado arestas, isto é, $A \subseteq \{(i,j): i,j \in V, i \neq j\}$. A densidade de um grafo é dada pela relação entre sua ordem (número de vértices) e seu tamanho (número de vértices e arestas), ou seja, representa a razão entre o número de arestas *A* e o número de vértices *V*. A proporção divide grafos em densos ou esparsos. Assim considerando um grafo completo G', onde cada vértice pertencente a $V\ (G\ ')$ tem conectado a ele $(|\ V\ (G')\ | - 1)$ arestas, há uma relação de que quanto mais próximo $|A(G)|$ for de $|A(G')|$ mais denso esse grafo G será considerando. Quando menor o valor de $|A(G)|$ mais esparso o grafo G será considerado.

Certos tipos de grafos desempenham um papel importante na teoria dos grafos e em suas aplicações, por isso são classificados em famílias especiais. Essas classificações refletem a forma do grafo e, portanto, são fáceis de entender. Os grafos completos são grafos simples (que não apresentam loops e arestas paralelas), onde há uma aresta entre cada par de seus vértices, sendo matematicamente designados por K_n, onde *n* é a ordem do grafo, o grafo ilustrado na Figura 2(a) é um grafo completo com 8 vértices. Grafos aleatórios são muito úteis para testar algoritmos, onde em sua criação é definida uma densidade de arestas podendo ser mais esparsos ou mais densos, e as arestas entre os vértices são ligadas aleatoriamente. O grafo ilustrado na Figura 2(b) representa um grafo aleatório, sendo esses os que serão utilizados para testar os algoritmos nesse estudo (ARENALES et al.,2015).

Figura 2 - (a) Grafo completo K_8 (b) grafo aleatório com 8 vértices , (c) grafo bipartido completo $K_{4,3}$ e (d) grafo árvore



Fonte: O autor (2022)

Um grafo é dito ser bipartido quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2, tais que toda aresta de G une um vértice de V_1 a outro de V_2. O grafo da Figura 2(c) é um $K_{4,3}$, ou seja, um grafo bipartido completo que contém duas partições, uma com 4 vértices e a outra com 2 vértices. Ele é completo pois todos os vértices de uma

partição estão ligados a todos os vértices da outra partição. Por fim, uma árvore é um grafo acíclico e conexo. Um grafo é conexo se há pelo menos uma cadeia ligando cada par de vértices deste grafo, e acíclico quando não há cadeias fechadas entre os vértices, como o grafo da Figura 2(d). Há também os grafos planares que são grafos expressos em um plano sem que ocorra o cruzamento de arestas, como na Figura 2(b). Assim problemas combinatórios, como o problema do *max-cut* são modelados utilizando a teoria dos grafos com suas representações como instâncias para a solução.

2.2 PROBLEMA DO CORTE MÁXIMO

O problema do corte máximo ou *max-cut* pode ser descrito da seguinte forma: dado um grafo não direcionado *G = (V,A)* com |*V*| = *n* vértices e |*A*| = *m* arestas ponderadas. Qualquer bipartição dos vértices de *V*, representada por *(S, $\bar{S}$)*, define um corte de *G*, ou seja, o subconjunto de arestas com uma extremidade em *S* e a outra em $\bar{S}$, onde *S* ou $\bar{S}$ pode ser vazio. O *max-cut* consiste em encontrar *(S, $\bar{S}$)* tal que a soma dos pesos das arestas do corte seja o maior possível (FARIA, 2020). O *max-cut* possui muitas formulações equivalentes, as quais aparecem com diferentes nomes na literatura, além de *max-cut* as denominações mais comuns são: *Maximum-2-Satisfiability*, *Weighted Signed Graph Balancing* e *Unconstrained Quadratic 0-1 programming* (HAMER, 1991).

No particionamento dos subconjuntos, dado um grafo *G = (V, A)*, uma partição *(S, $\bar{S}$)* de *V* é uma decomposição de *V* como união disjunta de dois subconjuntos *S* e $\bar{S}$, ou seja, $V = S \cup \bar{S}$ com $S \cap \bar{S} = \emptyset$. Dado um grafo G = (V, A) e $S, \bar{S} \subset V$, o conjunto formado pelas arestas que possuem um dos extremos em S e o outro em $\bar{S}$ será representado por $[S, \bar{S}]$. O subconjunto de S formado pelos vértices que possuem apenas um dos extremos em S $\subset$ V , o qual é representado por $[S, \bar{S}] = \delta(S)$ é chamado de corte de arestas de G. As arestas de $\delta(S)$ são ditas arestas de corte. A remoção de todas as arestas de $\delta(S)$ de G produz um subgrafo de G desconexo (SUCUPIRA, 2017).

De modo genérico, num grafo *G = (V, A)* com $|V| = n$, pode-se rotular os vértices com os inteiros de 1 a *n*, isto é, $V = \{1, 2, \cdots, n\}$ e assim representar as arestas de *G* como pares não ordenados de vértices $\{i, j\}$ com $i < j$. O peso de uma aresta $\{i, j\}$ será representado por c_{ij} . Se o peso das arestas de *G* for sempre positivo, considerando que todo grafo *G* com *n* vértices é subgrafo do grafo completo K_n, pode-se considerar sempre o grafo completo K_n e atribuir o valor $c_{ij} = 0$ caso os vértices *i* e *j* de K_n não sejam adjacentes em *G*.

Segundo Faria (2020) o problema de corte máximo pode ser reduzido a um problema de satisfabilidade, bem conhecido da área da computação e se relaciona com outros, como os problemas de partição. O problema do *max-cut* (G, ω), pode ser formulado com uma função objetivo de otimização como descrito na equação (2.1).

$$\omega\big(\delta(S)\big) = \sum_{i \in S, j \in \bar{S}} \omega_{ij} \qquad (2.1)$$
$$\mathrm{MAXCUT}(\mathrm{G}, \omega) = \max_{S \subset V} \omega\big(\delta(S)\big)$$

Em que se quer maximizar $\omega\big(\delta(S)\big)$, que representa o corte das arestas ω_{ij}.De acordo com Sucupira (2017) o problema do corte máximo pode ser simplificado, considerando-se todas as arestas com igual peso, que pode ser tomado unitário. Nesse caso, o que interessa é a cardinalidade do corte de arestas encontrado, ou seja, o corte máximo é aquele que apresenta o maior número de arestas e o problema do corte máximo simples (*simple max-cut*) pode ser formulado como um problema de decisão no seguinte formato, com a instância sendo um grafo simples *G = (V, A)* e um inteiro positivo v^{ω}, que representa o corte procurado. Tem-se a questão : Existe um subconjunto $S \subset V$ tal que o número de arestas com um extremo em *S* e o outro em *V / S* é igual a v^{ω}? Esse problema, apesar de sua formulação simples, tem uma extrema complexidade para ser resolvido com métodos exatos, sendo utilizadas abordagens meta-heurísticas para soluções mais rápidas, porém, nem sempre ótimas.

2.3 META-HEURÍSTICAS

Os métodos heurísticos são entendidos como uma técnica inspirada em processos intuitivos, é um método de raciocínio para solução de problemas que utiliza técnicas de tentativa e erro. Entre eles, os métodos heurísticos construtivo e de refinamento se destacam. As heurísticas construtivas visam produzir soluções inicialmente viáveis, enquanto as de refinamento aprimoram os resultados inicialmente encontrados (WANG, 2010).

Como o método heurístico realiza uma busca limitada no espaço de solução, uma meta-heurística como o próprio termo "meta" sugere, estão um nível acima das heurísticas, por ser um método heurístico geral ou mais aprimorado, mais utilizado na resolução de problemas combinatórios como o *max-cut* nessa pesquisa. O método meta-heurístico segue o princípio de intensificação e diversificação. O objetivo da intensificação é conduzir uma exploração mais detalhada das áreas mais promissoras do espaço de busca, enquanto a diversificação pode evitar que uma determinada solução fique presa no processo de busca em ótimos locais (HUSSAIN et al., 2019).

A computação meta-heurística (MH) é uma computação adaptativa que aplica regras heurísticas gerais na solução de uma categoria de problemas computacionais. Para alcançar bons resultados, a MH deve expandir a busca em uma ampla gama de áreas não visitadas, concentrando-se em áreas promissoras (soluções de alta qualidade) por meio de experiência das iterações de busca para convergir de forma otimizada (KHAJEHZADEH et al. 2011). Para isso um algoritmo eficaz propaga uma nova solução por meio de randomização e passos aleatórios para mantê-la longe da área de busca atual, de forma que o movimento de exploração alcance todas as áreas do espaço de busca visitadas pelo menos uma vez. Por outro lado, usando as informações sobre os resultados obtidos em buscas locais intensivas e experiências anteriores de busca, o algoritmo tenta convergir rapidamente sem utilizar movimentos excessivos (YANG et al., 2015).

Para a solução do *max-cut* algoritmos de otimização de busca local eficientes, que tem métodos que expandem a busca em uma ampla gama de áreas ainda não visitadas, são a *Greedy Randomized Adaptive Search Procedure* (GRASP), a *Variable Neighborhood Search* (VNS), Busca Tabu (BT) e a *Simulated Annealing* (SA) as quais foram implementadas neste estudo. Nos métodos de pesquisa global, que focam em áreas com soluções melhores buscando convergir em valores ótimos após diversas iterações, foi implementado um Algoritmo Genético (AG). Existem também muitos métodos híbridos que combinam as qualidades de algoritmos de busca local com algoritmos de busca global ou meta-heurísticas populacionais (LI; TIAN, 2015; PHAM; HUYNH, 2015; SAHLI et al., 2014; MIRJALILI, 2019)

2.3.1 GRASP

A tradicional meta-heurística GRASP, originalmente proposta por Feo e Resende (1989), é um algoritmo baseado em duas fases: uma fase de construção aleatória usando uma função gulosa; e uma fase de melhoria para atingir um ótimo local (LÓPEZ-SÁNCHEZ; SÁNCHEZ-ORO; HERNÁNDEZ-DÍAZ, 2019). Após fazer a leitura dos dados do problema, é necessário incluir uma regra que estabeleça o critério de parada do procedimento, ou seja, enquanto essa condição não for atendida (como um número específico de iterações) o algoritmo continua sendo executado (SOHRABI, ZIARATI e KESHTKARAN, 2019).

Para a etapa construtiva, inicialmente é necessário selecionar uma solução viável *s*, consolidada através da interação entre um elemento de cada vez, selecionado de forma aletória, com base em uma Lista de Candidatos Restrita (LCR) (RIBEIRO; PLASTINO; MARTINS, 2006). Essa solução se transformará na solução corrente. O conjunto LCR é composto pelos

elementos mais interessantes de uma lista constituída de candidatos a serem incluídos na solução, seguindo um critério guloso, e é exatamente essa técnica que garante diferentes soluções a cada interação da GRASP.

Para determinar a quantidade de elementos do conjunto LCR é necessário definir o valor do parâmetro α, que pode variar de *0* a *1*. Conforme apontam López-Sánchez, Sánchez-Oro e Hernández-Díaz (2019), se $\alpha = 0$, o conjunto LCR se comporta como um algoritmo heurístico construtivo do tipo guloso, já se $\alpha = 1$, todos os elementos candidatos fazem parte do conjunto LCR e o processo de escolha se torna totalmente aleatório. Dado o supracitado, ocorre a segunda fase da GRASP, que é a fase de melhoria local.

A busca local funciona como uma fase de refinamento. Essa fase busca na vizinhança de *s* uma melhora na função objetivo, e, caso encontre, ela é armazenada. O procedimento de melhoria é executado até que não se encontre um vizinho com melhor desempenho (CRAVO e AMARAL, 2019). É importante ressaltar que a fase de busca local depende de um ou mais operadores de vizinhança. Dado o fim do algoritmo, a solução é definida. A MH GRASP tem maior robustez quando na fase da busca local utiliza técnicas mais aprimoradas de pesquisa, como a *Variable Neighborhood Descent* e VNS, fazendo dessa forma MH híbridas com técnicas mais aprimoradas.

2.3.1.1 VND

Na etapa da busca local, para tornar o algoritmo mais robusto é realizada uma hibridização com o *Variable Neighborhood Descent* (VND) que é um método de busca local proposto por Mladenović e Hansen (1997), que utiliza distintos operadores de vizinhança para explorar o espaço de soluções, substituindo a solução corrente por uma nova apenas se houver melhoria (BESTEN; STÜTZLE, 2001; JABAL-AMELI; ARYANEZHAD; GHAFFARI-NASAB, 2011). Como estratégia de busca, geralmente são utilizadas as mais tradicionais, sendo elas: "*first improvement*" ou "*best improvement*" (HANSEN *et al.*, 2017).

Para ser considerado VND, o algoritmo deve possuir pelo menos dois operadores de vizinhança, fazendo com que seja mais provável alcançar um ótimo global do que com um único operador (HANSEN et al., 2019). No entanto, dependendo do problema abordado, a busca pelo melhor vizinho pode exigir um alto esforço computacional, principalmente se houver um grande conjunto de estruturas de vizinhança. Assim sendo, é importante que o usuário faça uma análise da quantidade de estruturas de vizinhança necessárias.

Ressalta-se que, conforme Resende e Ribeiro (2010), a eficiência da segunda fase depende de alguns aspectos, tais como: uma construção inicial de boa qualidade, uma estrutura de vizinhança robusta e, principalmente, da estratégia usada na fase de melhoria local. Mais detalhes da GRASP podem ser encontrados em Feo e Resende (1995) e Resende e Ribeiro (2019).

2.3.1.2 VNS

Para a hibridização da GRASP com o algoritmo meta-heurístico *Variable Neighborhood Search* (VNS), denominado em português busca em vizinhança variável, foi proposto por Mladenovic e Hansen (1997) para resolver problemas combinatórios, sendo inspirado em conceitos consolidados de otimização matemática e atua explorando em espaços de busca, a partir de uma solução inicial e tem como objetivo, realizar buscas em torno da solução inicial na procura de soluções melhores (DELGADO-ANTEQUERA et al., 2020).

Segundo Karakostas et al. (2020), fundamentalmente, esta meta-heurística possui um estágio para melhorar uma solução determinada e um estágio de perturbação para escapar de armadilhas de ótimos locais, pois a busca é realizada em um local diferente a cada vez, fazendo com que esse processo diversifique os mecanismos de busca. A fase de melhoria e o processo de perturbação e a etapa de mudança de vizinhança (incluindo a exploração do espaço de solução por meio de trocas sistemáticas de estruturas de vizinhança) são executados alternadamente até que o critério de parada predefinido seja atendido (HANSEN et al., 2016).

O VNS é inicializado gerando uma solução inicial e, a seguir, seleciona um conjunto de operadores de vizinhança, que representam o número de vizinhanças a serem exploradas. Partindo da solução inicial, é gerada a primeira vizinhança. Se a solução encontrada for melhor do que a solução atual, a solução vizinha se torna a nova solução. Se não houver solução melhor do que a atual em uma vizinhança, é explorada uma próxima vizinhança. Quando a última vizinhança do operador selecionado é analisada, o critério de parada é ativado (AFFI et al., 2018).

Ao contrário de outros métodos MH baseados em busca local, o método VNS não segue uma trajetória, mas gradualmente explora vizinhanças gradativamente mais distantes da solução corrente e focaliza a busca em torno de uma nova solução se e somente se um movimento de melhora é realizado. Em sua versão original, o método VNS utiliza o método VND para a etapa de busca local (KARAKOSTAS et al., 2020).

2.3.2 Busca tabu

A Busca tabu (BT) é um procedimento auxiliar que se adapta a um algoritmo de busca local de forma a orientá-lo na exploração contínua do espaço de soluções. A aplicação da busca tabu evita o retorno a uma mesma solução ótima local, já localizada anteriormente, de forma a superar essa solução e atingir um resultado ótimo ou próximo ao ótimo global (MARINHO, 2020). A busca tabu foi proposta por Glover (1989), este método consiste em uma rotina iterativa para construir vizinhanças com ênfase na proibição da parada em um ótimo local, o uso de estratégias para modificar a vizinhança da solução durante o processo de busca pode ultrapassar os limites do algoritmo de busca local clássico (HOU et al., 2020).

Ser tabu (proibido) visitar a área de busca que já foi visitada, é a ideia principal da BT, promovendo assim diversificação na busca, pois para explorar o espaço de soluções e evitar ficar preso em ótimos locais, a BT aplica qualquer procedimento de busca local intensamente. Existem duas funções principais na BT, memória adaptativa e exploração responsiva. A primeira é denominada lista tabu e armazena o histórico de ações realizadas no passado durante a busca para evitar ficar preso dentro de um ciclo. Já a exploração responsiva usa a memória adaptativa para focar o processo de busca em boas áreas e nas melhores soluções para uma maior intensificação, e explorar novas áreas promissoras para exploração. (ABDEL-BASSET; ABDEL-FATAH; SANGAIAH, 2018)

É importante destacar que, a busca tabu se trata de uma meta-heurística que se utiliza de estruturas flexíveis de memória para armazenar conhecimento sobre um espaço de busca, que se mantém durante um determinado espaço de tempo ou certo número de iterações de soluções ótimas, para auxílio de outros algoritmos. Esses algoritmos geralmente não utilizam memória acumulada de iterações anteriores, ou possuem uma estrutura de memória mais rígida (ZHANG et al., 2020).

A aplicação da busca tabu se inicia na identificação de uma boa solução e a partir de iterações, o procedimento avança para soluções melhores na vizinhança da solução já encontrada. Avanços para soluções já encontradas, que pelo procedimento já estão na lista tabu, não são permitidos. Se a conclusão de avanço na vizinhança é o não-avanço, significa que a melhor solução ótima possível, pelo menos dentro dessa vizinhança, foi identificada (MARINHO, 2020).

A busca tabu está fundamentada em três etapas. Primeiramente é utilizada uma estrutura de dados do tipo fila para guardar o histórico da evolução do processo de busca. Depois, por meio de um mecanismo de controle, é feito o balanceamento para a atualização ou não da

solução, com base nas informações registradas na lista tabu representando o conjunto de restrições e a qualidade do valor da função desejada. Por fim, incluem-se procedimentos que alteraram as estratégias de diversificação e intensificação. A busca tabu explora a área viável, e sempre toma o melhor movimento possível em qualquer instante, de forma a alternar o processo de diversificação e intensificação por meio da análise dos atributos proibidos armazenados na lista de tabus (HOU et al., 2020).

2.3.3 Algoritmos genéticos

Os algoritmos genéticos possuem sua base interativa apoiada na biologia, mais especificamente na evolução natural, sendo propostos inicialmente em 1975 por John Holland. Desde então, tornaram-se uma ferramenta popularmente aplicada em problemas de otimização nos últimos anos (WHITLEY, 2019). Os algoritmos genéticos realizam seus processos iterativos por meio de gerações, estas são formadas inicialmente no problema e submetidas a uma avaliação. Caso não sejam satisfatórios os resultados obtidos, uma nova geração é formada, a partir da atual, repetindo o processo. Pinho et al. (2013) descrevem os algoritmos genéticos como um processo evolutivo, que produz novas gerações por meio de operadores probabilísticos, de forma que este novo conjunto formado tende a ter um melhor resultado em comparação aos anteriores.

Segundo Rahmani Hosseinabadi (2019) os algoritmos genéticos possuem muitos termos próprios, como *indivíduo* sendo uma potencial solução, *população* que representa o conjunto de indivíduos, *geração* sendo uma população em um instante de tempo, no qual ao passar do mesmo sofre alteração em sua população inicial. *Pais* e *filhos*, sendo os *pais* os que passaram por reprodução e os *filhos*, dela decorrentes. *Cromossomo*, que é a estrutura de dados de um indivíduo, gene sendo uma posição única presente no cromossomo, *alelos* que são o valor de um gene específico. Por fim a *mutação* que são variações no cromossomo dos indivíduos e o *crossover* onde é realizada uma combinação no cromossomo dos indivíduos.

A geração da população pode ser obtida de forma aleatória e precisa conter alternativas viáveis para o problema explorado (HILLIER; LIEBERMAN, 2013). Segundo Mirjalili (2019), a seleção de indivíduos para reprodução é o procedimento utilizado previamente à construção da nova geração de indivíduos, sendo estes totalmente influenciados por esta etapa. O processo de seleção dos indivíduos sobreviventes que irão compor a população de uma nova geração no algoritmo genético mimetiza a seleção natural dos seres vivos e são repetidos até se obter um número de indivíduos desejado, sendo possível a realização dessa seleção de diferentes formas.

Esta pesquisa foca na seleção por torneio, em que, de forma geral, os indivíduos são selecionados aleatoriamente na população (em uma quantidade pré-definida) e têm seus cromossomos comparados entre si por meio da função de avaliação; o melhor indivíduo entre os selecionados é escolhido (MIRJALILI, 2019).

Cruzamentos e mutações ocorrem na etapa posterior à seleção. Para o cruzamento, esta pesquisa foca no uniforme, neste tipo de cruzamento dois indivíduos são selecionados e os seus respectivos genes possuem uma determinada probabilidade de ter os seus alelos trocados. Segundo Metawa, Hassan e Elhoseny (2017) essa é uma maneira justa de tratar os genes dos indivíduos, e que a probabilidade de troca dos alelos não é necessariamente fixa, podendo variar (como qualquer probabilidade) de 0 a 1. Para o caso de mutação, são realizadas mudanças aleatórias no cromossomo dos indivíduos, o que faz com os indivíduos resultantes não se diferenciem muito dos originais.

Um aspecto importante sobre o cruzamento e a mutação é a taxa atrelada a esses dois processos, para o cruzamento essa taxa determina se será feito o *crossover* entre certos indivíduos da população e para a mutação a taxa se o indivíduo sofrerá ou não a mutação, quanto maior a taxa, maiores são as chances de ocorrerem. Altas taxas de cruzamento podem gerar mais rapidamente novos indivíduos e, por consequência, perde informações da população anterior de forma mais rápida. Quanto maior a taxa de mutação, a busca pela solução ganha maior grau de aleatoriedade, sendo uma vantagem contra pontos de ótimos locais onde o algoritmo pode estagnar a sua busca precipitadamente (JAFAR-ZANJANI; INAMPUDI; MOSALLAEI, 2018).

2.3.4 *Simulated Annealing*

De acordo com Youssef (2001), a MH *simulated annealing* (SA) é derivada de um processo de recozimento de sólidos desenvolvido por Metropolis (1953). Kirkpatrick et al. (1983) introduziram uma analogia com a otimização combinatória, que foi melhorado por Cerny (1985). *Annealing* se trata de um processo de resfriamento térmico que primeiro liquefaz um cristal em alta temperatura e, em seguida, reduz lentamente sua temperatura até atingir o ponto de congelamento, momento em que o sistema atinge o estado mínimo de energia. Assim com toda a evolução, a SA pode ser usada para resolver vários problemas de otimização combinatória inclusive o *max-cut*.

Em síntese, segundo Delahaye, Chaimatanan e Mongeau (2019), SA é um algoritmo de busca local, que parte de uma solução inicial e explora iterativamente uma única nova solução

vizinha da solução corrente. A característica de maior diferenciação é a probabilidade de aceitação de movimentos de piora, esse critério é chamado de condição de metropolis, onde em problemas de maximização, por exemplo, é considerada uma variação Δ. Essa variação representa a diferença entre uma nova solução vizinha aleatória e a solução corrente, se Δ for maior que zero o método aceita a mudança, caso contrário, a solução vizinha candidata também tem a possibilidade de ser aceita, com uma probabilidade em que um valor aleatório seja menor que $e^{\Delta/T}$, onde T é um parâmetro chamado de temperatura que regula a probabilidade de se aceitar soluções de piora.

A temperatura T assume, inicialmente, um alto valor T_0. Após um número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura, essa temperatura é gradativamente diminuída por uma razão de resfriamento. Em altas temperaturas, as partículas se movem com maior liberdade fazendo com que a solução possa mudar significativamente. A temperatura diminuirá com o tempo e a probabilidade de movimento das partículas diminuirá até que o sistema atinja um estado fundamental. No caso de uma piora no valor da função objetivo, uma solução é mais provável de ser aceita quando a temperatura está alta, isto é, tipicamente no início da busca. Quando a temperatura está baixa, normalmente no final da busca, a prioridade é dada à melhoria da solução corrente (CHOPARD; TOMASSINI, 2018).

Diversos autores estudam condições para a aceitação de movimentos de piora, principalmente com foco no esquema de resfriamento, que é uma função que controla a temperatura na iteração atual. Infelizmente, essas análises geralmente provam que a convergência para o ótimo global tende ao infinito, impossibilitando assim aplicações práticas na obtenção desse melhor resultado.

Tendo fundamentado os principais temas norteadores da pesquisa, foi realizada uma revisão em relação aos estudos desenvolvidos em relação ao problema do *max-cut* e suas principais abordagens.

3 REVISÃO DA LITERATURA

Na esperança de encontrar soluções do *max-cut* de instâncias de grafos densos, foi proposta pela primeira vez por Goemans e Williamson (1995) a formulação de um algoritmo aproximativo capaz de alcançar 87,86% das vezes a solução ótima para um conjunto de instâncias do *max-cut*, com a condição restritiva de que os pesos das arestas não tenham valores negativos. Os autores utilizaram a relaxação em programação semidefinida.

Hadlock (1975) traz um método exato que pode ser resolvido em tempo polinomial para um caso especial do *max-cut*, solucionando apenas instâncias de grafos planares. É feita uma tradução do problema para o *maximum weighted matching problem que* é resolvido por inspeção. De forma que as arestas que não pertencem a um corte máximo de um grafo G são as duais de arestas que são duplicadas em uma volta de inspeção ótima do grafo dual de G.

Della, Kaminski e Paschos (2007) trabalha com uma técnica exata para o *max-cut*, aplicada a grafos com grau máximo e limitada aos grafos gerais, no algoritmo são enumerados cortes em um subgrafo *H* de *G*, realizando assim uma extensão para se obter uma forma ótima de corte em *G*, com a execuções em tempos exponenciais. Os autores Rendl, Rinaldi e Wiegele (2008) também apresentam um método exato para o *max-cut*, consistindo em uma configuração *Branch-and-Bound* que aplica uma versão dinâmica do método *bundle* como procedimento delimitador. Esta abordagem usa a dualidade Lagrangiana para obter uma solução "quase ótima" da relaxação *max-cut* semidefinida básica, reforçada por desigualdades triangulares. O método se mostrou promissor até mesmo na solução de grafos mais densos, porém para grafos com mais de 100 vértices os tempos se tornaram exponenciais.

Em 2021 Hrga e Povh trouxeram um método de solução exata paralela para ser executado em supercomputadores, baseado no método dos multiplicadores de direção alternada (ADMM), nesse método foi implementado um novo algoritmo *Branch-and-Bound* para *max-cut* que explora os limites superiores válidos advindos de um arredondamento de solução quase viável do ADMM.

Abordagens heurísticas e meta-heurísticas de tipos variados foram propostos na literatura para a obtenção de soluções do problema do *max-cut*. Uma hibridização de algoritmos genéticos com heurísticas de melhoria local foi proposta por Kim, Kim e Moon (2001). Nessa abordagem foi realizada uma aplicação em instâncias reais em circuitos de chips VLSI. Outra abordagem que utilizou algoritmos genéticos foi a implementada por Mansour, Awad e El-Fakih. (2006), que utilizou um algoritmo genético incremental para resolver o problema do

max-cut. Nenhuma dessas abordagens comparou os resultados obtidos com o estado da arte do *max-cut*, tendo comparações que se limitam às diferentes versões propostas pelos autores em cada trabalho. Outra abordagem foi a de Wang e Wang (2010) que aplicou algoritmos genéticos ao problema do *max-cut fuzzy*.

Nas abordagens meta-heurísticas, Festa et al. (2001) utilizaram métodos de busca local baseada na GRASP e, no ano seguinte, desenvolveram uma abordagem de VNS e VND (FESTA et al., 2002). Esses métodos atingiram resultados em um tempo computacional relativamente pequeno, mas esses resultados não alcançaram resultados tão bons quanto os obtidos por Goemans e Williamson (1995) com uma abordagem centrada na utilização de relaxação em programação semidefinida.

Algoritmos meta-heurísticos evolucionários meméticos foram apresentados por Wu e Hao (2012), atingindo resultados que comparados com os melhores algoritmos propostos se mostrou competitivo na busca por soluções ótimas para o *max-cut*. Em Kochenberger et al. (2013) foi implementado um novo algoritmo meta-heurístico de busca tabu, que obteve bons resultados em instâncias com grande número de vértices. Ambos os métodos forneceram algumas das melhores soluções conhecidas para instâncias de teste utilizadas na literatura.

Mohades e Kahaei (2021) propôs um algoritmo aproximativo de otimização Riemanniano rápido e preciso para resolver o problema do *max-cut*. Para isso, foi implementado um algoritmo de descida de gradiente, retornando resultados de simulação que mostraram que o método proposto é muito eficiente em certos grafos estudados na literatura, em média 35 vezes mais rápida que as técnicas mais conhecidas, com uma pequena queda nos resultados, em média 0,96 no valor de corte máximo para as demais técnicas.

Uma revisão sistemática dos algoritmos heurísticos e meta-heurísticos propostos na literatura para solução do *max-cut* foi realizada por Dunning et al. (2018). Um total de 37 algoritmos heurísticos foram replicados, sendo testados em mais de 2.000 instâncias de diferentes tamanhos e densidades. O desempenho do algoritmo foi utilizado como dados de entrada para um modelo de *machine learning*. Os resultados mostraram que algoritmos genéticos apresentam melhor desempenho em instâncias densas de médio porte, enquanto algoritmos não evolutivos apresentam melhor desempenho em instâncias de grande porte.

Com base na análise de trabalhos anteriores, os métodos meta-heurísticos mostraram serem amplamente utilizados para a obtenção de soluções para o *max-cut*, com métodos híbridos e de programação semidefinida encontrando bons resultados aproximativos, porém, a

literatura é escassa em abordagens exatas para a solução do problema. Della, Kaminski e Paschos (2007) chegaram a propor um método que consistia em enumerar cortes em um subgrafo e então estendê-los de forma ótima aos cortes. A técnica foi aplicada a grafos de grau máximo limitado e a grafos gerais, porém, em ambos os casos, o algoritmo obteve tempos exponenciais. Assim esta pesquisa inova ao desenvolver um algoritmo exato, com uma abordagem totalmente inovadora, fundamentada em árvores de busca com a solução de sistemas de equações lineares binárias, com os procedimentos metodológicos apresentados a seguir.

4 METODOLOGIA

4.1 CLASSIFICAÇÃO DA PESQUISA

Quanto aos procedimentos metodológicos desta pesquisa, em relação à abordagem foi classificada como quantitativa, que segundo Lakatos e Marconi (2017) é a pesquisa que segue um método científico que utiliza diferentes técnicas estatísticas para quantificar e catalogar os dados. Assim esta pesquisa realizou experimentos com uma coleta de dados rígida e objetiva, com análises a partir de quantificações, através de ferramentas estatísticas, apresentando os resultados, a partir de uma estrutura de tabelas e gráficos. Quanto à natureza a pesquisa foi classificada como básica, que segundo Gil (2019) tem por objetivo gerar conhecimento novo para o avanço da ciência, buscando gerar verdades, ainda que temporárias e relativas, de interesses mais amplos.

Quanto aos objetivos foi classificada como uma pesquisa axiomática e normativa. A pesquisa foi axiomática pois foi baseada no problema do *max-cut* já idealizado na literatura, e normativa pois apresentada uma nova abordagem matemática exata de otimização para o *max-cut*, que prescreve soluções ao problema (BERTRAND; FRANSOO, 2002). Quanto aos procedimentos foi uma pesquisa bibliográfica e experimental. A pesquisa bibliográfica é um trabalho de natureza exploratória, que propicia bases teóricas para auxiliar no exercício reflexivo e crítico sobre o tema em estudo (GIL, 2019). Já a experimental segundo Lüdke e André (2013), objetiva identificar as variáveis que influenciam o objeto e definir as formas de controle e de observação dos efeitos que a variável produz no objeto.

Esta pesquisa ainda por trazer algoritmos foi de um tipo de pesquisa em computação que visa à apresentação de algo presumivelmente melhor, que segundo Wazlawick (2020) objetiva ao desenvolvimento de um método novo ou incremental, realizando um extenso trabalho de comparação, definindo bem o método usado para implementar e realizar os experimentos.

4.2 DESCRIÇÃO DO PROCEDIMENTO

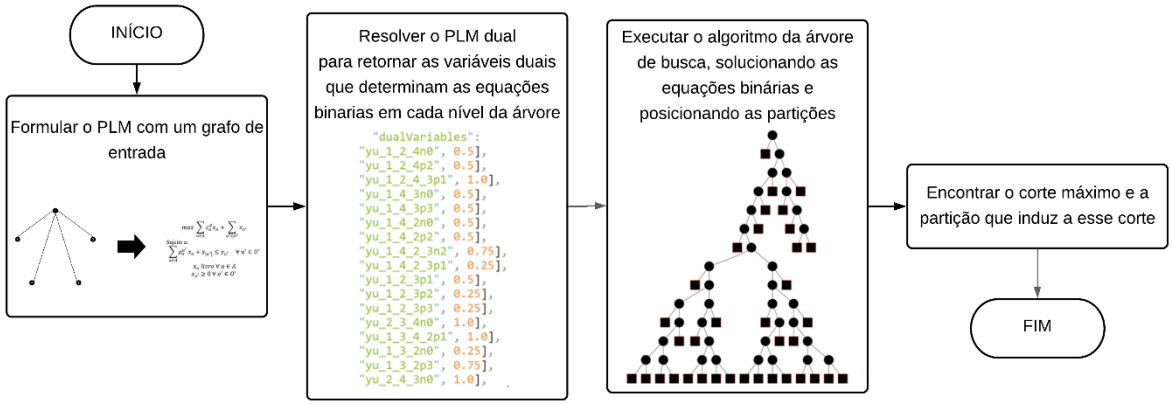
Esta pesquisa foi dividida em quatro fases, na primeira fase foi realizado o desenvolvimento e implementação da abordagem exata para a solução do problema. Na segunda fase foram implementadas cinco meta-heurísticas, sendo quatro de busca local e uma de busca populacional. Na terceira fase, foram realizados os experimentos computacionais com diferentes tipos de instâncias de grafos. Por fim, na quarta fase, foram realizadas as análises

estatísticas. Esse procedimento segue a abordagem proposta por Lima, Melo e Lins (2021), com avanços na descrição da abordagem exata, implementação de mais quatro meta-heurísticas e dois experimentos mais robustos.

4.2.1 Fase 1: abordagem exata

O algoritmo exato proposto abordou o problema simplificado do *max-cut*, o conhecido *Simple Max-Cut Problem* (SMCP). O SMCP foi solucionado através da redução do problema para um tipo muito especial do *Subset Sum Problem* (SSP), isto permitiu a utilização de árvores de busca binária para resolver a instância do SSP com regras de remoção dos nós, de modo que o número de nós na árvore tendeu a ser reduzido. O algoritmo final decide se há ou não um corte de determinado tamanho nas arestas de um grafo, particionando seus vértices em dois subconjuntos. Esta abordagem foi fundamentada nas quatro etapas ilustradas na Figura 3.

Figura 3 – Fluxograma com as etapas da abordagem exata



Fonte: O autor (2022)

Na primeira etapa tendo como instância um grafo, é formulado um Programa Linear Mestre (PLM), que utiliza um conjunto de Unificadores do Caminho Diagonal *O*, obtido por meio de uma técnica de mapeamento de grafos. Na segunda etapa, o PLM é resolvido por meio de um *software* de programação linear retornando as variáveis duais que são, juntamente com o grafo, a base para o algoritmo da árvore de busca. Na terceira etapa, o algoritmo da árvore é finalmente executado, particionando os vértices do grafo em dois subconjuntos diferentes, até que, por fim, na última etapa, o algoritmo retorna o *max-cut* do grafo e os dois subconjuntos que induzem esse corte.

Para melhor compreensão da abordagem, a Tabela 1 apresenta os principais termos utilizados nesta abordagem e as suas definições.

Tabela 1 – Definição dos principais termos da abordagem exata

Termo	Definição
ORG_h	Um grafo plano, chamado de grafo orbital radial, que se destaca por ter suas arestas com formato de órbitas e raios, no seu interior, um conjunto de dígonos que formam uma *flor* central. O tamanho desse grafo é definido por meio de um parâmetro *h*, que são números inteiros h = 1, ..., n. Assim o ORG_h é capaz de representar qualquer conjunto de arestas de grafos, como um subconjunto de arestas, com a especificidade de as arestas serem dobradas: uma aresta (*i, j*) aparece como duas arestas distintas, nas formas de (*i, j*) e (*j, i*).
Dígono	Duas arestas no meio do grafo orbital radial, com formato de uma pétala, onde um arco representa uma aresta (*i, j*) e o outro arco representa uma aresta (*j, i*). O número de dígonos num ORG_h é 4h. As arestas dos dígonos são rotuladas com as combinações dos vértices 1, ..., 4h com o vértice 4h + 1.
Flor	A junção de todas as pétalas de arestas dobradas no centro do ORG_h, sendo assim o conjunto de todos os dígonos, com o formato de uma flor no centro do ORG_h.
Unificadores do caminho diagonal	Uma marcação especial que representa os passos dentro do ORG_h. Alternam-se entre um vértice e a diagonal de uma face. O caminho inicia num vértice na órbita exterior e adentra pela diagonal de uma face, até chegar ao vértice da órbita central, onde passa para o próximo vértice da órbita central girando no sentido anti-horário, realizando assim o caminho de retorno para a órbita exterior. Os unificadores que são vértices são nomeados por (*i, j, k*), que representa as três arestas (*i, j*), (*j, k*), (*k, i*) ao redor do vértice. Já as faces são nomeadas por (*i, j, k, l*), que representa as quatro arestas (*i, j*), (*j, k*), (*k, l*) e (*l, i*) que formam a face. O caminho diagonal num ORG_h é composto por 2h percursos de adentramento orbital, que é uma saída da órbita externa, visita à órbita interna e retorno à externa.
Árvore de Busca Binária	Um algoritmo de busca que utiliza a estruturação de dados em árvore. Utiliza quatro sementes, com raízes diferentes, que tem como prioridade ramificar-se criando filhos na esquerda, sendo explorados filhos na direita quando um filho na esquerda tem alguma contradição que o transforma numa folha. Os filhos carregam as mesmas informações dos pais. Cada nível da árvore tem uma inequação referente às variáveis da solução dual, sendo assim calculada uma equação linear binária que é uma adaptação da inequação dual. Para os filhos da esquerda, a desigualdade é transformada em igualdade com o valor de referência γ_u subtraído de 2, já os filhos da direita têm apenas o sinal da desigualdade transformado em igualdade. Se a solução de uma variável de aresta na solução da equação tiver valor 1, é posicionado os vértices em partições opostas, se for igual a 0 os vértices são posicionados na mesma partição. No final todas as árvores de busca são orladas por nós contraditórios ou folhas.

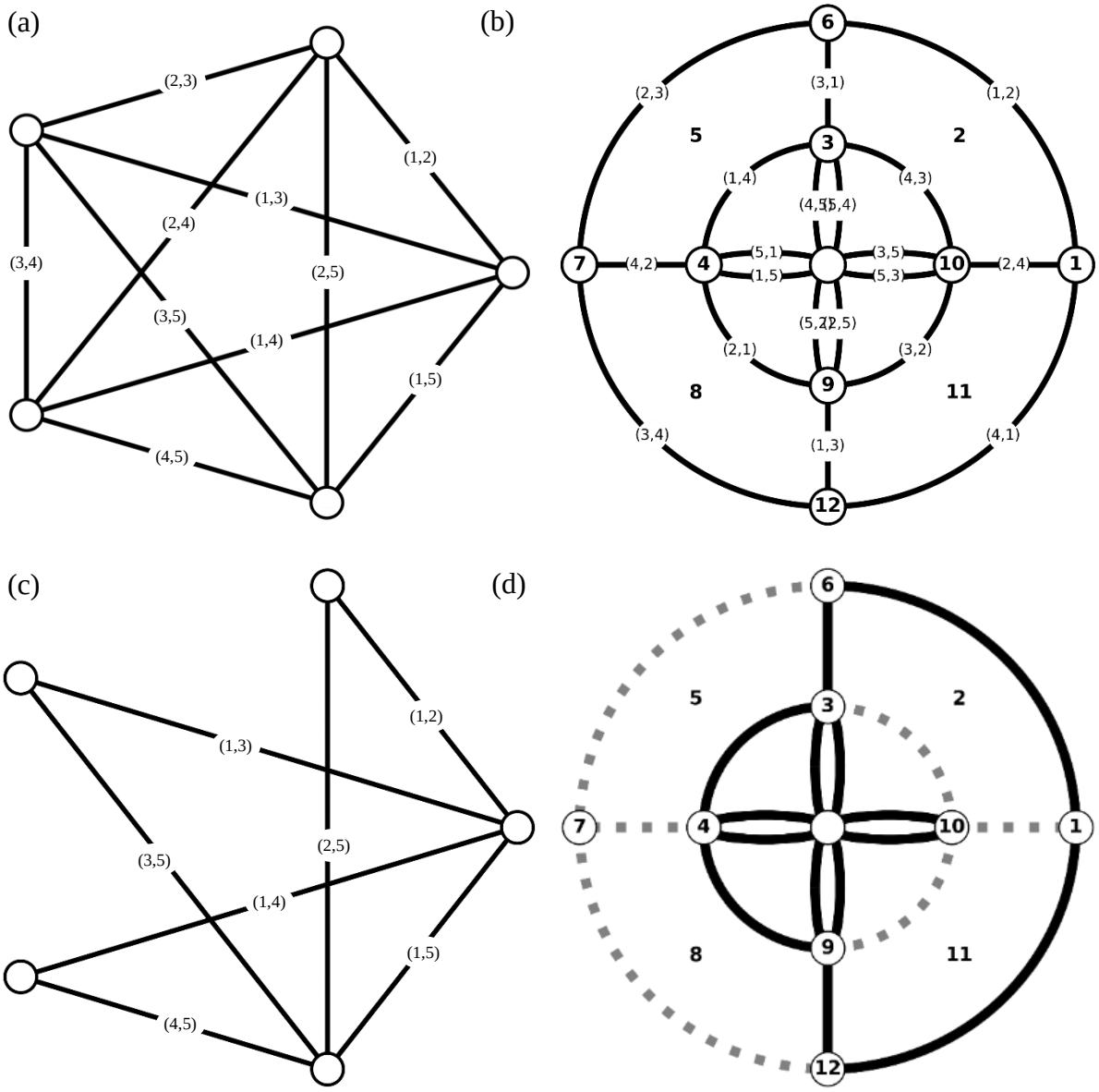
Fonte: O autor (2022)

A partir dessas definições, é possível aprofundar a apresentação dos métodos que compõem a abordagem exata proposta nesta pesquisa.

4.2.1.1 Mapa de grafos

A característica fundamental deste método foi a criação de grafos planares ORG_h, estes grafos estão intimamente relacionados aos grafos completos K_{4h+1} com o conjunto de arestas dobrado, uma aresta (i, j) aparece como duas arestas distintas, nas formas de: (i, j) e (j, i). ORG_h, para números inteiros h = 1, ..., n, possui $8h^2 + 1$ vértices e $16h^2 + 4h$ arestas. O grafo é plano e existem $2h$ órbitas, cada uma contendo $4h$ vértices. Existe um vértice central extra de grau $8h$ e $4h$ dígonos ao redor do vértice central, o que forma a $flor$ do ORG_h. A Figura 4 a seguir mostra a representação do ORG_h com h = 1 e o grafo completo K_5, com um grafo de 5 vértices mapeado no ORG_1.

Figura 4 - (a) grafo K_5 , (b) ORG_1, (c) grafo aleatório com 5 vértices, (d) grafo mapeado.



Fonte: O autor (2022)

O menor ORG_h, é o com $h = 1$. A Figura 4b apresenta o grafo denominado $K_5^{(2)}$, ou seja, K_5 com cada aresta dobrada. Assim, grafos podem ser representados dentro do conjunto de arestas deste ORG_h. Um ponto importante é que a identificação numérica de vértices, faces e arestas pode ser implicitamente dada apenas por *h*, pois o caminho diagonal alterna-se essencialmente como um contorno de um vértice e o limite de uma face, então assim é possível determinar o primeiro percurso de adentramento orbital completo, e a partir deste é possível gerar todos os outros, formando o caminho diagonal completo. A lógica para a formação do primeiro percurso de adentramento orbital segue no Algoritmo 1.

Algoritmo 1 – Primeiro Percurso de Adentramento Orbital

```
1: def gerarPrimeiroPercurso (h):
2:     primeiro   = [1,2,4*h]
3:     semente    = primeiro + [3]
4:     primeiro percurso  = [primeiro,semente]
5:     contador = 0
6:     for t in range(4*h-4):
7:         nova entrada = [t for t in primeiro percurso [-1]]
8:         if contador é par:
9:             nova entrada[1] = (nova entrada[1] + 2)
10:        else:
11:            nova entrada[3] = (nova entrada[3] + 2)
12:        primeiro percurso.append(nova entrada)
13:        contador+=1
14:    meio1 = [1,4*h,4*h-1]
15:    primeiro percurso.append(meio1)
16:    meio2    = [1,4*h,2]
17:    primeiro percurso.append(meio2)
18:    semente    = meio2 + [4*h-1]
19:    primeiro percurso.append(semente)
20:    contador = 0
21:    for t in range(4*h-4):
22:        nova entrada = [t for t in primeiro percurso [-1]]
23:        if contador é par:
24:            nova entrada[1] = (nova entrada[1] - 2)
25:        else:
26:            nova entrada[3] = (nova entrada[3] - 2)
27:        primeiro percurso.append(nova entrada)
28:        contador+=1
29:    último = [1,2,3]
30:    primeiro percurso.append(último)
31:    return primeiro percurso
```

O primeiro passo do caminho diagonal sempre é uma tripla formada por [1, 2, 4h], no segundo passo é iniciado uma sequência de quadras que são calculadas de acordo com uma

semente que é formada com a adição de um novo elemento 3 no primeiro passo. Com essa semente é iniciado um contador igual a 0, em um laço de repetição de 4h – 4 vezes. Se o contador for par, uma nova entrada que é uma cópia do último passo inserido na lista do primeiro percurso é adicionadas duas unidades no segundo elemento; se não, são adicionadas duas unidades no quarto elemento. Quando chega à metade do primeiro percurso, o primeiro unificador do meio é [1, 4h, 4h-1] e o segundo é [1, 4h, 2]. Assim, é iniciado uma nova série de quadras tomando-se por base uma semente [1, 4h, 2, 4h-1]. Entra-se então num laço de repetição igual ao primeiro, com a exceção de subtrai-se duas unidades do segundo ou do quarto elemento ao invés de se adicionar. Assim todo primeiro percurso é finalizado por [1, 2, 3]. Para formar todo o conjunto de unificadores do caminho diagonal, o número de percursos de adentramento orbital num ORG_h é 2h percursos, assim a formação a partir do segundo percurso do caminho diagonal, cada novo passo do percurso é igual ao passo do percurso anterior somando-se duas unidades a cada elemento, e cada um desses passos precisa ser normalizado para iniciar sempre com o que tiver o menor elemento. Um caminho diagonal completo é apresentado na Figura 5. A ordem obtida é de suma importância pois elas são a base para a formulação do programa linear.

4.2.1.2 Programa linear mestre

Para cada ORG_h será definido um programa linear chamado de Programa Linear Mestre (PLM). Na formulação matemática a seguir é apresentado o modelo primal do PLM, sendo cada variável associada a uma aresta do grafo representada por x_a.

$$max \sum_{a \in A} x_a + \sum_{o \in O} z_o \tag{4.1}$$

Sujeito a:

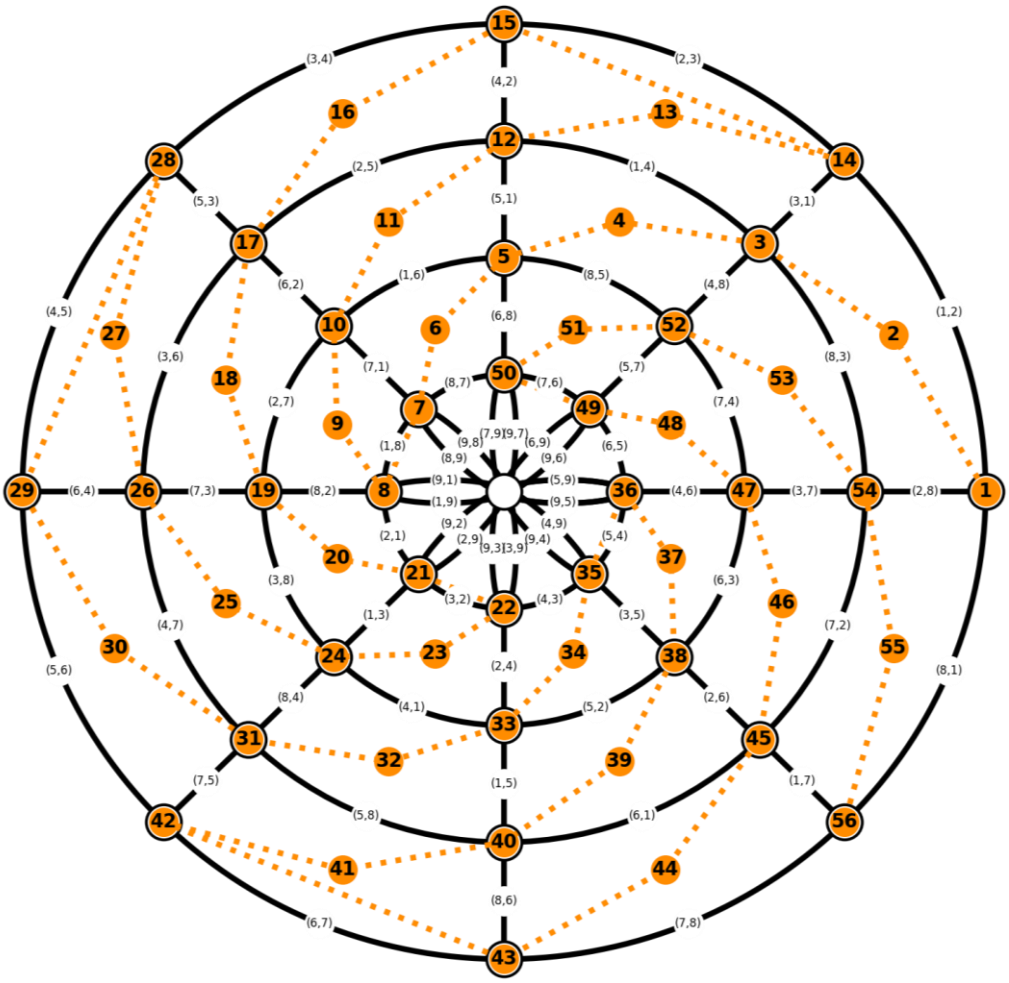
$$\sum_{a \in u} \rho_a^u \, x_a + z_{|u|} \leq \gamma_u \quad \forall \, u \in U \tag{4.2}$$

$$x_a \; livre \; \forall \; a \in A$$
$$z_o \geq 0 \; \forall o \in O \tag{4.3}$$

A função objetivo (4.1) é uma maximização do somatório de todas as variáveis das arestas no grafo mais o somatório das variáveis de apoio que recebem sua enumeração pelos unificadores do caminho diagonal *O* do ORG_h, intitulada z_o. A restrição (4.2) é a que retorna as variáveis duais y_u, sendo que neste PLM, o conjunto *O* é assinado de uma maneira especial, em que eles correspondem às variáveis duais. O número de restrições no PLM é dado pela expressão: $128h^2$ - 40h, correspondendo ao conjunto *O* sinalizado, formando assim um novo

conjunto U, onde para cada $o \in O$, se o for uma tripla então são sinalizados de quatro formas distintas, $n0, p1, p2\ e\ p3$. Se o for uma quadra então são sinalizados de oito formas distintas $n1, n2, n3, n4, p1, p2, p3\ e\ p4$. Para ilustração do conjunto O, o ORG_2 com quatro percursos de adentramento orbital, que juntos constituem os 56 unificadores do conjunto O, são representados na Figura 5.

Figura 5 – Representação dos unificadores do caminho diagonal do ORG_2



1 : 128	15 : 342 ↦ 234	29 : 456 ↦ 456	43 : 678 ↦ 678
2 : 1283	16 : 3425 ↦ 2534	30 : 4756 ↦ 4756	44 : 6178 ↦ 1786
3 : 1483	17 : 3625 ↦ 2536	31 : 4758 ↦ 4758	45 : 6172 ↦ 1726
4 : 1485	18 : 3627 ↦ 2736	32 : 4158 ↦ 1584	46 : 3726 ↦ 2637
5 : 1685	19 : 3827 ↦ 2738	33 : 4152 ↦ 1524	47 : 3746 ↦ 3746
6 : 1687	20 : 1382 ↦ 1382	34 : 3524 ↦ 2435	48 : 4657 ↦ 4657
7 : 187	21 : 321 ↦ 132	35 : 354 ↦ 354	49 : 576 ↦ 576
8 : 182	22 : 324 ↦ 243	36 : 465 ↦ 465	50 : 687 ↦ 687
9 : 1827	23 : 3241 ↦ 1324	37 : 3546 ↦ 3546	51 : 5768 ↦ 5768
10 : 1627	24 : 3841 ↦ 1384	38 : 3526 ↦ 2635	52 : 4857 ↦ 4857
11 : 1625	25 : 3847 ↦ 3847	39 : 5261 ↦ 1526	53 : 3748 ↦ 3748
12 : 1425	26 : 3647 ↦ 3647	40 : 5861 ↦ 1586	54 : 3728 ↦ 2837
13 : 1423	27 : 3645 ↦ 3645	41 : 5867 ↦ 5867	55 : 7281 ↦ 1728
14 : 123	28 : 345 ↦ 345	42 : 567 ↦ 567	56 : 781 ↦ 178

Fonte: O autor (2022)

Para os unificadores do caminho diagonal que são triplas (*i, j, k*) no conjunto de inequações (4.4), sempre haverá quatro restrições, onde y_{ijk} intitula o nome da restrição seguida pela sinalização $n0$, onde não há nenhum sinal negativo na operação entre as variáveis e pelas sinalizações $p1, p2\ e\ p3$ representando a posição do sinal de soma na operação, as variáveis são $x_{ij}, x_{jk}, x_{ki}\ e\ z_o$, a variável ρ_a^u assume valor de 1 ou -1, a depender do sinal da variável da aresta em u, o valor de γ_u nas restrições com sinalizações *n* tem o valor 2 na desigualdade e as com sinalização *p* o valor de γ_u é igual a 0. Por exemplo $y_{124}p1{:}\ +\ x_{12} - x_{24} - x_{41} + z_{124} \leq 0$. Os que são quadras (*i, j, k, l*) no conjunto de inequações (4.5), são oito restrições intituladas y_{ijkl}, as sinalizações são $n1, n2, n3, n4, p1, p2, p3\ e\ p4$, por exemplo $y_{1243}n3{:}\ +\ x_{12} + x_{24} - x_{43} + x_{31} + z_{1243} \leq 2$. A variável $z_{|u|}$ é uma variável em que a sinalização de *u* é retirada considerando assim apenas *o*.

$$\begin{aligned} y_{ijk}n0 &\rightarrow +x_{ij} + x_{jk} + x_{ki} + z_{ijk} \leq 2 \\ y_{ijk}p1 &\rightarrow +x_{ij} - x_{jk} - x_{ki} + z_{ijk} \leq 0 \\ y_{ijk}p2 &\rightarrow -x_{ij} + x_{jk} - x_{ki} + z_{ijk} \leq 0 \\ y_{ijk}p3 &\rightarrow -x_{ij} - x_{jk} + x_{ki} + z_{ijk} \leq 0 \end{aligned} \tag{4.4}$$

$$\begin{aligned} y_{ijkl}n1 &\rightarrow -x_{ij} + x_{jk} + x_{kl} + x_{li} + z_{ijkl} \leq 2 \\ y_{ijkl}n2 &\rightarrow +x_{ij} - x_{jk} + x_{kl} + x_{li} + z_{ijkl} \leq 2 \\ y_{ijkl}n3 &\rightarrow +x_{ij} + x_{jk} - x_{kl} + x_{li} + z_{ijkl} \leq 2 \\ y_{ijkl}n4 &\rightarrow +x_{ij} + x_{jk} + x_{kl} - x_{li} + z_{ijkl} \leq 2 \\ y_{ijkl}p1 &\rightarrow +x_{ij} - x_{jk} - x_{kl} - x_{li} + z_{ijkl} \leq 0 \\ y_{ijkl}p2 &\rightarrow -x_{ij} + x_{jk} - x_{kl} - x_{li} + z_{ijkl} \leq 0 \\ y_{ijkl}p3 &\rightarrow -x_{ij} - x_{jk} + x_{kl} - x_{li} + z_{ijkl} \leq 0 \\ y_{ijkl}p4 &\rightarrow -x_{ij} - x_{jk} - x_{kl} + x_{li} + z_{ijkl} \leq 0 \end{aligned} \tag{4.5}$$

Este programa linear mestre primal foi executado e resolvido no *software* livre SCIP *Optimization Suite* 7.0 (GAMRATH et al., 2021), mais especificamente utilizando o SoPlex (*Linear Programming Solver*). A escolha deste software está apoiada na sua eficiência e na sua possibilidade de resolver o dual a partir do programa linear primal, onde as variáveis duais y_u são encontradas. As variáveis duais são a base elementar para a árvore de busca binária, pois a cada nível em que se desce na árvore é um elemento *u* no conjunto de caminhos diagonais sinalizados *U*, representado pelas variáveis y_u e suas inequações correspondentes, cada inequação é transformada em uma equação linear para ser resolvida dentro de cada nó da árvore.

4.2.1.3 Árvore de busca binária

A formação dessa Árvore de Busca Binária (ABB) constitui em um grafo não direcionado em que os nós possuem informações acerca do problema, sendo a principal uma potencial solução para o problema do *max-cut* corrente (não necessariamente completa). Assim a solução é representada por dois conjuntos de vértices disjuntos de um grafo qualquer de entrada, podendo ter uma solução incompleta, ou seja, não contendo todos os vértices na bipartição, uma vez que o processo de busca pela árvore começa por um nó raiz que contém uma bipartição da solução com três vértices inseridos entre dois conjuntos, sendo essa bipartição de vértices representada por "[[ ], [ ]]". O pseudocódigo que sintetiza o novo algoritmo proposto está no Algoritmo 2 e três exemplos de execução do algoritmo são apresentados na Subseção 5.1.

Algoritmo 2 – Árvore de Busca Binária

```
Grafo = G, variáveis duais = lista de variáveis, ν^ω = Número arestas do Grafo
ℵ = [ [[1, 2, 4h], [ ]], [[1, 2], [4h]], [[1, 4h], [2]], [[1], [2, 4h]] ]
while ω_corr < ν^ω:
    for i in ℵ:
        Inicialize a Γ
        Partição = ℵ_i
        Crie Θ_esq nível inferior
        Descer um nível para Θ_esq
        Resolver a equação e calcular N
        Θ_corr = Θ_esq
        while houver Θ para percorrer:
            if Solução da equação Θ_corr não é contraditória e N ≤ N_máx
                Crie Θ_esq
                Resolver a equação, calcular N e se possível posicionar vértice na partição
                Θ_corr= Θ_esq
            else:
                Suba um nível
                Crie Θ_dir
                Resolver a equação, calcular N e se possível posicionar vértice na partição
                Θ_corr = Θ_dir
            if o número de vértices na partição = 4h:
                Crie Θ_esq com o vértice 4h+1 na partição da esquerda
                Crie Θ_dir com o vértice 4h+1 na partição da direita
                if N do Θ_corr > N_máx:
                    pass
                else:
                    if o ω_corr = ν^ω:
                        return (Corte Máximo, Partição)
    if ω_corr ≠ ν^ω
    ν^ω = ν^ω − 1
```

As bipartições iniciais foram nomeadas de sementes ($\aleph$), as árvores de busca para qualquer problema do *max-cut* são obtidas pela execução sequencial do algoritmo para cada partição inicial considerada, utilizando-se 4 sementes fixas, essas sementes representam todos os possíveis modos de iniciar uma bipartição, sendo elas: [[1, 2, 4h], [ ]], [[1, 2], [4h]], [[1, 4h], [2]] e [[1], [2, 4h]]. As partições são preenchidas à medida que o algoritmo realiza a descida na árvore, a cada nível um vértice é considerado para ser inserido ou não, dessa forma a solução é obtida iteração a iteração e avaliada de acordo com o critério de busca.

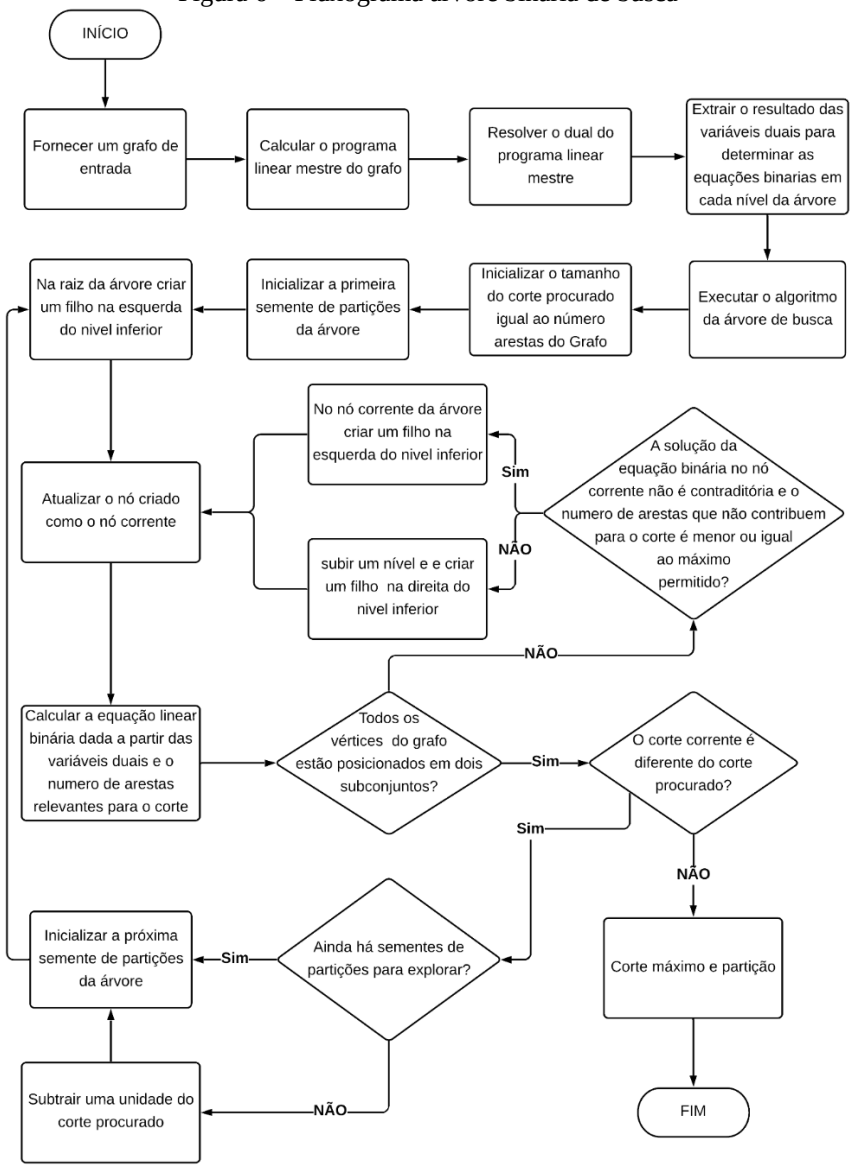
O nó inicial da ABB é denominado de nó pai ou raiz (Γ), que pode ser ramificado em outros dois nós filhos, um nó à esquerda (Θ_{esq}) e um nó à direita (Θ_{dir}), que recebem a partição do pai (Θ_{corr}). Isto permite que o processo de busca seja sistêmico, visitando primeiramente o nó filho à esquerda (um nível abaixo), este nó contém uma equação, advinda das inequações das variáveis duais referente a cada nível da árvore, a ser atendida pela nova bipartição. Não havendo dados suficientes para satisfazer a equação o algoritmo irá visitar o nó filho à esquerda do nó corrente, este novo nó terá a sua própria equação, e carregará informações das variáveis não descobertas na tentativa de solução de equação binária do nó pai, buscando assim mais informações até que seja possível resolver ou não as equações armazenadas.

Caso não seja possível encontrar uma solução factível para as equações, o nó é considerado como um nó contraditório (representado de forma quadrada na árvore), portanto uma solução não interessante para o problema. Ocorrendo uma contradição, o algoritmo retorna para o nó pai e visita desta vez o filho à direita, considerando as equações armazenadas até o nó pai. Não sendo possível atender às equações, o algoritmo retorna para o nó viável mais próximo que ainda não visitou o seu filho à direita. Não sendo o nó inicial (semente) o algoritmo continua a sua procura, caso contrário o nó com o corte máximo desejado não pode ser encontrado nesta árvore. Sendo possível atender às equações com a partição do nó filho à direita, o algoritmo retorna a sua abordagem de busca original, explorando primeiramente os filhos da esquerda.

Outra contradição que pode inviabilizar a busca no nó depende do tamanho do corte procurado v^{ω}, caso a solução preliminar da partição não consiga cortar um número mínimo de arestas necessárias para obter a solução desejada, o nó passa a ser contraditório e a busca continua a partir de outro nó disponível em níveis acima. A quantidade de arestas que a solução corrente não pode passar é armazenado no parâmetro *N*, de número de arestas irrelevantes para o corte. Cada busca terá um $N_{máx}$ que é uma contagem do número máximo de arestas que

podem não contribuir com o corte de acordo com o v^{ω}, se este for igual a quantidade de arestas do grafo o $N_{máx}$ é igual a 0, sendo assim um limite para determinar se a potencial solução no nó é desejada ou não. Todo o procedimento da ABB pode ser melhor entendido na Figura 6.

Figura 6 – Fluxograma árvore binária de busca



Fonte: O autor (2022)

No processo de busca, alguns nós, a depender do nível considerado, terão uma potencial solução do problema de corte máximo em que resta posicionar um vértice em um dos conjuntos. Esses nós são denominados de nós terminais, pois a procura pode ser encerrada neles, isso ocorre quando o número de vértices na partição for igual a $4h$, restando assim posicionar apenas o último vértice. Os nós terminais são analisados e se $\omega_{corr} = v^{\omega}$, ou seja, quando o corte corrente no nó da árvore for igual ao corte procurado na iteração atual, o algoritmo para e retorna essa solução como a solução ótima do problema, caso contrário o nó não gera mais filhos e a busca reinicia de um nó viável em um nível superior que ainda não visitou o seu filho a direita.

O algoritmo sempre irá procurar nas árvores inicialmente um corte que seja capaz de passar por todas as arestas do grafo. Não sendo possível satisfazer esse objetivo, busca-se um corte que consegue passar por todas as arestas menos um, e assim por diante, até se encontrar um nó que contenha o corte máximo desejado, denominado de nó pentagonal.

4.2.2 Fase 2: meta-heurísticas

Afim de buscar a solução do problema do *max-cut* com métodos heurísticos, que diferentemente de abordagens exatas não encontram soluções ótimas em casos gerais, foram implementadas cinco meta-heurísticas para a solução do problema, primeiramente duas meta-heurísticas de busca local hibridas, GRASP-VND e GRASP-VNS, seguidas pela busca tabu, algoritmos genéticos e pôr fim *Simulated Annealing*, para a definição dos parâmetros das MH foi realizado um *tuning*, onde diferentes combinações dos parâmetros foram testados em três instâncias, com 5, 15 e 30 vértices e as combinações com a melhor relação entre tamanho do corte e tempo foram escolhidas.

4.2.2.1 GRASP

Foi implementada uma *Greedy Randomized Adaptive Search Procedure* (GRASP). Esta meta-heurística utilizou na sua etapa de construção a clássica Lista de Candidatos Restrita (LCR) e um parâmetro alpha que controlou a abordagem gulosa e aleatória do processo. O Algoritmo 3 apresenta a função GRASP Construtiva tendo como entrada o parâmetro alpha, um grafo com as listas dos vértices e arestas e o tamanho da partição. Assim é inicializada na Linha 2 uma lista vazia para a formação da partição.

Algoritmo 3 – GRASP Construtiva

```
1 : def GraspConstrutiva( alpha, G(V,A), tamPartição ):
2 :     partição = [ ]
3 :     for i in range( tamPartição ):
4 :         candidatos = [x for x in Vértices if x not in partição]
5 :         LCR = [ ]
6 :         VérticesOrdenados = CacularVérticesOrdenados( candidatos )
7 :         arestasMin = VérticesOrdenados[0][0]
8 :         arestasMax = VérticesOrdenados[-1][0]
9 :         for i in range( len( VérticesOrdenados ) ):
10:             if( VérticesOrdenados[i][0] ≥ ( arestasMin + alpha * ( arestasMax – arestasMin ) ) ):
11:                 LCR.append(VérticesOrdenados[i])
12:         vértice = escolhaAleatória(LCR)
13:         partição.append(vértice)
14:     return partição
```

A lista de candidatos é formada por cada vértice na lista de vértices que ainda não está na lista da partição, uma LCR vazia, uma lista de vértices ordenados, que calcula para cada um dos vértices na lista de candidatados a quantidade de arestas influenciadas pelo vértice, sendo colocado em ordem crescente do vértice que menos influencia nas arestas até o vértice que está presente na maioria das arestas.

Assim arestasMin é a quantidade de arestas influenciadas pelo primeiro vértice da lista de vértices ordenados e arestasMax a quantidade de arestas influenciadas pelo último vértice da mesma lista. Para avaliar os vértices que serão adicionados na LCR, para cada vértice na lista de vértices ordenados é averiguada uma condição em que se o valor da quantidade de arestas influenciadas é maior ou igual a arestasMin + alpha * (arestasMax – arestasMin), para todos os vértices que atendem essa condição são adicionados nessa lista restrita, da qual é selecionado aleatoriamente um vértice que é adicionado na partição.

Todo esse procedimento é repetido até que a quantidade de iterações seja igual ao tamanho da partição pré-definida na entrada do algoritmo. Nessa pesquisa há uma hibridização da GRASP com dois métodos de busca local, a VND e a VNS que serão apresentadas a seguir.

4.2.2.1.1 GRASP-VND

A primeira MH híbrida implementada com a GRASP utiliza o método *Variable Neighborhood Descent* (VND) na etapa de busca local. Para a formação da solução inicial o alpha de melhor adaptação para os experimentos nesta pesquisa foi de 0,5, sendo escolhido como parâmetro de tamanho da partição um valor aleatório entre a quantidade de vértices do grafo. A partição denominada de partEsq é gerada a partir da função GraspConstrutiva, apresentada anteriormente, e a partição denominada de partDir é a diferença simétrica entre os vértices do grafo e a partEsq.

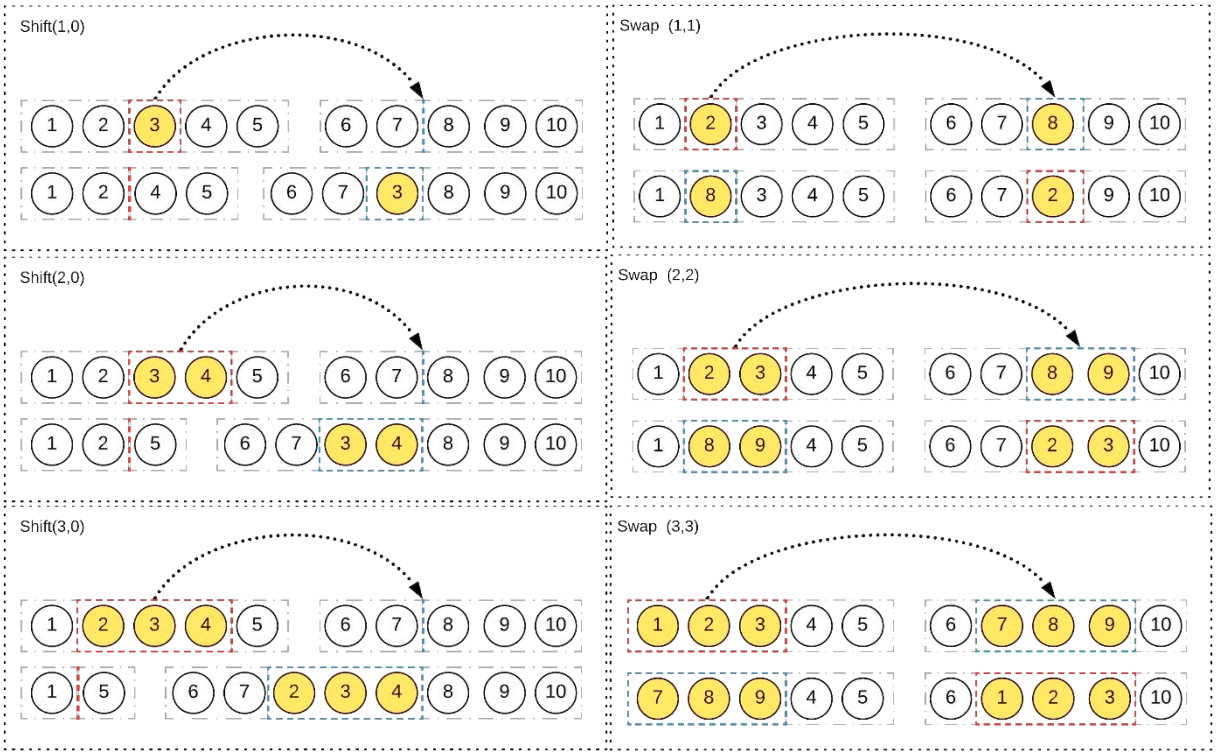
A utilização do VND torna o algoritmo capaz de explorar diferentes espaços de soluções com a utilização sequencial de operadores de vizinhança, que buscam realizar trocas entre os conjuntos de nós presentes na solução corrente. Seis operadores foram utilizados, três que consistem em trocar até *n* vértices entre dois subconjuntos, enquanto as outras três consistem em trocar dois segmentos entre subconjuntos diferentes. A seguir é apresentado o pseudocódigo da meta-heurística híbrida GRASP-VND no Algoritmo 4, sendo detalhado iterativamente cada passo a ser executado.

Algoritmo 4 – Meta-heurística GRASP-VND

```
1 : def MH_GRASP_VND( G(V,A) ):
2 :     alpha = 0.5
3 :     tamPartição = valorAleatório(QuantidadeDeVértices)
4 :     partEsq = GraspConstrutiva(alpha, tamPartição)
5 :     partDir = diferençaSimétrica(Vértices, partEsq)
6 :     soluçãoInicial = [partEsq, partDir]
7 :     k = 0
8 :     solução = solucaoInicial
9 :     vizinhanças = [ Shift1, Shift2, Shift3, Swapp11, Swapp22, Swapp33]
10:     while k < len(vizinhanças):
11:         Vizinhos = vizinhancas[k](solução)
12:         vizinhoCorrente = bestImprovement(Vizinhos)
13:         if tamanhoCorte(vizinhoCorrente) > tamanhoCorte(solução):
14:             solução = vizinhoCorrente
15:             k = 0
16:         else:
17:             k += 1
18:     return solução, tamanhoCorte(solução)
```

Os operadores de vizinhanças que foram utilizados estão ilustrados na Figura 7.

Figura 7 – Operadores de vizinhanças entre subconjuntos



Fonte: O autor (2022)

Uma descrição de cada operador de vizinhança que foi implementado é descrito a seguir:

- Shift(1,0) — Consiste na transferência de um vértice de um subconjunto para outro subconjunto.
- Shift(2,0) — Consiste na transferência de dois vértices, k1 e k2, de um subconjunto para outro.
- Shift(3,0) — Consiste na transferência de três vértices, k1, k2 e k3, de um subconjunto para outro.
- Swap(1,1) — Consiste na troca de um vértice de um subconjunto com um vértice de outro subconjunto.
- Swap(2,2) — Consiste na troca de dois vértices adjacentes, k1 e k2, de um subconjunto com dois vértices adjacentes, k3 e k4, de outro subconjunto.
- Swap(3,3) — Consiste na troca de três vértices adjacentes, k1, k2 e k3, de um subconjunto com três vértices adjacentes, k4, k5 e k6, de outro subconjunto.

Assim, enquanto houver operadores de vizinhança, na linha 11 do pseudocódigo é formada uma lista com as vizinhanças geradas pelo operador da iteração atual, e um vizinho corrente é escolhido a partir do método do *best improvement*, em que o melhor indivíduo da vizinhança é selecionado. Assim, se o valor do corte máximo desse vizinho corrente for maior que o tamanho do corte da solução, a solução passa a ser o vizinho corrente, e a busca local tenta melhorar a solução no mesmo operador, se não a busca local executa a formação de uma nova vizinhança com o próximo operador da lista de vizinhanças.

4.2.2.1.2 GRASP-VNS

A segunda MH com a GRASP na etapa de busca local teve uma hibridização utilizando o método *Variable Neighborhood Search* (VNS). No pseudocódigo do Algoritmo 5 são apresentados os passos para a execução deste método na solução do *max-cut*. No GRASP-VNS, o alpha de melhor adaptação também foi de 0,5, com uma solução inicial gerada da mesma forma do que a apresentada na GRASP-VND. O parâmetro do máximo de iterações sem melhoria nesse caso foi igual a 2, com os mesmos seis operadores de vizinhança que o método anterior, a diferença é que neste método há um laço de enquanto não houver 2 iterações sem melhoria em todo o processo de busca, o procedimento se repete.

Neste procedimento a solução corrente é igual a solução inicial e enquanto houver operadores de vizinhanças, é gerada uma lista de vizinhos a partir do operador da iteração da

solução corrente, e dessa lista é selecionado aleatoriamente um vizinho. Este vizinho passa pelo método VND, assim é realizada uma verificação para averiguar se o tamanho do corte do vizinho melhorado pela VND é maior do que o corte da solução corrente. Se for, então o vizinho VND passa a ser a nova solução corrente, e o procedimento se repete com o mesmo operador de vizinhança, se não o procedimento é realizado com o próximo operador.

Algoritmo 5 – Meta-heurística GRASP-VNS

```
1 : def MH_GRASP_VNS(G(V,A) ):
2 :     alpha = 0.5
3 :     tamPartição = valorAleatório(QuantidadeDeVértices)
4 :     partEsq = GraspConstrutiva(alpha, tamPartição)
5 :     partDir = diferençaSimétrica (Vértices, partEsq)
6 :     soluçãoInicial = [partEsq,partDir]
7 :     maxIteSemMelhoria = 2
8 :     solução = solucaoInicial
9 :     vizinhancas = [ Shift1, Shift2, Shift3, Swap11, Swap22, Swap33]
10:     i = 0
11:     while i < maxIteSemMelhoria:
12:         soluçãoCorrente = solucaoInicial
13:         k= 0
14:         while k < len(vizinhancas):
15:             Vizinhos = vizinhancas[k](soluçãoCorrente)
16:             vizinhoAleatório = escolhaAleatória(Vizinhos)
17:             vizinhoVND = VND(vizinhoAleatório)
18:             if tamanhoCorte(vizinhoVND) > tamanhoCorte(soluçãoCorrente):
19:                 soluçãoCorrente = vizinhoVND
20:                 k = 0
21:             else:
22:                 k+= 1
23:         if tamanhoCorte(soluçãoCorrente) > tamanhoCorte(solução):
24:             solução = soluçãoCorrente
25:             i = 0
26:         else:
27:             i += 1
28:     return solução, tamanhoCorte(solução)
```

Realizados os procedimentos de busca com todos os operadores de vizinhança, é verificado se o tamanho do corte da solução corrente é maior que o tamanho do corte da solução principal. Se sim, a solução passa a ser a solução corrente, e por ter havido melhoria o procedimento se repete; se não, adiciona-se uma unidade a *i*, que representa a quantidade de iterações sem melhoria e se inicia igual a zero. A repetição do procedimento de busca é realizada enquanto não for atingido o máximo de iterações sem melhoria. No final, o algoritmo retorna a solução, que é a bipartição, e o tamanho do corte induzido pela bipartição.

4.2.2.2 Busca tabu

Na construção da solução inicial da meta-heurística busca tabu, apresentado no pseudocódigo do Algoritmo 6, foi utilizado um procedimento aleatório para gerar dois subconjuntos com a partição dos vértices do grafo. Uma solução corrente assume essa mesma solução inicial, sendo atribuído 0 à iteração *k*, o parâmetro da melhor iteração igual a *k*, e a *BTmáx* que é o número máximo de iterações sem melhoria no valor da melhor solução igual a 4 e uma lista tabu inicialmente vazia. Assim, é gerada uma lista de vizinhanças ordenada de forma decrescente, essa vizinhança é formada pelos seis operadores de vizinhança descritos no tópico 4.2.2.1.1, como essa lista está em ordem decrescente, uma nova solução corrente é a primeira bipartição dessa lista, e também é adicionada na primeira posição da lista tabu essa nova solução corrente.

Algoritmo 6 – Meta-heurística Busca Tabu

```
1 : def MH_BT(G(V,A) ):
2 :     soluçãoInicial = bipartição aleatória
3 :     soluçãoCorrente = soluçãoInicial
4 :     k = 0
5 :     melhorIter = k
6 :     BTmáx = 4
7 :     lista Tabu = [ ]
8 :     VizinhosOrdenados = listaDeVizinhança(soluçãoCorrente)
9 :     soluçãoCorrente = VizinhosOrdenados[0][1]
10:     listaTabu.insert(0, VizinhosOrdenados[0][1])
11:     solução = soluçãoCorrente
12:     while (k - melhorIter ≤ BTmax):
13:         k += 1
14:         VizinhosOrdenados = listaDeVizinhança(soluçãoCorrente)
15:         for j in VizinhosOrdenados:
16:             if j[1] not in listaTabu:
17:                 soluçãoCorrente = j[1]
18:                 listaTabu.insert(0,j[1])
19:                 listaTabu = listaTabu[:5]
20:                 break
21:         if tamanhoCorte(soluçãoCorrente) > tamanhoCorte(solução):
22:             solução = soluçãoCorrente
23:             melhorIter = k
24:     return solução, tamanhoCorte(solução)
```

Após esses procedimentos o método entra em um laço repetitivo onde enquanto a iteração atual menos a iteração onde é obtido o melhor resultado for menor ou igual ao número máximo de iterações sem melhoria, é dado um passo para a próxima iteração, e uma nova lista de vizinhos ordenados é gerada. Assim, para cada vizinho nesta lista, é avaliado se cada partição não está na lista tabu. Se não estiver, a nova solução corrente é essa bipartição, sendo também

inserido na primeira posição da lista tabu. A lista tabu ainda passa por uma atualização onde ela só tem o tamanho que armazena no máximo 5 bipartições, assim quando a lista estiver cheia a bipartição que estiver na última posição é removida, e uma nova bipartição é inserida na primeira posição.

Se o tamanho do corte da solução corrente for maior que o tamanho do corte da solução, a solução passa a ser igual à solução corrente e a melhor iteração passa a ser a iteração atual. Dessa forma, o laço será repetido até o momento em que a diferença entre a iteração atual e a iteração do melhor resultado for maior que o máximo de iterações sem melhoria.

Assim, nesse procedimento, são permitidos movimentos de piora, de maneira determinística. A cada iteração o algoritmo moverá para o melhor vizinho, desde que não faça movimentos tabus. O movimento tabu será permitido se o movimento gerar uma solução melhor ou quando o movimento tabu for liberado. Assim serão ainda utilizados mecanismos auxiliares para intensificação, focando em regiões do espaço de busca mais promissores e a diversificação, utilizando recursos algorítmicos para forçarem a busca para um espaço de soluções ainda não explorados.

4.2.2.3 Algoritmos genéticos

Para implementação do algoritmo genético, esta pesquisa utilizou um tamanho de população com 200 indivíduos, sendo evoluídos em 30 gerações, com uma taxa de mutação de 20%, esses parâmetros trazem bons tempos computacionais, porem com menor acurácia na busca pela solução. Na formação da população inicial, são criados aleatoriamente indivíduos (bipartições) em quantidade definida pelo parâmetro do tamanho da população. Essas bipartições são inseridas em uma lista chamada de população, e é definido um melhor indivíduo de toda a população como solução inicial. O pseudocódigo da meta-heurística implementada para resolução do corte máximo é representado no Algoritmo 7.

Para cada geração, é utilizado um método de seleção por torneio, selecionando aleatoriamente três indivíduos na população, e entre esses três indivíduos é escolhido o melhor, sendo definido assim um pai e uma mãe entre os indivíduos da geração, sendo realizado um cruzamento entre os dois, formando um novo indivíduo com partes aleatórias de um e partes aleatórias do outro, sendo formada assim uma nova população a partir de sucessivos cruzamentos. Para cada indivíduo da nova população, ocorrerá mutações aleatórias no cromossomo dos indivíduos de acordo com a taxa de mutação de 20%, sendo adicionado ou retirado um vértice de uma das partições.

Algoritmo 7 – Meta-heurística Algoritmos Genéticos

```
1 : def MH_AG( G(V,A) ):
2 :     tamPop = 200
3 :     numGer = 30
4 :     taxaMut = 0.2
5 :     solução = [ [ ], [ ] ]
6 :     População = []
7 :     for i in range( tamPop ):
8 :         soluçãoInicial = bipartiçãoAleatória
9 :         individuo = soluçãoInicial
10:         População.append( individuo )
11:     melInd = MelhorIndividuo( População )
12:     for geração in range(numGer) :
13:         geração += 1
14:         novaPopulação = []
15:         while len(novaPopulação) < tamPop:
16:             pai = SeleçãoTorneio( População )
17:             mãe = SeleçãoTorneio( População )
18:             cruzamento = crossover( pai, mãe )
19:             novaPopulação.append( cruzamento )
20:         for indMut in novaPopulação:
21:             indMut = mutação( indMut, taxaMut )
22:         População = novaPopulação
23:         melIndCorr = MelhorIndividuo( População )
24:         if tamanhoCorte(melIndCorr) > tamanhoCorte( melInd ):
25:             melInd = melIndCorr
26:         if tamanhoCorte( melInd ) > tamanhoCorte( solução ):
27:             solução = melInd
28:     return solução, tamanhoCorte( solução )
```

Então, a lista da população principal passa a ser essa nova população gerada pelos indivíduos que passaram pelo cruzamento e pela mutação, e um melhor indivíduo corrente é selecionado da população. Se o tamanho do corte do melhor indivíduo corrente for maior que o tamanho do corte do melhor indivíduo, o melhor indivíduo passa a ser o indivíduo corrente e, da mesma forma, a solução passa a ser o melhor indivíduo se este for maior que a solução atual. Assim, após todas as gerações, o algoritmo retorna a solução com a bipartição e o corte induzido por esta.

4.2.2.4 Simulated Annealing

A MH Simulated Annealing utilizou um parâmetro de resfriamento *beta* igual a 0.1. Na temperatura inicial, chamada de *Tzero* foi usada a temperatura igual a 100, para não fazer com que o procedimento fosse muito pesado. Para o número de iterações para cada temperatura denominado *SAmax* foi utilizada 1 iteração. No *IterT*, que é número de iterações na temperatura T foi também utilizado 1 iteração, a temperatura corrente T é igual a temperatura inicial *Tzero*,

a iteração *K* é inicialmente 0, o melhor *K*, ou melhor iteração, é inicialmente igual a iteração corrente e a solução inicial é gerada aleatoriamente.

Algoritmo 8 – Meta-heurística *Simulated Annealing*

```
1 : def MH_SA( G(V,A) ):
2 :     beta = 0.1
3 :     Tzero = 100
4 :     SAmax = 1
5 :     IterT = 1
6 :     T = Tzero
7 :     k = 0
8 :     melhorK = k
9 :     soluçãoInicial = bipartiçãoAleatória
10:     soluçãoCorrente = soluçãoInicial
11:     solução = soluçãoCorrente
12:     While T > 0.01:
13:         While IterT < SAmax:
14:             k += 1
15:             IterT += 1
16:             Vizinhos = listaDeVizinhança(soluçãoCorrente)
17:             soluçãoAleatória = escolhaAleatória(Vizinhos)
18:             delta = tamanhoCorte(soluçãoAleatória) - tamanhoCorte(soluçãoCorrente)
19:             if delta >  0 :
20:                 soluçãoCorrente = soluçãoAleatória
21:             else:
22:                 x = valor aleatório entre 0 e 1
23:                 if x < e^(delta/T):
24:                     soluçãoCorrente = soluçãoAleatória
25:         if tamanhoCorte(soluçãoCorrente) >  tamanhoCorte(solução):
26:             solução = soluçãoCorrente
27:             melhorK = k
28:             T += 2 * ( beta * T)
29:         else:
30:             T = T / (1 + ((k- melhorK)/100) + beta * T)
31:             IterT = 0
32:     return solução,tamanhoCorte(solução)
```

Após inicialização, o método entra em um laço de repetição, onde enquanto *T* for maior que 0,001, que foi a temperatura de resfriamento de melhor adaptação para encontrar melhores soluções e enquanto o número de iterações na temperatura *T* for menor que número de iterações para cada temperatura. No início do procedimento são adicionadas uma unidade para o *k* e *IterT* e é criada uma lista de vizinhos, que contém todas a possíveis vizinhanças com todos os operadores, sendo eles, nesse caso, shift1, shift2, shift3, swap11, swap22 e swap33. Assim, é selecionado aleatoriamente um vizinho e calculado o *delta* entre a solução aleatória e a solução corrente.

O *delta* sendo maior que 0, a solução corrente é substituída pela solução aleatória. Caso contrário, é calculado aleatoriamente um valor de *x* entre 0 e 1, se *x* for menor que $e^{delta/T}$ a solução corrente é substituída pela solução aleatória, quando o laço de repetição *IterT* < *SAmax* for rompido é verificado se o tamanho do corte da solução corrente é maior que o tamanho do corte da solução. Se sim, a solução corrente passa a ser a nova solução e a melhor iteração passa a ser a iteração corrente, sendo também adicionado *2 * (beta * T)* na temperatura corrente *T*. Se não, a temperatura corrente passa a ser *T / (1 + ((k- melhorK)/100) + beta * T)*, com o número de iterações na temperatura *T* igual a 0. Quando a temperatura for resfriada chegando a ser menor que 0,01 o procedimento é interrompido e assim o algoritmo retorna a solução com o corte obtido.

4.2.3 Fase 3: Experimentos computacionais

O algoritmo exato proposto nesta pesquisa e as meta-heurísticas apresentadas foram implementados em Python 3 e executados em um computador com processador Intel Core i7, 8GB de memória instalada (RAM), sistema operacional Windows 10 de 64bits. As instâncias utilizadas foram geradas aleatoriamente por um código capaz de gerar grafos de qualquer tamanho e densidade desejados, sendo estes parâmetros requeridos para a geração. Não foi aplicado benchmark por razão dos trabalhos publicados na literatura não disponibilizarem bibliotecas de instâncias de grafos compatíveis com o método exato desta pesquisa. As quantidades de instâncias de grafos que foram executadas nos algoritmos estão apresentadas na Tabela 2.

Tabela 2 – Instâncias em número de grafos para o experimento

Densidade / Grafos	Range	25%	50%	75%	TOTAL
Exemplo I	(5, 13, 17)	-	-	3	3
Experimento I	(5, 9, 13)	90	90	90	270
Experimento II	(5 até 41)	37	37	37	111
TOTAL					**384**

Fonte: O autor (2022)

No exemplo I foi demonstrado o funcionamento da ABB para três grafos aleatórios, com 5, 13 e 17 vértices, todos com 75% de densidade. O experimento I executa o algoritmo ABB em 270 grafos aleatórios, sendo grafos com tamanho de 5, 9 e 13 vértices, com as densidades de 25, 50 e 75% com 30 diferentes grafos para cada combinação de caso. Por fim, o experimento II analisa o comportamento da ABB em relação as MH e executa os algoritmos

para grafos de 25, 50 e 75%, com a quantidade de vértices com um limite inferior de $h = 1$ e superior de $h = 10$, tendo assim grafos com 5 até 41 vértices.

Assim, serão consideradas 384 instâncias, nas quais a abordagem exata terá o algoritmo executado apenas uma vez, pois o procedimento será o mesmo sempre que executado para um determinado grafo. Já os algoritmos meta-heurísticos serão executados trinta vezes para serem realizadas as análises estatísticas devido a sua aleatoriedade.

4.2.4 Fase 4: Análises estatísticas

Estatística é a ciência de coletar, organizar, analisar, interpretar e apresentar dados. Alguns especialistas preferem chamá-la de ciência de dados, que é uma trilogia que envolve modelagem, análise e tomada de decisão (DOANE; SEWARD, 2014). A partir dos experimentos computacionais foram coletados os dados da solução e tempo de execução dos algoritmos. Com a realização dos experimentos, os dados coletados foram organizados em tabelas. O Apêndice A contém os resultados do experimento I da ABB, com informações dos cortes e tempos para cada classe de grafos, além da média dos tempos. Já os resultados do experimento II estão nos Apêndices B, C, D, E, F e G, com informações de melhor, média e desvio padrão para os cortes e tempos de execução do algoritmo.

Assim para uma melhor análise dos resultados além da estatística descritiva, já citada, foi também utilizada uma análise descritiva visual através de gráficos de pontos, de linha, utilizando também cálculos de regressão polinomial, para previsões de tempo de solução para casos que o algoritmo não foi executado. Para identificar a regressão de melhor adaptação foram verificados os erros absolutos médios e os erros da raiz quadrática média, tendo assim uma melhor compreensão dos dados das meta-heurísticas e das comparações entre as abordagens, facilitando as interpretações.

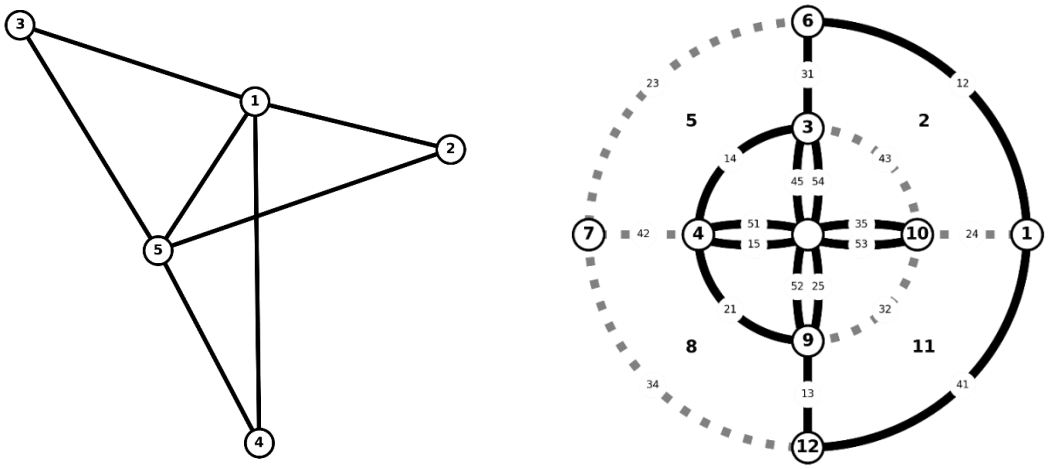
Para analisar a complexidade da ABB, adotou-se a notação Big O, que utiliza uma noção de certa constante vezes n. Além disso, considera-se ainda a objetividade e a simplicidade nessa notação para evitar ambiguidades. Dessa forma, a notação Big O expressa a complexidade de um algoritmo através de um único termo cujo coeficiente é igual a 1, o que permite ter uma ideia concisa de como o tempo de processamento do algoritmo se comporta com grandes valores de entrada. A análise da eficiência de um algoritmo através da notação Big O é uma análise do limite superior assintótico do tempo de processamento do algoritmo, ou seja, um tempo de processamento para o pior caso (AHO; ULLMAN, 1992).

5 RESULTADOS E DISCUSSÕES

5.1 DEMONSTRAÇÃO DE EXEMPLOS DO ALGORITMO

Como primeiro exemplo para demonstração do algoritmo da árvore de busca binária é considerado um grafo $G = (V, A)$, onde V = [1,2,3,4,5] e A = [[1,2],[1,3],[1,4],[1,5], [2,5],[3,5],[4,5]], sendo assim um grafo com cinco vértices e sete arestas, o grafo G_1 é aleatório com 75% de densidade. A Figura 8 apresenta o grafo G_1, assim como o ORG_1 mapeando G_1 contendo 14 arestas grifadas, sendo assim G_1 um subgrafo do ORG_1.

Figura 8 – Grafo aleatório G_1 com h = 1 e ORG_1 representando as arestas de G_1



Fonte: O autor (2022)

Para a realização da formulação do programa linear são utilizados o conjunto A de arestas de G_1, o conjunto de caminhos diagonais do ORG_1, *O = [124, 1243, 143, 142, 1423, 123, 152, 253, 354, 145, 234, 1342, 132, 243, 1324, 134]* e o conjunto de caminhos diagonais do ORG_1 sinalizados, *U = [124n0, 124p1, 124p2, 124p3, 1243n1, 1243n2, 1243n3, 1243n4, 1243p1, 1243p2, 1243p3, 1243p4, 143n0, 143p1, 143p2, 143p3, 142n0, 142p1, 142p2, 142p3, 1423n1, 1423n2, 1423n3, 1423n4, 1423p1, 1423p2, 1423p3, 1423p4, 123n0, 123p1, 123p2, 123p3, 234n0, 234p1, 234p2, 234p3, 1342n1, 1342n2, 1342n3, 1342n4, 1342p1, 1342p2, 1342p3, 1342p4, 132n0, 132p1, 132p2, 132p3, 243n0, 243p1, 243p2, 243p3, 1324n1, 1324n2, 1324n3, 1324n4, 1324p1, 1324p2, 1324p3, 1324p4, 134n0, 134p1, 134p2, 134p3, 152n0, 152p1, 152p2, 152p3, 253n0, 253p1, 253p2, 253p3, 354n0, 354p1, 354p2, 354p3, 145n0, 145p1, 145p2, 145p3]*. Assim, a partir desses conjuntos a função objetivo e as restrições são descritas conforme (4.1) e (4.2) como:

$$Maximize \quad x_{12} + x_{13} + x_{14} + x_{15} + x_{25} + x_{35} + x_{45} + z_{124} + z_{1243} + z_{143} + z_{142} + z_{1423} + z_{123} + z_{152} + z_{253} + z_{354} + z_{145} + z_{234} + z_{1342} + z_{132} + z_{243} + z_{1324} + z_{134}$$

Sujeito a:

$$y_{124n0}: +x_{12} + x_{24} + x_{41} + z_{124} \leq 2$$
$$y_{124p1}: +x_{12} - x_{24} - x_{41} + z_{124} \leq 0$$
$$y_{124p2}: -x_{12} + x_{24} - x_{41} + z_{124} \leq 0$$
$$y_{124p3}: -x_{12} - x_{24} + x_{41} + z_{124} \leq 0$$
$$y_{1243n1}: -x_{12} + x_{24} + x_{43} + x_{31} + z_{1243} \leq 2$$
$$y_{1243n2}: +x_{12} - x_{24} + x_{43} + x_{31} + z_{1243} \leq 2$$
$$y_{1243n3}: +x_{12} + x_{24} - x_{43} + x_{31} + z_{1243} \leq 2$$
$$y_{1243n4}: +x_{12} + x_{24} + x_{43} - x_{31} + z_{1243} \leq 2$$
$$y_{1243p1}: +x_{12} - x_{24} - x_{43} - x_{31} + z_{1243} \leq 0$$
$$y_{1243p2}: -x_{12} + x_{24} - x_{43} - x_{31} + z_{1243} \leq 0$$
$$y_{1243p3}: -x_{12} - x_{24} + x_{43} - x_{31} + z_{1243} \leq 0$$
$$y_{1243p4}: -x_{12} - x_{24} - x_{43} + x_{31} + z_{1243} \leq 0$$
$$y_{143n0}: +x_{14} + x_{43} + x_{31} + z_{143} \leq 2$$
$$y_{143p1}: +x_{14} - x_{43} - x_{31} + z_{143} \leq 0$$
$$y_{143p2}: -x_{14} + x_{43} - x_{31} + z_{143} \leq 0$$
$$y_{143p3}: -x_{14} - x_{43} + x_{31} + z_{143} \leq 0$$
$$y_{142n0}: +x_{14} + x_{42} + x_{21} + z_{142} \leq 2$$
$$y_{142p1}: +x_{14} - x_{42} - x_{21} + z_{142} \leq 0$$
$$y_{142p2}: -x_{14} + x_{42} - x_{21} + z_{142} \leq 0$$
$$y_{142p3}: -x_{14} - x_{42} + x_{21} + z_{142} \leq 0$$
$$y_{1423n1}: -x_{14} + x_{42} + x_{23} + x_{31} + z_{1423} \leq 2$$
$$y_{1423n2}: +x_{14} - x_{42} + x_{23} + x_{31} + z_{1423} \leq 2$$
$$y_{1423n3}: +x_{14} + x_{42} - x_{23} + x_{31} + z_{1423} \leq 2$$
$$y_{1423n4}: +x_{14} + x_{42} + x_{23} - x_{31} + z_{1423} \leq 2$$
$$y_{1423p1}: +x_{14} - x_{42} - x_{23} - x_{31} + z_{1423} \leq 0$$
$$y_{1423p2}: -x_{14} + x_{42} - x_{23} - x_{31} + z_{1423} \leq 0$$
$$y_{1423p3}: -x_{14} - x_{42} + x_{23} - x_{31} + z_{1423} \leq 0$$
$$y_{1423p4}: -x_{14} - x_{42} - x_{23} + x_{31} + z_{1423} \leq 0$$
$$y_{123n0}: +x_{12} + x_{23} + x_{31} + z_{123} \leq 2$$
$$y_{123p1}: +x_{12} - x_{23} - x_{31} + z_{123} \leq 0$$
$$y_{123p2}: -x_{12} + x_{23} - x_{31} + z_{123} \leq 0$$
$$y_{123p3}: -x_{12} - x_{23} + x_{31} + z_{123} \leq 0$$
$$y_{234n0}: +x_{23} + x_{34} + x_{42} + z_{234} \leq 2$$
$$y_{234p1}: +x_{23} - x_{34} - x_{42} + z_{234} \leq 0$$
$$y_{234p2}: -x_{23} + x_{34} - x_{42} + z_{234} \leq 0$$
$$y_{234p3}: -x_{23} - x_{34} + x_{42} + z_{234} \leq 0$$
$$y_{1342n1}: -x_{13} + x_{34} + x_{42} + x_{21} + z_{1342} \leq 2$$
$$y_{1342n2}: +x_{13} - x_{34} + x_{42} + x_{21} + z_{1342} \leq 2$$
$$y_{1342n3}: +x_{13} + x_{34} - x_{42} + x_{21} + z_{1342} \leq 2$$
$$y_{1342n4}: +x_{13} + x_{34} + x_{42} - x_{21} + z_{1342} \leq 2$$
$$y_{1342p1}: +x_{13} - x_{34} - x_{42} - x_{21} + z_{1342} \leq 0$$
$$y_{1342p2}: -x_{13} + x_{34} - x_{42} - x_{21} + z_{1342} \leq 0$$
$$y_{1342p3}: -x_{13} - x_{34} + x_{42} - x_{21} + z_{1342} \leq 0$$
$$y_{1342p4}: -x_{13} - x_{34} - x_{42} + x_{21} + z_{1342} \leq 0$$
$$y_{132n0}: +x_{13} + x_{32} + x_{21} + z_{132} \leq 2$$
$$y_{132p1}: +x_{13} - x_{32} - x_{21} + z_{132} \leq 0$$
$$y_{132p2}: -x_{13} + x_{32} - x_{21} + z_{132} \leq 0$$
$$y_{132p3}: -x_{13} - x_{32} + x_{21} + z_{132} \leq 0$$
$$y_{243n0}: +x_{24} + x_{43} + x_{32} + z_{243} \leq 2$$
$$y_{243p1}: +x_{24} - x_{43} - x_{32} + z_{243} \leq 0$$
$$y_{243p2}: -x_{24} + x_{43} - x_{32} + z_{243} \leq 0$$
$$y_{243p3}: -x_{24} - x_{43} + x_{32} + z_{243} \leq 0$$
$$y_{1324n1}: -x_{13} + x_{32} + x_{24} + x_{41} + z_{1324} \leq 2$$
$$y_{1324n2}: +x_{13} - x_{32} + x_{24} + x_{41} + z_{1324} \leq 2$$
$$y_{1324n3}: +x_{13} + x_{32} - x_{24} + x_{41} + z_{1324} \leq 2$$
$$y_{1324n4}: +x_{13} + x_{32} + x_{24} - x_{41} + z_{1324} \leq 2$$
$$y_{1324p1}: +x_{13} - x_{32} - x_{24} - x_{41} + z_{1324} \leq 0$$
$$y_{1324p2}: -x_{13} + x_{32} - x_{24} - x_{41} + z_{1324} \leq 0$$
$$y_{1324p3}: -x_{13} - x_{32} + x_{24} - x_{41} + z_{1324} \leq 0$$
$$y_{1324p4}: -x_{13} - x_{32} - x_{24} + x_{41} + z_{1324} \leq 0$$
$$y_{134n0}: +x_{13} + x_{34} + x_{41} + z_{134} \leq 2$$
$$y_{134p1}: +x_{13} - x_{34} - x_{41} + z_{134} \leq 0$$
$$y_{134p2}: -x_{13} + x_{34} - x_{41} + z_{134} \leq 0$$
$$y_{134p3}: -x_{13} - x_{34} + x_{41} + z_{134} \leq 0$$
$$y_{152n0}: +x_{15} + x_{52} + x_{21} + z_{152} \leq 2$$
$$y_{152p1}: +x_{15} - x_{52} - x_{21} + z_{152} \leq 0$$
$$y_{152p2}: -x_{15} + x_{52} - x_{21} + z_{152} \leq 0$$
$$y_{152p3}: -x_{15} - x_{52} + x_{21} + z_{152} \leq 0$$
$$y_{253n0}: +x_{25} + x_{53} + x_{32} + z_{253} \leq 2$$
$$y_{253p1}: +x_{25} - x_{53} - x_{32} + z_{253} \leq 0$$
$$y_{253p2}: -x_{25} + x_{53} - x_{32} + z_{253} \leq 0$$
$$y_{253p3}: -x_{25} - x_{53} + x_{32} + z_{253} \leq 0$$
$$y_{354n0}: +x_{35} + x_{54} + x_{43} + z_{354} \leq 2$$
$$y_{354p1}: +x_{35} - x_{54} - x_{43} + z_{354} \leq 0$$
$$y_{354p2}: -x_{35} + x_{54} - x_{43} + z_{354} \leq 0$$
$$y_{354p3}: -x_{35} - x_{54} + x_{43} + z_{354} \leq 0$$
$$y_{145n0}: +x_{14} + x_{45} + x_{51} + z_{145} \leq 2$$
$$y_{145p1}: +x_{14} - x_{45} - x_{51} + z_{145} \leq 0$$
$$y_{145p2}: -x_{14} + x_{45} - x_{51} + z_{145} \leq 0$$
$$y_{145p3}: -x_{14} - x_{45} + x_{51} + z_{145} \leq 0$$

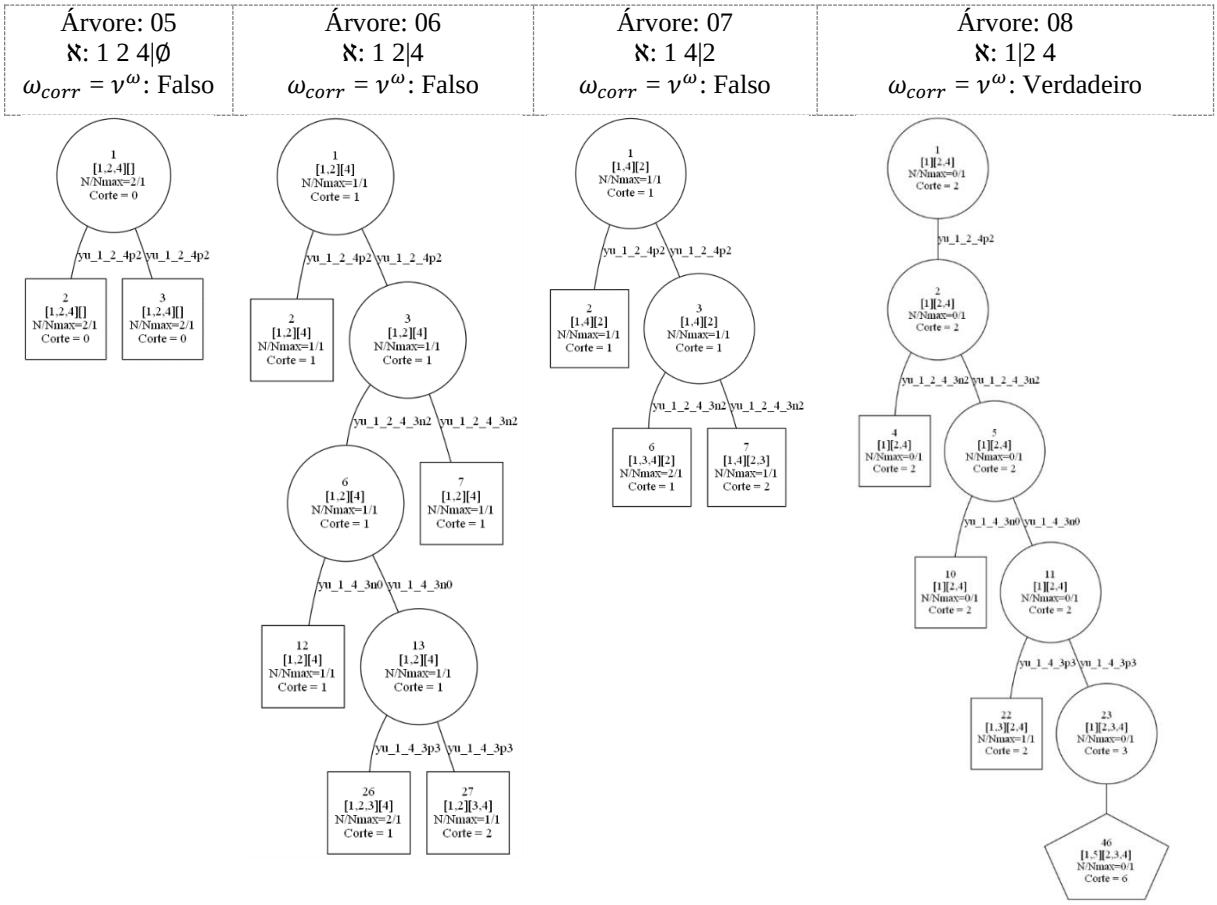
A partir da função objetivo e das restrições, este programa linear é executado no solver Soplex, configurado para retornar a solução dual. A partir da solução são retornadas as variáveis duais sinalizadas apresentadas a seguir:

1: y_{124p2}	8: y_{123p1}	15: y_{243p2}	22: y_{152p1}
2: y_{1243n2}	9: y_{234p1}	16: y_{243p3}	23: y_{253n0}
3: y_{143n0}	10: y_{234p2}	17: y_{1324n4}	24: y_{253p1}
4: y_{143p3}	11: y_{1342n2}	18: y_{1324p4}	25: y_{354n0}
5: y_{142p1}	12: y_{132n0}	19: y_{134n0}	26: y_{354p1}
6: y_{142p3}	13: y_{132p2}	20: y_{134p2}	27: y_{145n0}
7: y_{1423n4}	14: y_{243p1}	21: y_{152n0}	28: y_{145p2}

Essas vinte e oito variáveis duais representam as inequações que serão utilizadas em cada um dos níveis das árvores de busca. Neste caso a altura máxima que essa árvore poderá atingir são vinte e oito níveis. Em cada nível a inequação referente é transformada em duas equações, uma para o filho da esquerda e outra para o filho da direita. O valor da igualdade na equação do filho da esquerda é igual ao valor da inequação menos dois e é igual o valor da inequação no filho da direita.

Assim o problema do corte máximo simples é formulado como um problema de decisão no seguinte formato: dado um grafo G = (V, A), uma semente $\aleph_i$, e um inteiro positivo ν^{ω}, que representa o corte procurado, tendo todas as arestas do grafo inicialmente, há um corte de tamanho ν^{ω} na semente $\aleph_i$ no grafo G? Neste caso a Figura 9 apresenta as últimas quatro árvores com $\nu^{\omega} = 6$, uma aresta a menos que a busca inicial. A árvore 5 gerada pela semente $\aleph_{1\ 2\ 4|\emptyset}$ já cria dois filhos contraditórios inicialmente, essas podas são feitas devido a $N = 2$ superar o $N_{Máx} = 1$.

Figura 9 – Árvores de busca binaria do grafo aleatório de h = 1 com as sementes do ν^{ω} final



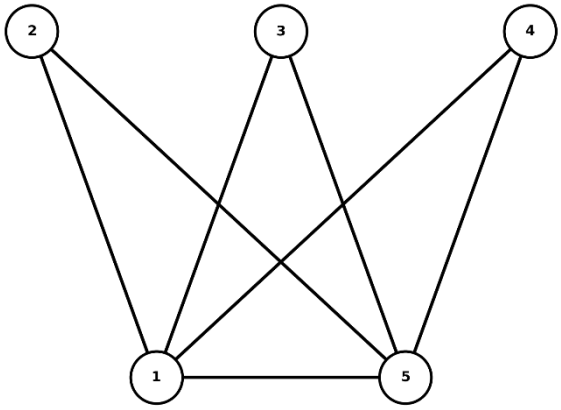
Fonte: O autor (2022)

A árvore 6 da semente $\aleph_{1\ 2|4}$ o primeiro filho da raiz, o nó 2 tem como equação $-x_{12} + x_{24} - x_{41} = -2$, as variáveis só podem assumir valores iguais a zero ou um, sendo

cada variável a representação de uma aresta. Neste caso, a bipartição induzida pela semente tem as variáveis $x_{12} = 0$, $x_{24} = 1$, e $x_{41} = 1$ e a equação fica $-0 + 1 - 1 = -2$, o que leva a uma contradição pois 0 não é igual a -2. Já no nó 3, não é posicionado nenhum vértice mas a equação binaria é correta. Assim, o algoritmo segue para o nó 6 e 12 sendo este contraditório pela equação, 13, 26 contraditório devido o $N = 2$ superar o $N_{Máx} = 1$, 27 sendo contraditório pela equação e o 7 também contraditório, encerrando assim a busca nessa semente e partindo para a próxima.

A árvore 7 gerada pela semente $\aleph_{1\,4|2}$ tem o nó 2 contraditório por causa da equação, indo para o nó 3 que gera o nó 6 sendo esse contraditório devido ao N. No filho 7, há uma contradição por conta da equação. Na árvore 8 gerada pela semente $\aleph_{1|2\,4}$ há o corte máximo; o percurso é 1, 2, 4 contraditório devido à equação; 5, 10 contraditório; 11, 22 contraditório; o nó 23 é o ponto do algoritmo em que o número de vértices posicionados é igual a 4h. Nesse caso, são criados um filho à esquerda com o vértice 4h+1 na partição da esquerda e outro filho à direita com o último vértice na direita. O filho da esquerda, o nó 46 é pentagonal, representando que ele encontrou o resultado ótimo, com corte máximo igual a 6 e uma bipartição que induz a este corte sendo [[1, 5], [2, 3, 4]]. A Figura 10 mostra o grafo biparticionado com os vértices do primeiro subconjunto localizados abaixo e os vértices do segundo subconjunto acima, ficando claro que as arestas cortadas são [1,2], [1,3], [1,4], [2,5], [3,5], [4,5] com apenas uma aresta sem ser cortada sendo ela a [1,5].

Figura 10 – Grafo aleatório h = 1 com a bipartição dos dois subconjuntos que induzem ao corte máximo

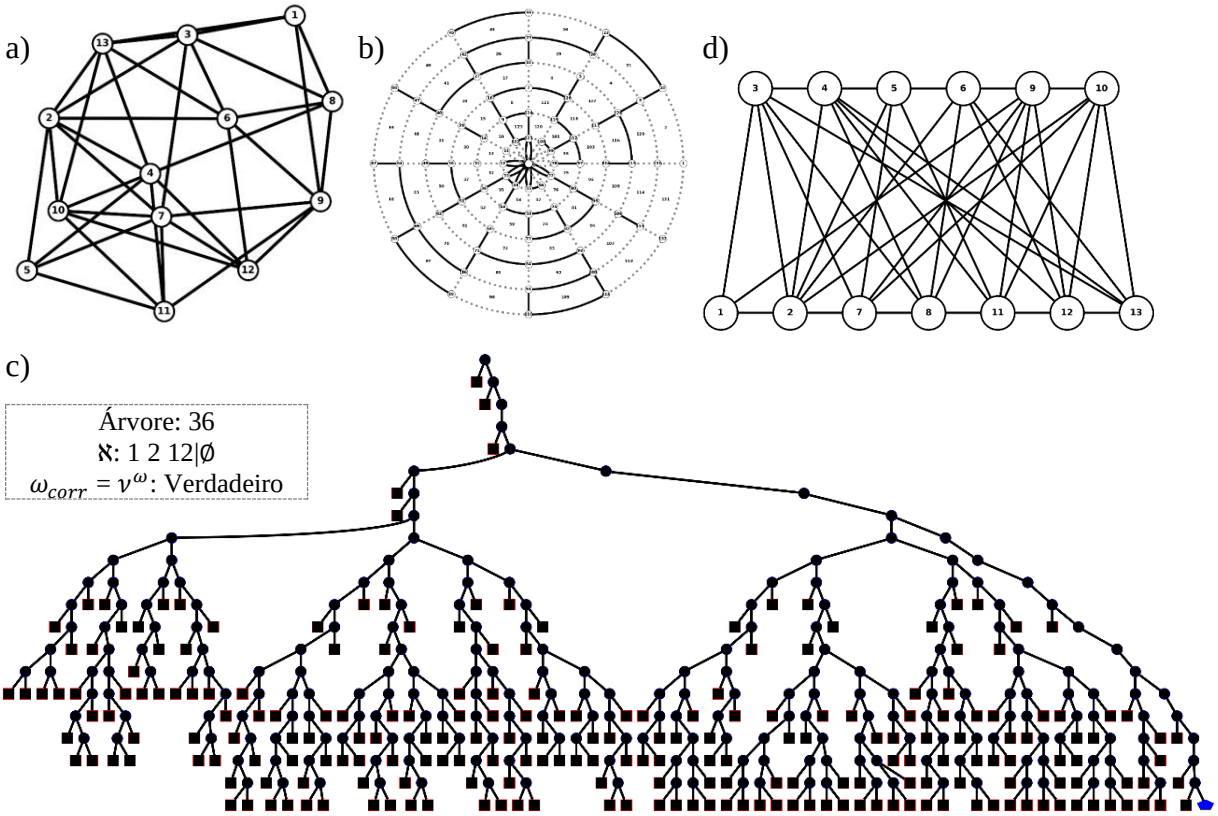


Fonte: O autor (2022)

Após essa apresentação detalhada do funcionamento do algoritmo, é apresentado também um segundo exemplo com h = 3, nesse exemplo é considerado um grafo $G = (V, A)$, sendo V = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] e A = [[1, 3], [1, 8], [1, 9], [1, 13], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 10], [2, 13], [3, 6], [3, 7], [3, 8], [3, 13], [4, 5], [4, 8], [4, 10], [4, 11], [4, 12], [4, 13], [5, 7], [5, 11], [6, 8], [6, 9], [6, 12], [6, 13], [7, 9], [7, 10], [7, 11], [7, 12], [8,

9], [9, 11], [9, 12], [10, 11], [10, 12], [10, 13]], sendo este um grafo com 13 vértices e 37 arestas, o grafo G_3 é um grafo aleatório com 75% de densidade. A Figura 11 apresenta o grafo G_3, assim como o ORG_3 com G_3 sendo representado como um subgrafo do ORG_3.

Figura 11 – a)Grafo aleatório de h = 3, b) ORG_3 representando as arestas, c) ABB do grafo aleatório de h = 3 com a semente do ν^{ω} final, d) Grafo com a bipartição dos dois subconjuntos que induzem ao corte máximo



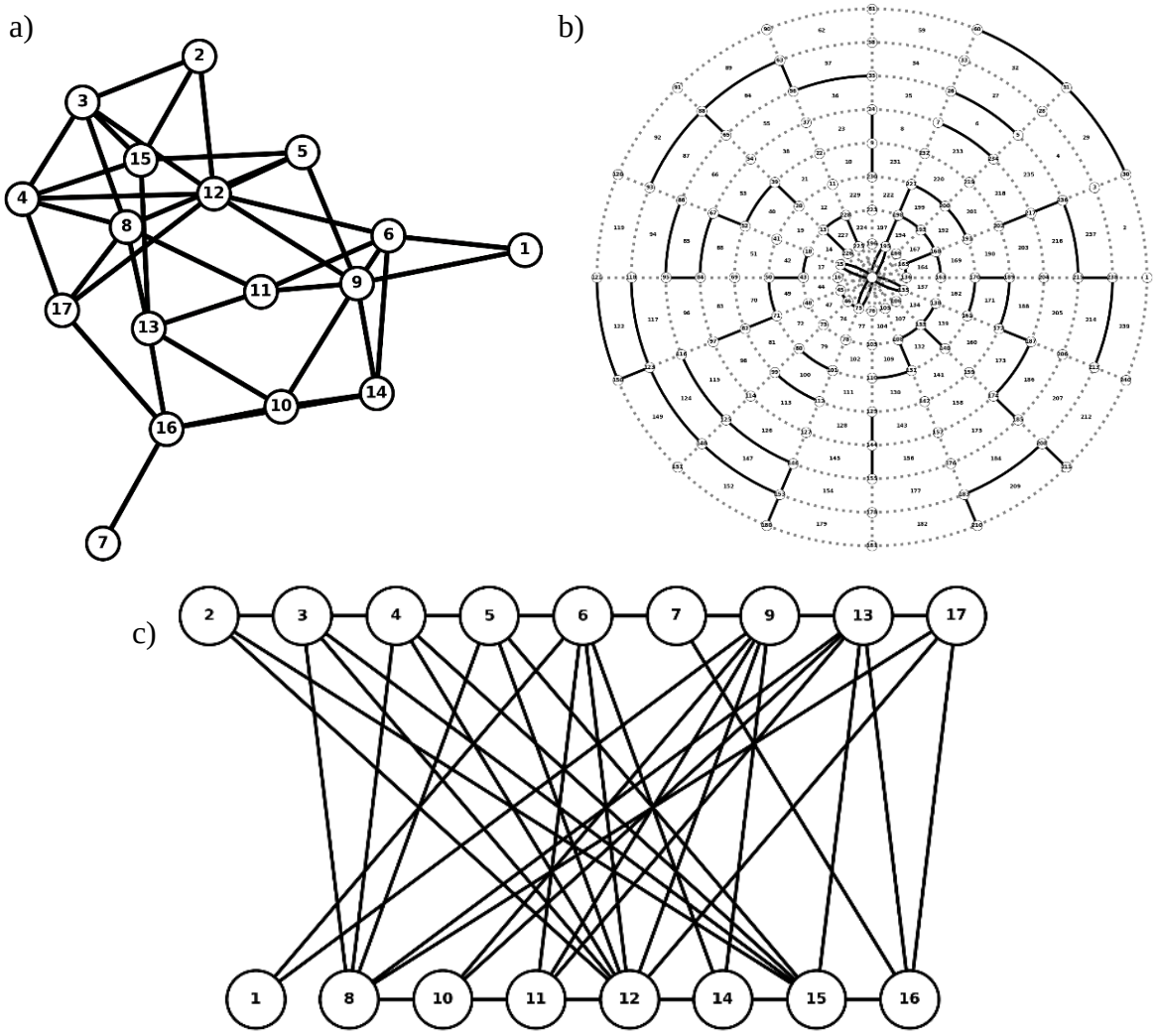
Fonte: O autor (2022)

Esse exemplo tem o PLM gerado conforme (4.1) e (4.2), tendo neste caso 1008 restrições, com o resultado dual retornando as variáveis duais. A partir disso com a execução da ABB são geradas 35 árvores sem sucesso na busca do corte máximo, inicializando com $\nu^{\omega} = 37$, até quando na semente $\aleph_{1\ 2\ 12|\emptyset}$, com $\nu^{\omega} = 27$, a árvore 36 encontra o nó pentagonal, tendo como bipartição os dois subconjuntos [[1, 2, 7, 8, 11, 12, 13], [3, 4, 5, 6, 9, 10]] induzindo ao corte máximo igual a 27. A Figura 11 d) mostra a representação do grafo biparticionado com as arestas cortadas separadas pelos dois conjuntos de vértices em lado opostos.

Como último exemplo de demonstração é considerado um caso de h = 4, com um grafo com um conjunto de vértices V igual a [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17], e um conjunto de arestas A, igual a [[1, 6], [1, 9], [2, 3], [2, 12], [2, 15], [3, 4], [3, 8], [3, 12], [3, 15], [4, 8], [4, 12], [4, 15], [4, 17], [5, 8], [5, 9], [5, 12], [5, 15], [6, 9], [6, 11], [6, 12], [6, 14], [7, 16], [8, 11], [8, 13], [8, 17], [9, 10], [9, 11], [9, 12], [9, 14], [10, 13], [10, 14], [10, 16], [11,

13], [12, 17], [13, 15], [13, 16], [14, 16], [16, 17]]. A Figura 12 apresenta o grafo G_4, assim como o ORG_4 com G_4 sendo representado como um subgrafo.

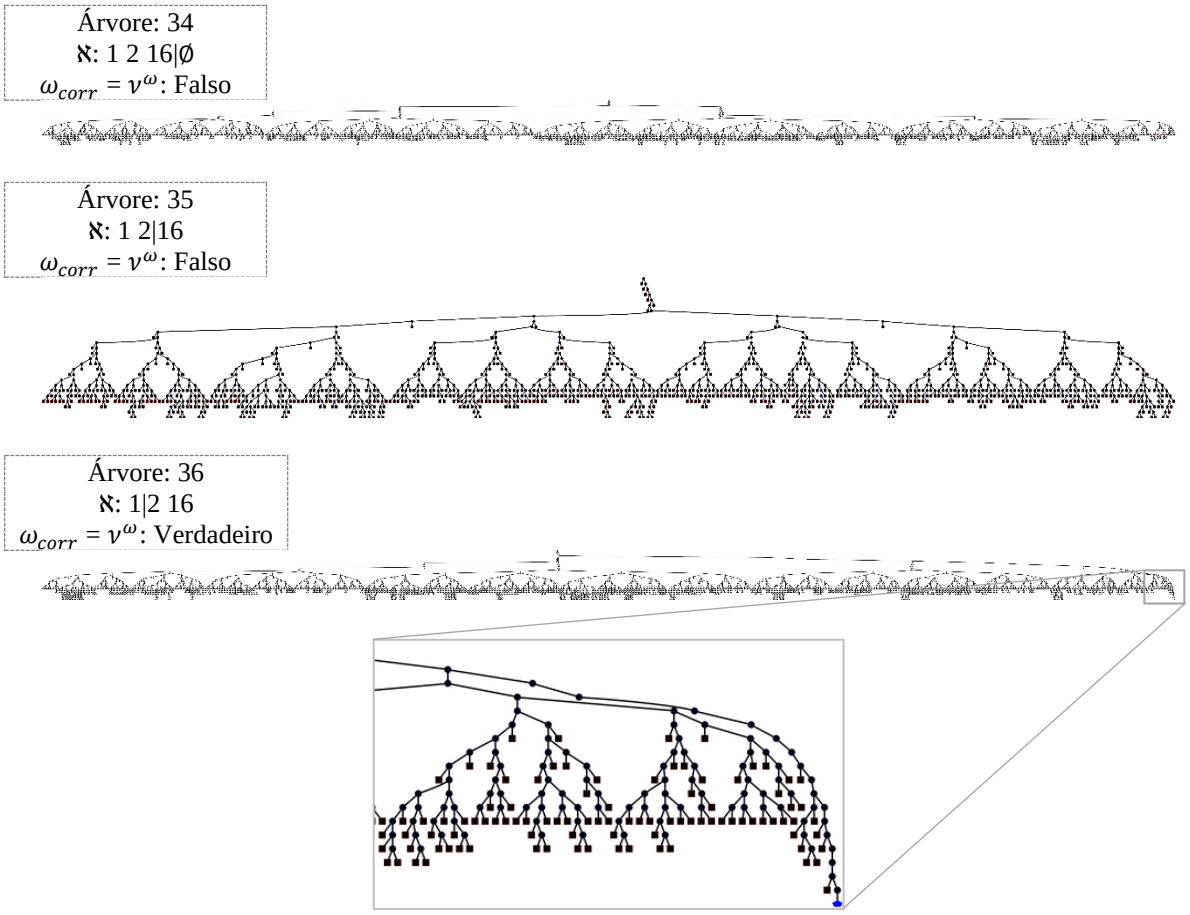
Figura 12 – a) Grafo aleatório de h = 4 b) ORG_4 representando as arestas de G_4 c) Grafo aleatório h = 4 com a bipartição dos dois subconjuntos que induzem ao corte máximo



Fonte: O autor (2022)

Nesta demonstração o programa linear mestre obtido por (4.1) e (4.2), tem como função objetivo a maximização do somatório das variáveis de cada aresta do grafo, mais o somatório das variáveis do caminho diagonal, com 1856 restrições, e limite das variáveis das arestas igual a zero ou um, e para as variáveis do caminho diagonal maior ou igual a zero. As variáveis duais obtidas por meio da solução dual do PLM, trazem as equações binárias utilizadas nos níveis da ABB. Este exemplo é interessante pois com a execução da ABB são geradas trinta e seis árvores para encontrar o corte máximo, a Figura 13 apresenta as árvores do último v^{ω} demonstrando primariamente grandes larguras que podem ser atingidas pelas árvores.

Figura 13 – Árvores de busca binária do grafo aleatório de h = 4 com as sementes do ν^{ω} final



Fonte: O autor (2022)

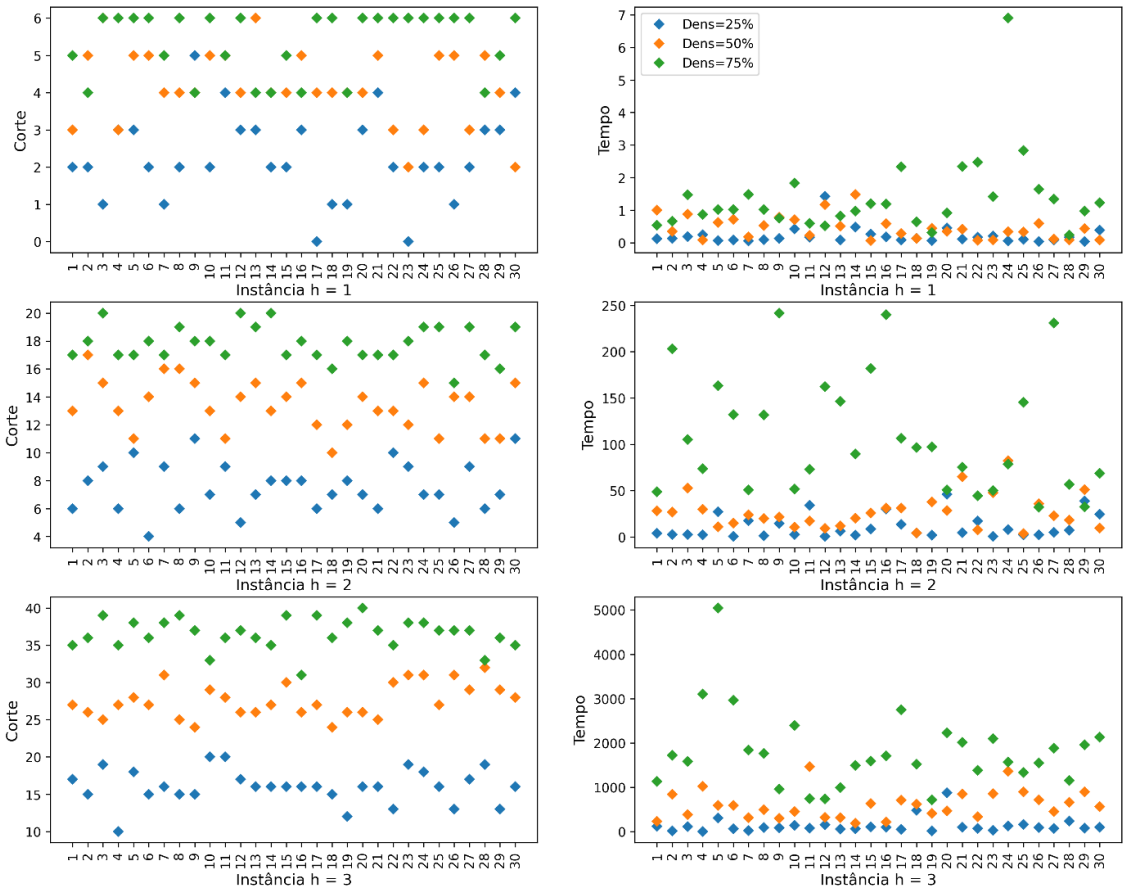
O algoritmo realiza buscas sem atingir o resultado final até a árvore 35, inicializando com $\nu^{\omega} = 38$, até quando na semente $\aleph_{1|\,2\,16}$, com $\nu^{\omega} = 29$, a árvore 36 encontra o nó pentagonal, tendo como bipartição os dois subconjuntos [[1, 8, 10, 11, 12, 14, 15, 16], [2, 3, 4, 5, 6, 7, 9, 13, 17]] induzindo ao corte máximo igual a 29. A Figura 13 c) mostra a representação do grafo biparticionado com as arestas cortadas separadas pelos dois conjuntos de vértices em lados opostos. Para melhor análise e entendimento da ABB, na próxima seção reportados os resultados dos experimentos com uma série de grafos aleatórios, primeiramente apenas para ABB e, em seguida, comparando-a com as MH.

5.2 EXPERIMENTO I: ÁRVORE DE BUSCA BINÁRIA

O experimento I mostra o funcionamento do algoritmo exato para o corte máximo na obtenção da solução de diferentes tipos de grafos aleatórios, a Figura 14 mostra os diferentes resultados para 270 grafos, que são divididos em três classes de grafos com noventa cada, sendo h = 1 grafos com cinco vértices, h = 2 grafos com nove vértices e h = 3 grafos contendo 13

vértices, cada classe é subdividida em três diferentes densidades sendo elas 25, 50 e 75%, conforme descrito na Seção 4.2.3.

Figura 14 – Resultados do corte máximo e tempo computacional em segundos do Experimento I



Fonte: O autor (2022)

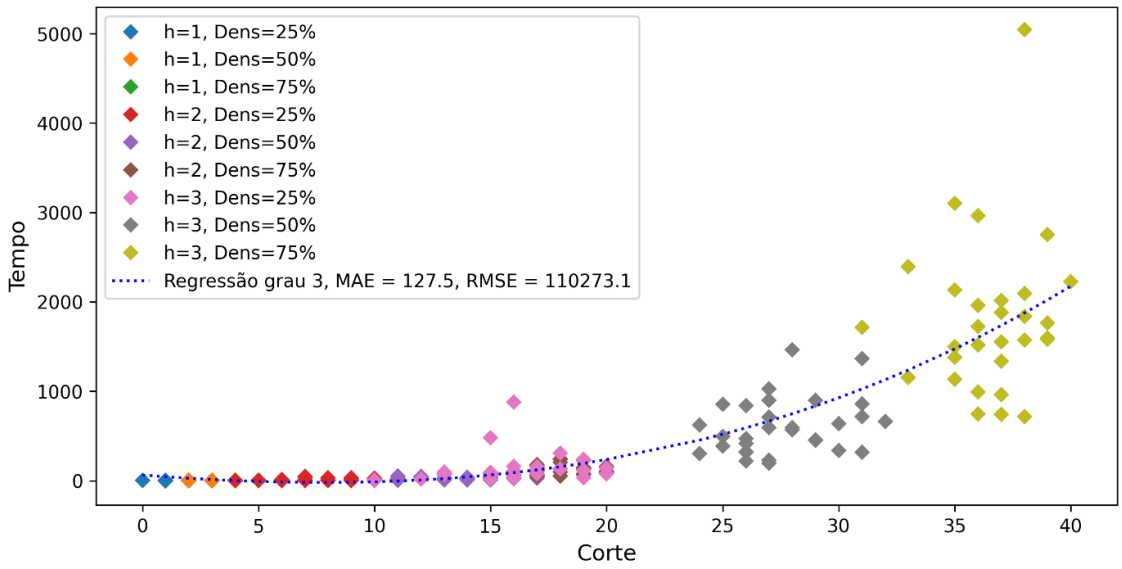
As instâncias de h = 1 mostram resultados de cortes que ficam entre 0 e 6 arestas, com tempos de processamento computacional numa média de 0,208 segundos para 25% de densidade, 0,455 e 1,385 segundos para 50 e 75% de densidade de arestas. Já as instâncias com h = 2 apresentam cortes entre 4 e 20 arestas com média de tempo computacional entre 0 e 250 segundos, esses grafos com 9 vértices, tem uma média de tempo de solução de 11,2 segundos para 25%, 26,8 segundos para 50% e 108,7 segundos para os de 75%, apresentando uma maior variabilidade para as maiores densidades. Para os grafos de 13 vértices os cortes máximos para esses ficam entre 10 e 40 arestas, com tempos de execução do algoritmo variando entre 0 a 5000 segundos, com uma média de 135, 607 e 1803 segundos respectivamente para cada densidade, sendo o caso com 5000 segundos um caso discrepante isolado.

As maiores variações de tempo de solução para as instâncias com maior densidade e tempos mais parecidos para densidades menores podem ser explicadas pelo fato de que

instâncias com menores densidades tiveram maior facilidade de encontrar nós contraditórios na árvore de busca, devido principalmente ao atributo de poda *N*, o qual avalia o número de arestas que não contribuem para o corte máximo. Os casos com maiores densidades por terem uma quantidade maior de arestas que estão ligadas a mais vértices têm uma menor probabilidade de encontrar uma combinação de subconjuntos de vértices que leve a um nó contraditório devido ao *N* na árvore.

Outra análise relevante quanto ao tempo computacional do algoritmo é apresentada na Figura 15, onde é feita uma relação entre o tamanho do corte e o tempo para encontrá-lo, essa análise mostra a ascendência dos tempos em relação aos tamanhos dos cortes. É interessante notar que em alguns casos os tamanhos dos cortes são semelhantes apesar de quantidades diferentes de vértices, como no caso dos grafos de 13 vértices h = 3 com 25% de densidade, que apresenta tempos e tamanho de cortes muito semelhantes ao grafos de h = 2 com 50 e 75% de densidade.

Figura 15 – Ascendência dos tempos em relação aos tamanhos dos cortes do experimento I



Fonte: O autor (2022)

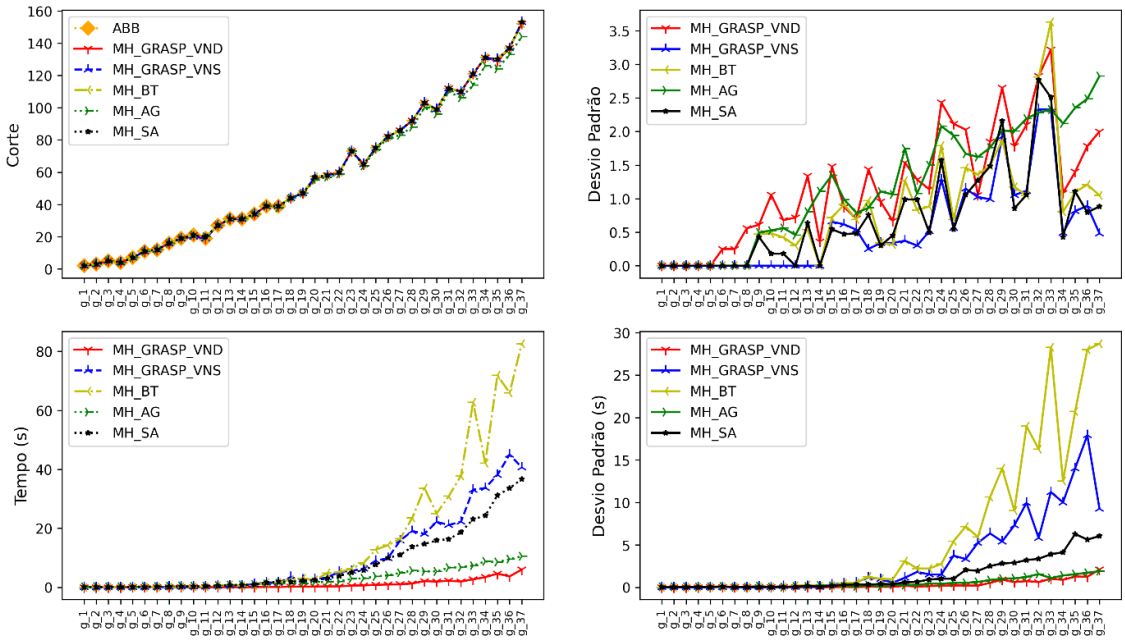
Aplicando uma regressão polinomial nesses resultados, a regressão de melhor adaptação e menor erro foi a com grau 3, com um erro absoluto médio de 127,5 e raiz quadrática média de erro igual a 110273,1. A fórmula de previsão encontrada foi $\hat{y} = 62{,}26 - 19{,}94x + 1{,}04x^2 + 0{,}2x^3$, em que $\hat{y}$ representa o tempo de previsão de solução e x o tamanho do corte. Para um caso de h = 10 em grafos de 41 vértices, por exemplo, com um corte com 350 arestas, o algoritmo demoraria em torno de 264 horas ou 11 dias, outro caso com h = 100 o tempo de solução previsto seria de 27 anos, esses tempos apresentam uma inviabilidade em casos de

grande porte na utilização desse algoritmo para instâncias de grande porte, para isso é necessário avaliar o algoritmo comparando-o com meta-heurísticas no experimento a seguir.

5.3 EXPERIMENTO II: ÁRVORE DE BUSCA BINÁRIA E META-HEURÍSTICAS

O experimento II traz uma comparação entre o algoritmo exato proposto e as cinco meta-heurísticas. São 111 instâncias, divididas em três grupos de grafos com densidades 25, 50 e 75% cada um com 37 grafos indo de h = 1 até h = 10, começando com 5 vértices e indo até 41 sedo executados 30 vezes. O resultado do primeiro grupo de densidade igual a 25% é apresentado na Figura 16, contendo as informações de melhor corte, tempo médio e desvio padrão para ambos, com exceção do tempo e desvios padrão para ABB.

Figura 16 – Resultados do corte máximo, tempo computacional e desvio padrão do experimento II para grafos com 25% de densidade



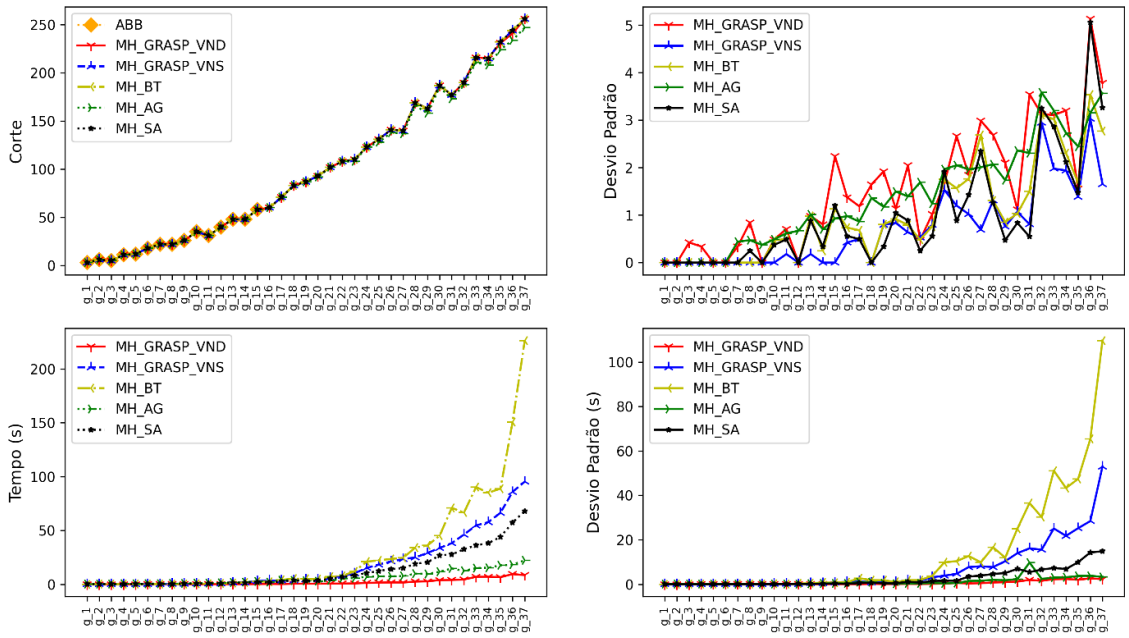
Fonte: O autor (2022)

O algoritmo da árvore de busca binária se mostrou eficiente na solução em relação ao tempo para solucionar grafos de até 22 vértices com um tempo computacional de 12266 segundos. As meta-heurísticas por outro lado se mostram capazes de atingir o resultado ótimo para as mesmas instâncias com pouco menos de 2 segundos. As meta-heurísticas encontram resultados para o corte máximo igual para todos os casos em que o método das árvores de busca também encontra, porém para as instâncias maiores os melhores valores de cortes obtidos só não são atingidos em todas as instâncias pela GRASP-VND e AG, que em alguns casos ficou um pouco abaixo.

O tempo começa a apresentar um crescimento a partir da instância g_18, com a GRASP-VND com os melhores tempos, seguido pelo AG, SA, GRASP-VNS e BT. Apesar da meta-heurística GRASP-VND ter os melhores tempos, ela não encontra o melhor resultado em todos os grafos e apresenta uma maior variabilidade nos desvios padrão do corte em relação as outras meta-heurísticas. Isso é consequência da fase de busca de vizinhança VND ser mais simples e não exigir grandes diversificações durante as buscas.

A Figura 17 mostra os resultados para os grafos com 50% de densidade.

Figura 17 – Resultados do corte máximo, tempo computacional e desvio padrão do Experimento II para grafos com 50% de densidade



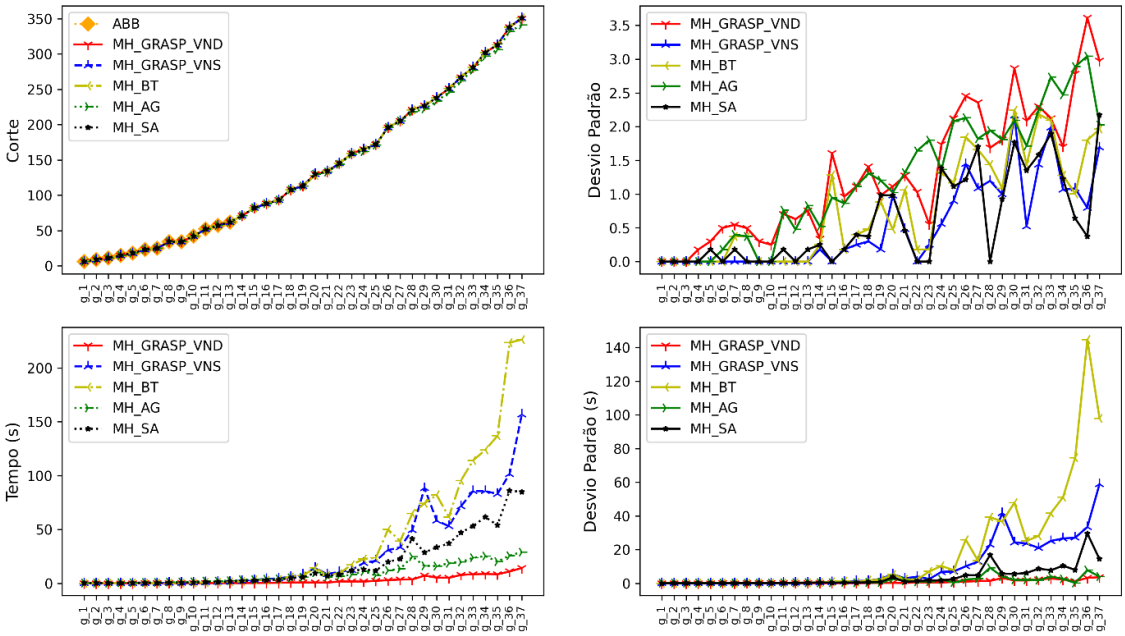
Fonte: O autor (2022)

Neste caso, as meta-heurísticas foram eficientes em encontrar o corte máximo ótimo para todos os casos em que a árvore de busca não excedeu um limite de cem mil segundos, para o último grafo em que o limite não foi excedido tendo 22 vértices a ABB encontrou o corte máximo em 76726 segundos, já as MH não excederam 3 segundos para encontrar o mesmo corte. Os tempos das MH começam a ter uma maior ascendência a partir do grafo g_22, no geral a GRASP-VND tem os melhores tempos com baixos desvios padrões, porém não atinge o melhor resultado em todos os casos e tem uma das maiores variabilidades nos desvios padrão do corte. Já a BT, além de ter os maiores tempos, tem também os maiores desvios padrão levando assim a uma alta variabilidade nos tempos desse algoritmo; o benefício é que esta MH

foi capaz de encontrar os melhores cortes com um baixo desvio padrão em relação as outras MH.

Os resultados das últimas instâncias com 75% de densidade são mostrados na Figura 18. Neste caso, a ABB foi capaz de resolver até a instância g_13, um grafo com 18 vértices, obtendo um corte de 62 arestas em um tempo de pouco menos de 38 mil segundos, após isso os resultados excederam 100 mil segundos. As MH por outro lado foram capazes de encontrar o ótimo nos mesmos casos com um tempo pouco mais de 2 segundos.

Figura 18 – Resultados do corte máximo, tempo computacional e desvio padrão do Experimento II para grafos com 75% de densidade



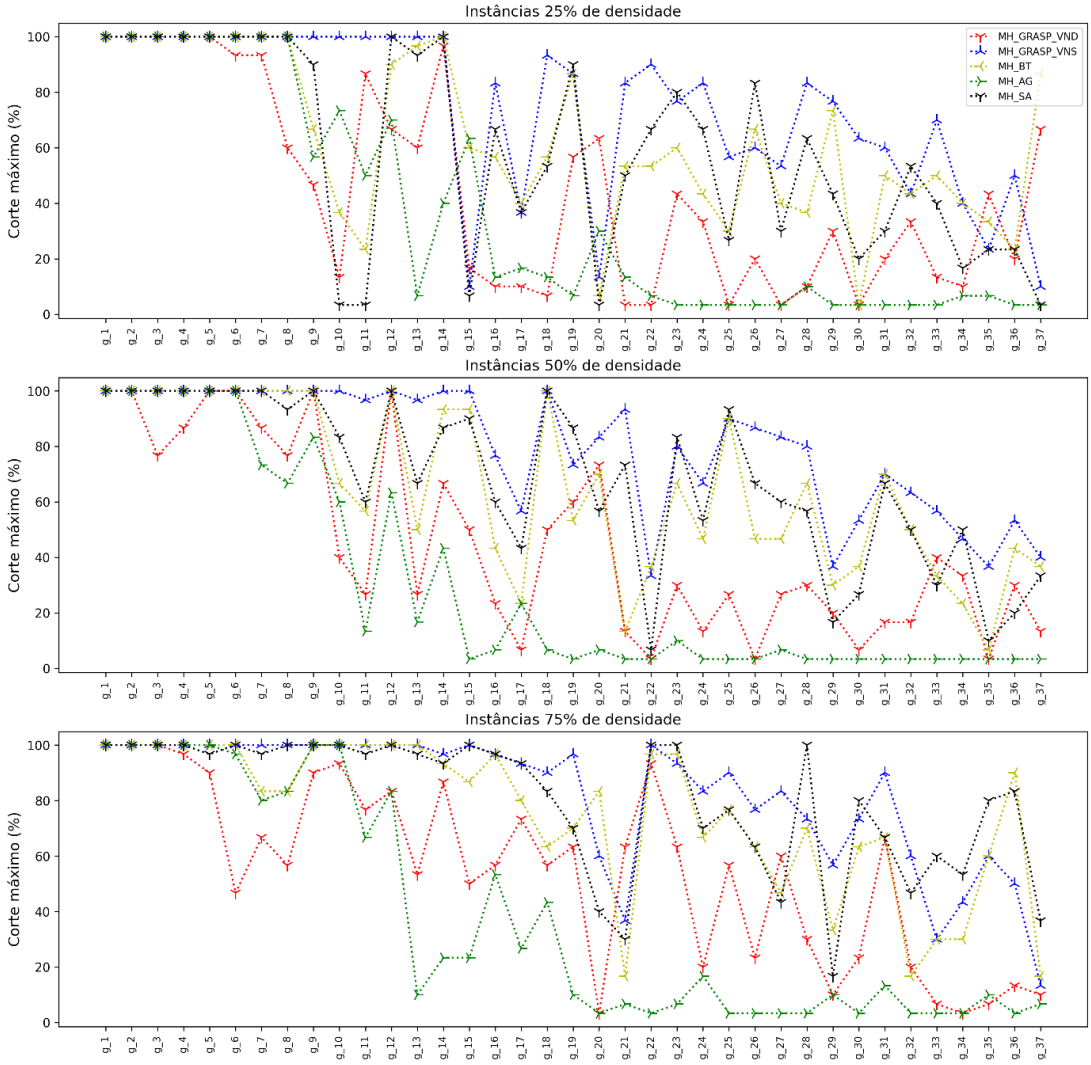
Fonte: O autor (2022)

As MH seguem o mesmo padrão das instâncias com densidades menores, a GRASP-VND e os AG, apresentam os menores tempos de solução com baixas variabilidades, essas maiores velocidades de soluções levam a essas MH não encontrarem os melhores resultados em todos os casos e terem as maiores variabilidades e pouca consistência de encontrar cortes de tamanhos similares em diferentes execuções do algoritmo. Já as MH que obtiveram os melhores resultados de corte e menor variabilidade foram a BT e a GRASP-VNS, isso com a consequência de maiores tempos de execução com maiores variabilidades mostradas pelo desvio padrão do tempo, isto resulta dessas meta-heurísticas utilizarem técnicas mais refinadas e com maiores diversificações e intensificações na fase de busca.

Para uma melhor visualização da eficiência das MH para encontrar os melhores resultados, foi avaliada uma relação percentual entre a quantidade de vezes que a MH foi capaz

de encontrar o melhor resultado em cada uma das trinta execuções do algoritmo para cada grafo. A Figura 19 apresenta essas relações percentuais nas instâncias com densidade de 25%, 50% e 75%.

Figura 19 – Acurácia das meta-heurísticas na obtenção do melhor corte na execução dos algoritmos no experimento II



Fonte: O autor (2022)

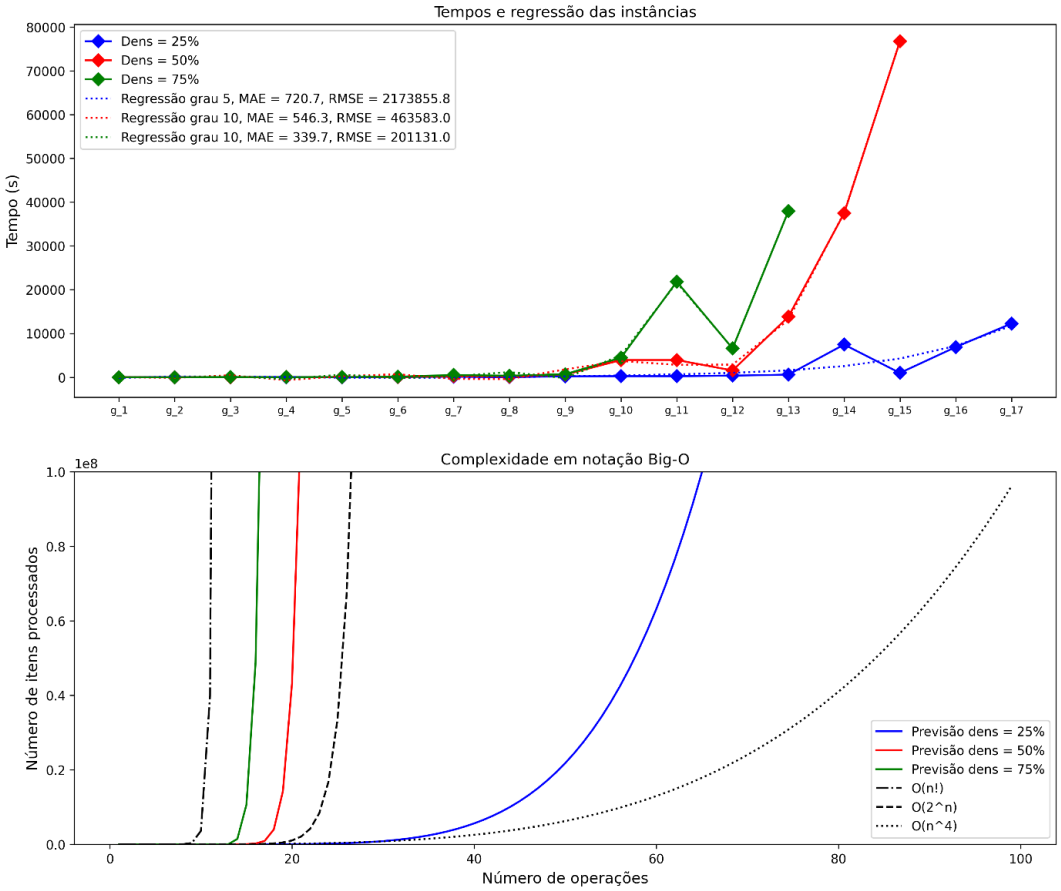
Para as instâncias com h = 1 e 2, ou g_1 até g_5, as MH apresentam uma acurácia de 100% na maioria dos casos com exceção da GRASP-VND e SA (dens. 75%). Porém para as instâncias maiores o nível de acurácia das MH começam a ter um maior declínio principalmente com os algoritmos genéticos e a GRASP-VND. As MH com os melhores desempenhos na maior parte dos casos são primeiramente a GRASP-VNS seguida pelas MH simulated anealling e busca tabu.

Algoritmos genéticos apresentam boas acurácias em instâncias de pequeno porte, a partir de grafos com 16 vértices ele apresenta maiores dificuldades de encontrar melhores

resultados, isto se deve as características estabelecidas para o tamanho populacional e quantidades de gerações no procedimento do algoritmo, pois maiores populações e mais gerações implicariam em tempos muito elevados em comparação as outras MH, com as características seguidas nesse experimento ele apresentou boa eficiência no tempo mas uma baixa acurácia e até mesmo não atingindo o melhor corte em muitos casos.

Devido aos tempos da ABB serem muito elevados comparados as MH a Figura 20 apresenta os tempos do algoritmo exato proposto nessa pesquisa para cada uma das densidades de arestas dos grafos, sendo realizado também regressões polinomiais de melhor adaptação para cada um dos casos, para densidade de 25% a regressão teve um grau 5 com um erro médio absoluto de 720, já os grafos com 50% e 75% de densidade a regressão de melhor adaptação foi a com grau 10, com erros médios absolutos de 546,3 e 339,7 respectivamente e a partir dessas regressões foram traçadas previsões com uma comparação entre as complexidades da notação Big-O.

Figura 20 – Tempos e complexidades na notação Big-O do algoritmo exato no experimento II



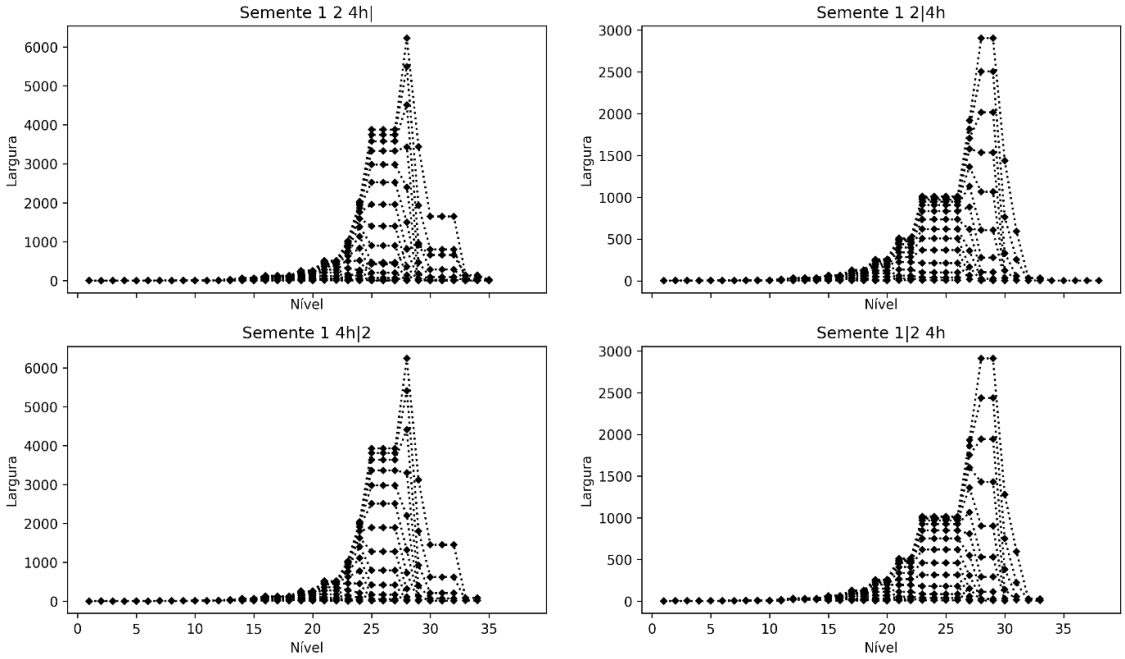
Fonte: O autor (2022)

As previsões das regressões mostram que a complexidade da árvore de busca binária para grafos com 75% de densidade ficam entre uma das maiores complexidades da notação

big-O que é $O(n!)$ e $O(2^n)$. Para os grafos de 50% de densidade eles ficam pouco abaixo dos de 75%, mais próximos da notação de $O(2^n)$. Por fim o caso mais promissor para as árvores de buscas são os grafos com baixas densidades como 25%, nesse caso a complexidade fica muito abaixo de $O(2^n)$ e um pouco mais próximo de $O(n^4)$.

Um dos principais motivos da ABB ter tempos tão elevados é que as alturas das árvores de busca são polinomiais, limitada pelas variáveis duais retornadas da solução dual do programa linear mestre, mas a largura das árvores de busca começam a atingir grandes níveis. No caso do grafo g_15 com 50% de densidade, que teve o maior tempo de processamento das árvores de busca com mais de 76 mil segundos, foram geradas 101 árvores no total, a semente 1 2 4h| gerou 26 árvores com uma largura de até 6 mil nós, as demais sementes geraram 25 árvores cada, com uma largura de até 6 mil nós para a semente 1 4h| 2, e de até 3 mil nós nas sementes 1 2| 4h e 1| 2 4h, apresentados na Figura 21.

Figura 21 – Larguras das árvores de busca do grafo g_15 com 50% de densidade



Fonte: O autor (2022)

O grande desafio para esse algoritmo é encontrar novos atributos de poda, como as contradições na solução das equações binárias e o *N* que avalia o número de arestas irrelevantes para o corte em cada passo na iteração de busca. Encontrando novos agentes de poda da árvore, é possível filtrar a busca em combinações de subconjuntos mais promissores para o corte máximo, reduzindo a largura da árvore e consequentemente os tempos de execução do algoritmo.

6 CONCLUSÃO

Esse trabalho apresentou uma nova abordagem exata para a resolução do problema do *max-cut*, trazendo uma fundamentação e estruturação de maneira didática de todos os procedimentos e etapas desse algoritmo que foi nomeado como árvore de busca binária. Além de toda fundamentação e descrição da abordagem, foi realizada a implementação deste algoritmo com a linguagem de programação *python* 3. Além do método exato foram implementadas cinco das principais meta-heurísticas utilizadas na otimização de problemas combinatórios, almejando identificar a eficiência na solução em relação ao método exato.

O primeiro experimento demonstrou que uma particularidade do método exato é que instâncias de grafos de menor densidade apresentam maior facilidade de encontrar nós contraditórios na árvore de busca, devido principalmente ao atributo de poda que avalia o número de arestas que não contribuem para o corte do grafo, sendo assim capazes de encontrar a solução ótima com maior rapidez, diferente de casos com maiores densidades, que por terem uma quantidade maior de arestas que estão ligadas a mais vértices tem uma menor probabilidade de encontrar uma combinação de subconjuntos de vértices que leve a um nó contraditório utilizando *N* na árvore.

Com a análise de regressão polinomial, fica claro que para aplicações em que as instâncias tenham portes menores, a abordagem exata é muito promissora pois sempre encontra as soluções ótimas para o problema, diferente das MH, que são necessárias inúmeras execuções. Porém em instâncias de grande porte a análise mostra uma inviabilidade desse método exato, com previsões de tempos de execução computacional de até 11 dias para um grafo com 41 vértices e impressionantes 27 anos para um grafo com 401 vértices. Para isso, foi necessário avaliar o algoritmo comparando com os métodos meta-heurísticos, os quais no segundo experimento deixam claro que as utilizações dessas abordagens heurísticas são capazes de encontrar os resultados para o corte máximo com tempos extremamente rápidos.

Além disso as MH se mostraram capazes de solucionar até mesmo instâncias com portes maiores, tendo acurácias maiores em instâncias de pequeno porte e acurácias menores em instâncias de maior porte. As meta-heurísticas que apresentaram melhores desempenhos na maioria dos casos foram primeiramente a GRASP-VNS, seguida pela *simulated annealing* e busca tabu. Os bons desempenhos dessas resultam em maiores tempos de execução com maiores variabilidades, fatores esses influenciados por essas meta-heurísticas utilizarem técnicas mais refinadas e com maiores diversificações e intensificações na fase de busca.

Assim para a solução do *max-cut* o método exato proposto é preferível em casos de pequeno porte, já os métodos meta-heurísticos existentes em casos de grande porte apresentam um melhor *trade-off* entre encontrar uma boa solução e o tempo necessário para tal, pois nessa primeira versão do algoritmo ABB, as complexidades ainda são muito elevadas para solucionar instâncias de grafos mais densos, isso se torna nítido com as previsões das regressões que mostram que a notação Big O fica entre $O(n!)$ e $O(2^n)$, sendo assim um grau de complexidade fatorial. Em grafos de densidades menores as complexidades são menores, mas ainda assim com uma inviabilidade para solução em tempo aceitável.

Se por um lado as alturas das árvores são limitadas pela quantidade de variáveis duais, o grande obstáculo para essa abordagem são as larguras que crescem exponencialmente na busca pela combinação da bipartição de vértices que resulta no corte máximo. Para resolver esse desafio são necessárias explorações mais aprofundadas no algoritmo para encontrar um novo critério de contradição a ser utilizado na árvore de busca binária, que seja capaz de diminuir sua largura e tornar o processo de busca mais rápido computacionalmente, impactando diretamente no tempo total de resolução requerido pelo algoritmo proposto. Consequentemente, viabilizaria a obtenção da solução ótima em tempos aceitáveis para qualquer grafo, ou pelo menos consideravelmente menor do que a abordagem tradicional.

Assim, como proposta para pesquisas futuras recomenda-se um aprofundamento nas técnicas matemáticas para limitar as larguras das árvores de busca, focando assim em caminhos mais curtos para encontrar o nó pentagonal, tornando possível filtrar a busca em combinações de subconjuntos mais promissores para o corte máximo, reduzindo dessa forma a largura da árvore e consequentemente os tempos de execução do algoritmo.

Outra proposta é trazer um maior aprimoramento nas meta-heurísticas implementadas nessa pesquisa, adicionando técnicas especiais de intensificação e diversificação como reconexação por caminhos, princípio da otimalidade próxima e relaxação adaptativa, ou ainda, utilizar técnicas de resfriamento mais complexas na MH SA, bem como utilizar métodos para definir uma temperatura inicial, ao invés de um valor fixo. Na MH AG utilizar mais indivíduos nas populações e trazer um limite de gerações que pare quando a solução não melhorar mais. Ainda é indicado implementar outras meta-heurísticas como Iterated Local Search, Colônia de Formigas e Algoritmos Meméticos.

REFERÊNCIAS

ABDEL-BASSET, M.; ABDEL-FATAH, L.; SANGAIAH, A. K. Metaheuristic Algorithms: A Comprehensive Review. Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications. p. 185–231, 2018.

AFFI, M; DERBEL, H; JARBOUI, B. Variable neighborhood search algorithm for the green vehicle routing problem. International Journal of Industrial Engineering Computations, v. 9, n. 2, p. 195–204, 2018.

AHO, A. V.; ULLMAN, J. D. Foundations of computer science. New York, NY: Computer Science Press, 1992.

ALIDAEE, B. et al. 0-1 Quadratic programming approach for optimum solutions of two scheduling problems, International Journal of Systems Science. p. 401-408, 1994.

ARAÚJO, C. V. D. Utilização de desigualdades válidas baseadas em condições de otimalidade na construção de algoritmos heurísticos para o problema do corte máximo. 2018. 60 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal do Ceará, Campus de Russas, Russas, 2018.

ARENALES, M. et al. Pesquisa Operacional. Elsevier: ABEPRO, Rio de Janeiro, 2015.

BENGIO, Y.; LODI, A.; PROUVOST, A. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. European Journal of Operational Research, 2020.

BERTRAND, J. W. M.; FRANSOO, J. C. Operations management research methodologies using quantitative modeling. International Journal of Operations & Production Management, v.22, n.2, p.241- 264, 2002.

BESTEN, M. D.; STÜTZLE, T. Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. MIC'2001–4TH METAHEURISTICS INTERNATIONAL CONFERENCE. p. 545-549, 2001

CERNY, V. Thermodynamical aproach to the traveling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory an Aplication, v. 45, n. 1, p. 41-51, 1985.

CHOPARD B., TOMASSINI M. Simulated Annealing. In: An Introduction to Metaheuristics for Optimization. Natural Computing Series. Springer, Cham, 2018.

CRAVO, G. L.; AMARAL, A. R. S. A GRASP algorithm for solving large-scale single row facility layout problems. American Journal of Operations Research , v. 106. p. 49-61, 2019.

DELAHAYE D., CHAIMATANAN S., MONGEAU M. Simulated Annealing: From Basics to Applications. In: GENDREAU M., POTVIN JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 272. Springer, Cham, 2019.

DELGADO-ANTEQUERA, L., CABALLERO, R., SÁNCHEZ-ORO, J., COLMENAR, J. M., & MARTÍ, R. Iterated greedy with variable neighborhood search for a multiobjective waste collection problem. Expert Systems with Applications, 145, 113101, 2020.

DELLA C. F.; KAMINSKI, M. J.; PASCHOS, V. T. An exact algorithm for MAX-CUT in sparse graphs. Operations Research Letters. p. 403–408, 2007.

DING, C. H. Q. et al. A min-max cut algorithm for graph partitioning and data clustering. Proceedings 2001 IEEE International Conference on Data Mining, p. 107-114, San Jose, CA, USA. IEEE Comput. Soc, 2001.

DOANE, D. P.; SEWARD, L. E. Estatística aplicada à administração e economia. 4ed. Porto Alegre: AMGH, 2014.

DUNNING, I.; GUPTA, S.; SILBERHOLZ, J. What works best when? a systematic evaluation of heuristics for max-cut and QUBO. INFORMS Journal on Computing, 2018.

ERDÓS, P. On bipartite subgraphs of graphs. Mat. Lapok. p. 283–288, 1967.

FARIA, L. et al. Maximum cuts in edge-colored graphs. Discrete Applied Mathematics, v. 281. p. 229-234, 2020.

FEO T.A.; RESENDE M.G.C. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters. p. 67–71, 1989.

FESTA, P. et al. GRASP and VNS for max-cut. Extended Abstracts of the Fourth Metaheuristics International Conference. p. 371–376, 2001.

FESTA, P. et al. Randomized heuristics for the max-cut problem. Optimization methods and software, Taylor & Francis, v. 17, n. 6. p. 1033–1058, 2002.

FRANZIN, A., STÜTZLE, T. Revisiting simulated annealing: A component-based analysis. Computers & Operations Research, 104, 191–206, 2019.

GALLO G.; HAMMER P.L.; SIMEONE B. Quadratic knapsack problems. Padberg M.W. (eds) Combinatorial Optimization. Mathematical Programming Studies, vol 12. Springer, Berlin, Heidelberg, 1980.

GAMRATH, G. et al. The SCIP Optimization Suite 7.0, ZIB-Report 20-10, Zuse Institute Berlin, 2021.

GAREY, M. R.; JOHNSON, D. S. Computers and Intractability: A guide to the theory of NP-completeness. W. H. Freeman, New York, 1979.

GIL, A. C. Métodos e técnicas de pesquisa social. 7. ed. São Paulo: Atlas, 2019.

GLOVER, F. Tabu search–part I. ORSA J Comput. p.190–206, 1989.

GOEMANS, M. X.; WILLIAMSON, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM, v. 42, n. 6. p. 1115–1145, 1995.

GOLDBARG, M. C.; LUNA, H. P. L. Otimização combinatória e programação linear: modelos e algoritmos (2ª ed.). Elsevier, Rio de Janeiro, 2005.

GUERRESCHI, G.G.; MATSUURA, A.Y. QAOA for Max-Cut requires hundreds of qubits for quantum speed-up. Scientific Reports - Nature, 2019.

HADLOCK, F. Finding a Maximum Cut of a Planar Graph in Polynomial Time, SIAM J. COMPUT. V. 4 (3), p. 221–225, 1975.

HAMER, P. The max-cut problem and quadratic 0-1 optimization; polyhedral aspects, relaxations and bounds. Annals of Operations Research, v. 33, n. 3. p. 151–180, 1991.

HANSEN, P. et al. Variable neighborhood search. Handbook of metaheuristics. Springer, Cham. p. 57-97, 2019.

HANSEN, P. et al. Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, v. 5, n. 3, p. 423-454, 2017.

HANSEN, P., MLADENOVIĆ, N., TODOSIJEVIĆ, R., & HANAFI, S. Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, 423–454, 2016.

HILLIER, F. S.; LIEBERMAN, G. J. Introdução à pesquisa operacional, 9 ed. Porto Alegre: AMGH, 2013.

HOLLAND, J.H. Genetic algorithms. Scientific American. p. 66–72, 1992.

HOU, N. et al. An efficient GPU-based parallel tabu search algorithm for hardware/software co-design. Frontiers of Computer Science, 2020.

HRGA, T.; POVH, J. MADAM: a parallel exact solver for max-cut based on semidefinite programming and ADMM. Computational Optimization and Applications, 80, 347–375, 2021.

HUSSAIN, K. et al. Metaheuristic research: a comprehensive survey. Artificial Intelligence Review 52, p. 2191–2233, 2019.

JABAL-AMELI, M. S.; ARYANEZHAD, M. B.; GHAFFARI-NASAB, N. A variable neighborhood descent based heuristic to solve the capacitated location-routing problem. International Journal of Industrial Engineering Computations, v. 2, n. 1, p. 141-154, 2011.

JAFAR-ZANJANI, S.; INAMPUDI, S.; MOSALLAEI, H. Adaptive Genetic Algorithm for Optical Metasurfaces Design. Scientific Reports, 2018.

KARAKOSTAS, P., SIFALERAS, A., & GEORGIADIS, M. C. Adaptive variable neighborhood search solution methods for the fleet size and mix pollution location-inventory-routing problem. Expert Systems with Applications, 113444, 2020.

KARP, R. M. Reducibility among Combinatorial Problems. Complexity of Computer Computations - Springer, 1972.

KHAJEHZADEH, M. et al. A survey on meta-heuristic global optimization algorithms. research journal of applied science engineering and technology 3(6): p. 569–578, 2011.

KIM, S.H.; KIM, Y.H.; MOON, B.R. A hybrid genetic algorithm for the max cut problem. MORGAN KAUFMANN PUBLISHERS INC. Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation. p. 416–423, 2001.

KIM, Y.H.; YOON, Y.; GEEM, Z. W. A comparison study of harmony search and genetic algorithm for the max-cut problem. Swarm and Evolutionary Computation, 2018.

KIRKPATRICK Jr., S., GELATT, C.; VECCHI, M. Optimization by simulated annealing. Decision Science, v. 220, n. 4598, p. 498-516, 1983.

KOCHENBERGER, G. A. et al. Solving large scale max cut problems via tabu search. Journal of Heuristics, Springer, v. 19, n. 4, p. 565–571, 2013.

KRARUP J., PRUZAN P.M. Computer-aided layout design. In: Balinski M.L., Lemarechal C. (eds) Mathematical Programming in Use. Mathematical Programming Studies, vol 9. Springer, Berlin, Heidelberg, 1978.

KRISLOCK, N. et al. Improved semidefinite bounding procedure for solving max-cut problems to optimality. Mathematical Programming, v. 143, n. 1–2, p. 61–86, 2014.

LAKATOS, E. M.; MARCONI, M. de A. Fundamentos de metodologia científica. 8. ed. São Paulo: Atlas, 2017.

LI, K.; TIAN, H. A de-based scatter search for global optimization problems. Discrete Dynamics in Nature and Society, 2015.

LIMA, G. L; MELO, I. E. S. DE; LINS, S. L. S. PROBLEMA DO CORTE MÁXIMO SIMPLES: Implementação de uma nova abordagem exata com árvores de busca e da meta-heurística GRASP-VND. In: ENEGEP 2021 Encontro Nacional de Engenharia de Produção, 2021, Online, 2021.

LÓPEZ-SÁNCHEZ, A. D.; SÁNCHEZ-ORO, J.; HERNÁNDEZ-DÍAZ, A. G. GRASP and VNS for solving the p-next center problem. American Journal of Operations Research , v. 104, p.295-303, 2019.

LÜDKE, M.; ANDRÉ, M. D. A. A Pesquisa em educação: abordagens qualitativas. 2.ed. São Paulo: EPU, 2013.

MANSOUR, N.; AWAD, M.; EL-FAKIH, K. Incremental genetic algorithm. The International Arab Journal of Information Technology, v. 3, n. 1, p. 42–47, 2006.

MARINHO, M. R. Otimização do problema de reconfiguração de sistemas de distribuição de energia elétrica por meio das Meta-Heurísticas Busca Tabu, GRASP e Path Relinking. 2020. Tese de doutorado (Doutorado em Engenharia Elétrica) - Universidade Estadual Paulista (UNESP), São Paulo, 2020.

METAWA, N., HASSAN, M. K., & ELHOSENY, M. Genetic algorithm based model for optimizing bank lending decisions. Expert Systems with Applications, 80, p. 75–82, 2017.

METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A., TELLER, E., Equation of state calculations by fast computing machines, Journal of Chemical Physics 21. 1087–1092, 1953.

MIRJALILI, S. Genetic Algorithm. Evolutionary Algorithms and Neural Networks. Studies in Computational Intelligence, vol 780. Springer, Cham, 2019.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. Computers & operations research, v.24, n. 11, p. 1097-1100, 1997.

MLADENOVIC, N; HANSEN, P. Variable neighborhood search. Computers & operations research, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

MOHADES, M. M.; KAHAEI, M. H. An Efficient Riemannian Gradient based Algorithm for Max-Cut Problems. in *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

NETTO, P. O. B.; JURKIEWICZ, S. Grafos: introdução e prática. Editora Blucher, ed. 2, 2017.

OTTERBACH, J. S. et al. Unsupervised Machine Learning on a Hybrid Quantum Computer. arXiv Quantum Physics, 2017.

PHAM, D. T; HUYNH T. T. B. An effective combination of genetic algorithms and the variable neighborhood search for solving travelling salesman problem. Conference on technologies and applications of artificial intelligence (TAAI), IEEE, pp 142–149, 2015.

PINHO, A. F. et al. Algoritmos genéticos: fundamentos e aplicações. Meta-heurísticas em pesquisa operacional. Curitiba: Omnipax. p. 21-32, 2013.

POLJAK S.; TUZA Z. Combinatorial Optimization, Vol. 20, American Mathematical Society, 1995.

PRESTES, E. Introdução à Teoria dos Grafos. Universidade Federal do Rio Grande do Sul, Instituto de Informática, Departamento de Informática Teórica, Technical Reports, 2016.

RAHMANI HOSSEINABADI, A. A. et al. Extended Genetic Algorithm for solving open-shop scheduling problem. Soft Computing 23, p. 5099–5116, 2019.

RENDL, F.; RINALDI, G.; WIEGELE, A. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming, 121(2), 307–335, 2008.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. Handbook of metaheuristics. Springer, Boston, 2010.

RESENDE, M. G. C; RIBEIRO, C. C. Greedy randomized adaptive search procedures: advances and extensions. Handbook of metaheuristics. Springer, Cham, 2019.

RIBEIRO, M. H.; PLASTINO, A.; MARTINS, S. L. Hybridization of GRASP metaheuristic with data mining techniques. Journal of Mathematical Modelling and Algorithms, v. 5, n. 1, p. 23-41, 2006.

SAHLI, Z. et al. Hybrid PSO-tabu search for the optimal reactive power dispatch problem. IECON 2014-40th annual conference of the IEEE industrial electronics society, IEEE, p. 3536–3542, 2014.

SOHRABI, S.; ZIARATI, K.; KESHTKARAN, M. A Greedy Randomized Adaptive Search Procedure for the Orienteering Problem with Hotel Selection. European Journal of Operational Research, v. 283, n. 2, p. 426-440, 2020.

SUCUPIRA, R. et al. Maximum Cuts in Edge-colored Graphs. Electronic Notes In Discrete Mathematics, v. 62, p. 87-92, 2017.

WANG, R., WANG, L. Maximum cut in fuzzy nature: Models and algorithms. Journal of Computational and Applied Mathematics, p. 240-252, 2010.

WANG, Y. A sociopsychological perspective on collective intelligence in metaheuristic computing. international journal of applied metaheuristic computing, p. 110–128, 2010.

WAZLAWICK, R.S. Metodologia de pesquisa para Ciência da Computação, 3.ed. . São Paulo : GEN LTC . 2020.

WHITLEY, D. Next Generation Genetic Algorithms: A User's Guide and Tutorial. Handbook of Metaheuristics. International Series in Operations Research & Management Science, v. 272. 3.ed.Cham: Springer. p. 245-274, 2019.

WU, Q.; HAO, J.-K. A memetic approach for the max-cut problem. In: SPRINGER. International Conference on Parallel Problem Solving from Nature, p. 297–306, 2012.

YANG, X. S. et al. Attraction and diffusion in nature-inspired optimization algorithms. Neural Computing and Applications, p. 1–8, 2015.

YANG, X. S; DEB, S. Cuckoo search: recent advances and applications. Neural Computing and Applications, p. 169–174, 2014.

YOUSSEF, H.; SAIT, S. M.; ADICHE, H. Evolutionary algorithms, simulated annealing end tabu search: a comparative study. Engineering Aplications of Artificial Inteligence, v. 14, p. 167-181, 2001.

ZHANG, H. et al. A Hybrid Method Integrating an Elite Genetic Algorithm with Tabu Search for the Quadratic Assignment Problem. Information Sciences, 2020.

Apêndice A – Resultados do tamanho do corte e tempos do Experimento I

Instancias corte e tempo computacional

	H = 1 V = (1,..,5)						H = 2 V = (1,..,9)						H = 3 V = (1,..,13)					
	25%		50%		75%		25%		50%		75%		25%		50%		75%	
id	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo	Corte	Tempo
1_1	2	0.122	3	0.999	5	0.542	6	3.942	13	28.323	17	49.0	17	123.8	27	231.1	35	1135.5
1_2	2	0.136	5	0.347	4	0.663	8	2.722	17	26.94	18	203.0	15	17.8	26	840.3	36	1724.9
1_3	1	0.193	6	0.875	6	1.472	9	2.771	15	52.916	20	105.3	19	113.5	25	385.9	39	1581.1
1_4	3	0.245	3	0.086	6	0.868	6	2.307	13	30.053	17	73.6	10	5.5	27	1024.4	35	3104.2
1_5	3	0.069	5	0.624	6	1.016	10	27.132	11	11.12	17	163.5	18	308.7	28	590.5	38	5044.5
1_6	2	0.082	5	0.715	6	1.017	4	0.562	14	15.13	18	132.2	15	68.8	27	593.6	36	2964.4
1_7	1	0.056	4	0.178	5	1.48	9	17.606	16	23.809	17	50.8	16	25.2	31	318.0	38	1837.9
1_8	2	0.092	4	0.532	6	1.019	6	1.357	16	20.105	19	131.7	15	90.4	25	496.1	39	1762.2
1_9	5	0.128	4	0.785	4	0.755	11	14.512	15	21.527	18	241.7	15	89.4	24	302.2	37	962.0
1_10	2	0.426	5	0.709	6	1.83	7	2.55	13	10.715	18	51.6	20	142.4	29	454.7	33	2393.2
1_11	4	0.171	5	0.228	5	0.594	9	34.277	11	17.276	17	73.0	20	78.0	28	1464.4	36	746.7
1_12	3	1.422	4	1.175	6	0.518	5	0.631	14	9.265	20	162.3	17	152.7	26	320.9	37	740.4
1_13	3	0.085	6	0.507	4	0.818	7	6.309	15	12.129	19	146.4	16	61.8	26	315.2	36	992.1
1_14	2	0.478	4	1.481	4	0.97	8	1.971	13	20.125	20	89.5	16	64.7	27	194.1	35	1496.5
1_15	2	0.269	4	0.066	5	1.196	8	8.701	14	25.941	17	181.8	16	104.6	30	637.2	39	1594.3
1_16	3	0.177	5	0.583	4	1.194	8	30.093	15	31.184	18	240.1	16	102.6	26	219.4	31	1712.3
1_17	0	0.081	4	0.28	6	2.333	6	13.653	12	31.352	17	106.6	16	52.8	27	712.0	39	2751.3
1_18	1	0.131	4	0.132	6	0.642	7	4.418	10	4.355	16	96.7	15	481.2	24	620.5	36	1519.3
1_19	1	0.064	4	0.44	4	0.312	8	1.997	12	38.024	18	97.3	12	19.1	26	415.4	38	717.4
1_20	3	0.453	4	0.352	6	0.919	7	46.056	14	28.512	17	51.0	16	880.2	26	469.0	40	2227.8
1_21	4	0.109	5	0.417	6	2.343	6	4.57	13	65.208	17	75.2	16	103.4	25	852.0	37	2016.4
1_22	2	0.172	3	0.078	6	2.47	10	17.302	13	7.741	17	44.5	13	72.8	30	335.3	35	1381.7
1_23	0	0.204	2	0.085	6	1.416	9	0.723	12	47.715	18	50.1	19	32.2	31	857.0	38	2094.1
1_24	2	0.058	3	0.344	6	6.9	7	8.138	15	82.218	19	78.7	18	131.4	31	1364.1	38	1573.5
1_25	2	0.101	5	0.333	6	2.832	7	2.387	11	3.714	19	145.3	16	162.2	27	898.4	37	1337.1
1_26	1	0.037	5	0.596	6	1.639	5	2.306	14	35.834	15	32.3	13	95.4	31	715.1	37	1552.1
1_27	2	0.087	3	0.11	6	1.343	9	4.964	14	23.014	19	231.1	17	75.7	29	452.0	37	1880.2
1_28	3	0.167	5	0.086	4	0.236	6	7.247	11	18.274	17	56.8	19	236.7	32	661.6	33	1151.6
1_29	3	0.042	4	0.435	5	0.975	7	38.812	11	51.007	16	32.6	13	78.9	29	898.9	36	1961.4
1_30	4	0.385	2	0.087	6	1.231	11	24.689	15	9.567	19	68.8	16	99.2	28	568.4	35	2132.8
Média tempo	-	0.208	-	0.455	-	1.385	-	11.2	-	26.8	-	108.7	-	135.7	-	606.9	-	1802.9

Apêndice B – Resultados do tamanho do corte das instâncias com 25% de densidade do Experimento II

Problema			Valores dos cortes														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_25	5	25	2	2	2	0,0	2	2	0,0	2	2	0,0	2	2	0,0	2	2	0,0
g_2_25	6	25	3	3	3	0,0	3	3	0,0	3	3	0,0	3	3	0,0	3	3	0,0
g_3_25	7	25	5	5	5	0,0	5	5	0,0	5	5	0,0	5	5	0,0	5	5	0,0
g_4_25	8	25	4	4	4	0,0	4	4	0,0	4	4	0,0	4	4	0,0	4	4	0,0
g_5_25	9	25	7	7	7	0,0	7	7	0,0	7	7	0,0	7	7	0,0	7	7	0,0
g_6_25	10	25	11	11	10	0,25	11	11	0,0	11	11	0,0	11	11	0,0	11	11	0,0
g_7_25	11	25	12	12	11	0,25	12	12	0,0	12	12	0,0	12	12	0,0	12	12	0,0
g_8_25	12	25	16	16	15	0,56	16	16	0,0	16	16	0,0	16	16	0,0	16	16	0,0
g_9_25	13	25	19	19	18	0,61	19	19	0,0	19	18	0,47	19	18	0,5	19	18	0,43
g_10_25	14	25	21	21	19	1,06	20	20	0,0	21	20	0,48	21	20	0,53	21	20	0,18
g_11_25	15	25	19	19	18	0,68	19	19	0,0	20	19	0,42	20	19	0,56	20	19	0,18
g_12_25	16	25	27	27	26	0,72	27	27	0,0	27	26	0,3	27	26	0,46	27	27	0,0
g_13_25	17	25	31	31	29	1,34	31	31	0,0	31	30	0,54	31	28	0,8	31	30	0,64
g_14_25	18	25	31	31	30	0,36	31	31	0,0	31	31	0,0	31	29	1,11	31	31	0,0
g_15_25	19	25	34	34	32	1,48	34	32	0,66	34	33	0,72	34	33	1,35	34	32	0,54
g_16_25	20	25	39	39	37	0,87	39	38	0,62	39	38	0,91	39	37	0,99	39	38	0,47
g_17_25	21	25	39	39	37	0,7	39	38	0,54	39	38	0,69	38	36	0,79	39	38	0,48
g_18_25	22	25	-	44	41	1,44	44	43	0,25	44	43	0,97	44	42	0,86	44	43	0,76
g_19_25	23	25	-	47	46	0,96	47	46	0,34	47	46	0,34	47	44	1,11	47	46	0,3
g_20_25	24	25	-	56	55	0,67	57	56	0,34	57	56	0,31	55	53	1,06	57	55	0,45
g_21_25	25	25	-	58	55	1,54	58	57	0,37	58	57	1,27	57	54	1,75	58	57	0,99
g_22_25	26	25	-	60	57	1,28	60	59	0,3	60	59	0,83	59	56	1,08	60	59	0,99
g_23_25	27	25	-	73	72	1,14	73	72	0,51	73	72	0,88	73	69	1,5	73	72	0,5
g_24_25	28	25	-	65	63	2,43	65	64	1,28	65	63	1,79	64	59	2,08	65	64	1,57
g_25_25	29	25	-	75	72	2,12	75	74	0,56	75	74	0,69	74	69	1,94	75	74	0,54
g_26_25	30	25	-	82	79	2,03	82	81	1,15	82	81	1,46	81	76	1,67	82	81	1,06
g_27_25	31	25	-	86	82	1,05	86	85	1,02	86	84	1,36	83	79	1,62	86	84	1,27
g_28_25	32	25	-	92	87	1,85	92	91	0,99	92	90	1,49	88	84	1,76	92	90	1,48
g_29_25	33	25	-	103	99	2,65	103	101	1,92	103	101	1,87	100	95	2,01	103	100	2,16
g_30_25	34	25	-	99	95	1,78	99	98	1,05	99	97	1,18	96	90	2,01	99	97	0,85
g_31_25	35	25	-	112	109	2,1	112	111	1,12	112	110	1,05	110	103	2,19	112	110	1,06
g_32_25	36	25	-	110	106	2,83	110	107	2,33	110	106	2,81	106	98	2,28	110	107	2,77
g_33_25	37	25	-	121	115	3,22	121	119	2,33	121	118	3,63	114	108	2,33	121	118	2,51
g_34_25	38	25	-	131	129	1,08	131	130	0,49	131	130	0,8	126	121	2,12	131	130	0,43
g_35_25	39	25	-	129	127	1,4	130	128	0,81	130	128	1,09	124	118	2,35	130	128	1,11
g_36_25	40	25	-	137	134	1,78	137	136	0,89	137	135	1,21	133	125	2,49	137	135	0,8
g_37_25	41	25	-	152	150	2,0	153	152	0,48	152	151	1,05	144	138	2,82	153	151	0,88

Apêndice C – Resultados do tempo computacional das instâncias com 25% de densidade do Experimento II

Problema			Tempo computacional														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_25	5	25	0.1	0.0	0.0	0.0	0.0	0.01	0.0	0.0	0.0	0.0	0.08	0.14	0.05	0.01	0.02	0.01
g_2_25	6	25	3.48	0.0	0.0	0.0	0.0	0.01	0.01	0.0	0.01	0.01	0.11	0.17	0.05	0.01	0.02	0.0
g_3_25	7	25	4.6	0.0	0.0	0.0	0.01	0.01	0.01	0.01	0.01	0.01	0.11	0.18	0.05	0.02	0.04	0.02
g_4_25	8	25	0.49	0.0	0.0	0.0	0.01	0.04	0.01	0.01	0.02	0.01	0.11	0.16	0.03	0.02	0.04	0.02
g_5_25	9	25	1.3	0.0	0.0	0.01	0.03	0.04	0.01	0.02	0.03	0.01	0.13	0.17	0.05	0.05	0.08	0.03
g_6_25	10	25	57.13	0.0	0.01	0.01	0.03	0.08	0.03	0.02	0.04	0.01	0.22	0.3	0.1	0.06	0.1	0.07
g_7_25	11	25	113.09	0.0	0.01	0.01	0.07	0.1	0.01	0.04	0.06	0.02	0.22	0.26	0.03	0.09	0.13	0.02
g_8_25	12	25	32.22	0.0	0.01	0.01	0.13	0.15	0.01	0.08	0.11	0.01	0.3	0.34	0.04	0.15	0.18	0.02
g_9_25	13	25	206.35	0.0	0.01	0.01	0.14	0.23	0.05	0.08	0.14	0.04	0.35	0.46	0.14	0.18	0.23	0.03
g_10_25	14	25	236.77	0.0	0.02	0.01	0.22	0.26	0.03	0.11	0.17	0.03	0.38	0.42	0.02	0.24	0.28	0.03
g_11_25	15	25	232.87	0.0	0.02	0.01	0.22	0.32	0.12	0.13	0.2	0.04	0.36	0.41	0.04	0.28	0.33	0.03
g_12_25	16	25	368.31	0.02	0.04	0.01	0.38	0.46	0.06	0.29	0.51	0.19	0.54	0.61	0.08	0.44	0.52	0.07
g_13_25	17	25	592.9	0.03	0.06	0.02	0.52	0.66	0.13	0.48	0.86	0.25	0.65	0.76	0.12	0.6	0.75	0.18
g_14_25	18	25	7460.58	0.03	0.05	0.01	0.51	0.62	0.05	0.31	0.53	0.15	0.69	0.86	0.22	0.66	0.72	0.07
g_15_25	19	25	981.09	0.02	0.07	0.02	0.73	1.06	0.35	0.4	0.66	0.22	0.72	0.9	0.17	0.82	0.97	0.18
g_16_25	20	25	6884.92	0.05	0.14	0.04	0.97	1.66	0.45	0.99	1.57	0.49	1.01	1.14	0.15	1.19	1.58	0.3
g_17_25	21	25	12265.57	0.07	0.1	0.02	1.09	1.56	0.57	0.98	1.63	0.56	1.28	1.38	0.1	1.49	1.92	0.33
g_18_25	22	25	-	0.1	0.21	0.06	1.97	3.46	1.18	1.16	2.63	1.24	1.14	1.29	0.25	1.4	1.88	0.32
g_19_25	23	25	-	0.1	0.18	0.05	1.43	2.57	0.96	1.57	3.05	0.99	1.17	1.27	0.1	1.72	2.25	0.4
g_20_25	24	25	-	0.13	0.22	0.05	2.0	2.7	0.55	1.6	2.45	0.94	1.6	1.78	0.28	2.0	2.45	0.36
g_21_25	25	25	-	0.15	0.31	0.17	2.09	3.34	1.13	2.27	4.82	3.1	1.65	1.77	0.27	2.49	3.02	0.59
g_22_25	26	25	-	0.17	0.32	0.08	2.93	5.29	1.81	2.69	5.14	2.22	1.83	1.98	0.28	3.01	3.83	0.64
g_23_25	27	25	-	0.35	0.55	0.14	3.39	5.53	1.51	3.23	6.31	2.21	2.29	2.91	0.45	3.86	5.0	0.92
g_24_25	28	25	-	0.32	0.65	0.19	4.11	6.4	1.51	4.65	8.26	2.75	2.69	3.01	0.41	4.36	5.78	1.01
g_25_25	29	25	-	0.52	0.8	0.21	5.8	9.04	3.73	6.73	12.77	5.42	3.41	3.7	0.52	6.09	7.63	1.01
g_26_25	30	25	-	0.63	0.94	0.2	5.26	9.87	3.33	5.79	14.26	7.16	3.24	4.03	0.53	6.42	9.95	2.05
g_27_25	31	25	-	0.74	1.02	0.2	8.24	15.85	5.24	8.57	16.45	5.94	4.45	4.93	0.63	8.46	10.96	1.94
g_28_25	32	25	-	0.67	1.28	0.52	9.85	19.06	6.34	11.7	23.59	10.63	4.78	5.65	0.86	10.14	13.78	2.5
g_29_25	33	25	-	1.07	2.09	0.9	11.68	18.13	5.4	12.33	33.63	14.0	4.02	5.38	1.02	9.49	14.69	2.81
g_30_25	34	25	-	0.57	1.93	0.61	12.44	22.16	7.29	12.18	25.02	9.03	4.05	5.4	1.05	9.57	15.9	2.88
g_31_25	35	25	-	1.25	2.24	0.73	11.35	21.14	9.95	13.09	30.91	18.98	4.7	6.62	1.22	10.98	16.33	3.2
g_32_25	36	25	-	0.98	1.95	0.61	13.12	22.19	5.8	18.26	37.82	16.28	4.81	6.82	1.56	13.84	18.65	3.36
g_33_25	37	25	-	0.94	2.7	1.01	13.59	33.07	11.23	19.36	62.76	28.28	5.66	7.36	1.08	15.62	23.11	3.89
g_34_25	38	25	-	2.06	3.43	0.88	18.69	33.69	10.01	24.12	42.07	12.5	6.75	8.77	1.38	17.04	24.28	4.12
g_35_25	39	25	-	3.21	4.62	1.27	19.46	38.1	14.02	39.28	71.94	20.73	6.16	8.54	1.55	19.73	31.2	6.28
g_36_25	40	25	-	1.8	3.69	1.24	22.54	45.01	17.92	26.35	65.91	28.0	6.89	9.5	1.71	22.68	33.57	5.62
g_37_25	41	25	-	2.74	6.0	2.09	23.67	40.66	9.25	41.37	82.61	28.71	7.62	10.52	1.9	24.5	36.7	6.03

Apêndice D – Resultados do tamanho do corte das instâncias com 50% de densidade do Experimento II

Problema			Valores dos cortes														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_50	5	50	3	3	3	0.0	3	3	0.0	3	3	0.0	3	3	0.0	3	3	0.0
g_2_50	6	50	6	6	6	0.0	6	6	0.0	6	6	0.0	6	6	0.0	6	6	0.0
g_3_50	7	50	5	5	4	0.42	5	5	0.0	5	5	0.0	5	5	0.0	5	5	0.0
g_4_50	8	50	11	11	10	0.34	11	11	0.0	11	11	0.0	11	11	0.0	11	11	0.0
g_5_50	9	50	12	12	12	0.0	12	12	0.0	12	12	0.0	12	12	0.0	12	12	0.0
g_6_50	10	50	18	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0	18	18	0.0
g_7_50	11	50	22	22	21	0.34	22	22	0.0	22	22	0.0	22	21	0.44	22	22	0.0
g_8_50	12	50	22	22	21	0.85	22	22	0.0	22	22	0.0	22	21	0.47	22	21	0.25
g_9_50	13	50	26	26	26	0.0	26	26	0.0	26	26	0.0	26	25	0.37	26	26	0.0
g_10_50	14	50	35	35	34	0.49	35	35	0.0	35	34	0.47	35	34	0.49	35	34	0.37
g_11_50	15	50	31	31	30	0.71	31	30	0.18	31	30	0.5	31	29	0.62	31	30	0.49
g_12_50	16	50	40	40	40	0.0	40	40	0.0	40	40	0.0	40	39	0.67	40	40	0.0
g_13_50	17	50	48	48	46	0.98	48	47	0.18	48	47	0.98	48	46	1.01	48	47	0.88
g_14_50	18	50	48	48	47	0.81	48	48	0.0	48	47	0.25	48	47	0.7	48	47	0.34
g_15_50	19	50	58	58	55	2.24	58	58	0.0	58	57	1.13	58	56	0.93	58	57	1.2
g_16_50	20	50	-	60	58	1.37	60	59	0.42	60	59	0.74	60	58	0.98	60	59	0.56
g_17_50	21	50	-	71	68	1.19	71	70	0.5	71	70	0.68	71	69	0.86	71	70	0.5
g_18_50	22	50	-	83	81	1.65	83	83	0.0	83	83	0.0	83	80	1.36	83	83	0.0
g_19_50	23	50	-	87	85	1.92	87	86	0.8	87	86	0.8	87	84	1.17	87	86	0.34
g_20_50	24	50	-	93	92	1.14	93	92	0.84	93	92	0.92	92	88	1.5	93	92	1.04
g_21_50	25	50	-	102	99	2.05	102	101	0.64	102	100	0.79	102	99	1.39	102	101	0.88
g_22_50	26	50	-	108	106	0.48	108	107	0.53	108	107	0.48	108	105	1.69	108	107	0.25
g_23_50	27	50	-	110	108	1.02	110	109	0.8	110	109	0.76	108	106	1.24	110	109	0.56
g_24_50	28	50	-	123	119	1.67	123	121	1.53	123	121	1.75	123	116	1.97	123	121	1.91
g_25_50	29	50	-	131	127	2.67	131	130	1.2	131	130	1.57	128	123	2.05	131	130	0.88
g_26_50	30	50	-	141	137	1.84	141	140	1.02	141	139	1.77	138	132	1.96	141	140	1.42
g_27_50	31	50	-	140	136	2.99	140	139	0.69	140	137	2.69	137	132	2.01	140	138	2.34
g_28_50	32	50	-	169	165	2.69	169	168	1.31	169	168	1.31	166	159	2.07	169	168	1.26
g_29_50	33	50	-	163	160	2.11	163	162	0.76	163	162	0.85	158	154	1.73	163	162	0.47
g_30_50	34	50	-	187	185	1.12	187	186	1.07	187	186	1.03	185	178	2.36	187	185	0.84
g_31_50	35	50	-	177	172	3.54	177	176	0.81	177	176	1.5	173	166	2.31	177	176	0.55
g_32_50	36	50	-	190	185	3.12	190	187	2.94	190	187	3.11	188	178	3.58	190	186	3.25
g_33_50	37	50	-	216	213	3.11	216	214	1.98	216	212	3.02	211	202	3.2	216	212	2.86
g_34_50	38	50	-	215	211	3.2	215	213	1.94	215	211	2.31	208	200	2.73	215	212	2.12
g_35_50	39	50	-	231	228	1.56	232	230	1.38	232	228	1.68	224	218	2.44	232	228	1.48
g_36_50	40	50	-	241	234	5.13	244	241	3.03	244	241	3.54	234	224	3.15	244	239	5.06
g_37_50	41	50	-	256	250	3.78	256	254	1.65	256	253	2.77	247	239	3.57	256	253	3.26

Apêndice E – Resultados do tempo computacional das instâncias com 50% de densidade do Experimento II

Problema			Tempo computacional														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_50	5	50	0.75	0.0	0.0	0.0	0.0	0.01	0.01	0.0	0.01	0.0	0.12	0.17	0.09	0.0	0.02	0.01
g_2_50	6	50	0.71	0.0	0.0	0.0	0.0	0.02	0.01	0.0	0.01	0.01	0.14	0.17	0.02	0.0	0.03	0.01
g_3_50	7	50	0.77	0.0	0.0	0.0	0.01	0.03	0.01	0.0	0.01	0.01	0.13	0.16	0.01	0.03	0.04	0.01
g_4_50	8	50	14.71	0.0	0.0	0.01	0.03	0.06	0.01	0.01	0.03	0.01	0.26	0.31	0.02	0.05	0.07	0.01
g_5_50	9	50	26.78	0.0	0.01	0.01	0.05	0.08	0.01	0.02	0.04	0.01	0.28	0.33	0.03	0.07	0.09	0.01
g_6_50	10	50	34.39	0.0	0.01	0.01	0.1	0.13	0.02	0.04	0.08	0.02	0.39	0.53	0.15	0.12	0.14	0.01
g_7_50	11	50	271.14	0.01	0.02	0.01	0.18	0.22	0.03	0.08	0.15	0.04	0.54	0.58	0.03	0.18	0.22	0.02
g_8_50	12	50	125.53	0.0	0.02	0.01	0.22	0.35	0.11	0.11	0.18	0.07	0.57	0.62	0.04	0.22	0.29	0.04
g_9_50	13	50	593.57	0.01	0.03	0.01	0.28	0.33	0.02	0.15	0.22	0.03	0.66	0.76	0.14	0.33	0.36	0.02
g_10_50	14	50	3934.63	0.02	0.04	0.01	0.44	0.69	0.19	0.23	0.53	0.22	1.07	1.16	0.04	0.47	0.6	0.09
g_11_50	15	50	3937.67	0.03	0.07	0.04	0.54	0.78	0.21	0.27	0.44	0.1	0.9	1.02	0.14	0.57	0.68	0.09
g_12_50	16	50	1543.95	0.03	0.06	0.02	0.68	0.84	0.09	0.47	0.68	0.3	1.2	1.28	0.05	0.7	0.88	0.15
g_13_50	17	50	13813.52	0.08	0.12	0.03	1.12	1.77	0.5	0.54	1.16	0.48	1.54	1.72	0.17	1.06	1.27	0.18
g_14_50	18	50	37450.11	0.08	0.12	0.03	1.23	1.74	0.46	0.96	1.69	0.76	1.79	2.11	0.31	1.3	1.47	0.19
g_15_50	19	50	76725.92	0.12	0.23	0.08	1.98	2.92	0.83	1.55	2.62	0.72	2.04	2.5	0.4	1.6	2.07	0.34
g_16_50	20	50	-	0.16	0.34	0.09	2.08	3.46	1.06	1.24	2.32	0.96	1.56	1.85	0.31	1.35	1.55	0.15
g_17_50	21	50	-	0.12	0.2	0.05	1.87	3.4	1.38	1.71	4.1	2.68	2.91	3.84	0.97	1.89	2.61	0.53
g_18_50	22	50	-	0.28	0.49	0.18	2.3	3.48	0.77	2.75	5.17	1.89	2.99	3.59	0.95	2.93	3.75	0.71
g_19_50	23	50	-	0.22	0.48	0.12	2.85	4.53	1.5	2.06	4.86	1.77	2.59	3.03	0.4	2.74	3.25	0.41
g_20_50	24	50	-	0.29	0.47	0.11	3.08	4.36	0.99	2.89	4.78	1.04	2.71	2.9	0.25	2.99	3.41	0.46
g_21_50	25	50	-	0.27	0.55	0.17	3.98	6.35	1.91	3.69	7.12	1.86	3.24	3.47	0.33	4.13	5.21	1.1
g_22_50	26	50	-	0.43	0.63	0.12	5.7	7.61	1.57	5.55	7.73	1.9	4.91	5.41	0.43	5.85	6.79	0.77
g_23_50	27	50	-	0.55	0.77	0.19	5.97	10.25	3.12	6.6	12.03	3.78	5.28	5.75	0.52	6.85	8.35	1.43
g_24_50	28	50	-	0.77	1.26	0.29	10.04	15.12	3.88	8.4	20.98	9.86	6.08	6.84	0.44	8.38	10.33	1.61
g_25_50	29	50	-	0.82	1.53	0.56	12.08	18.32	4.7	6.02	22.15	10.38	6.9	7.43	0.54	9.98	12.37	1.59
g_26_50	30	50	-	1.1	1.68	0.38	10.44	21.2	7.74	11.84	23.53	12.7	5.42	7.38	1.48	8.71	14.15	3.46
g_27_50	31	50	-	0.97	1.64	0.47	9.74	23.34	8.1	11.32	24.55	9.88	5.61	7.57	1.56	9.6	14.86	3.88
g_28_50	32	50	-	1.17	2.43	0.81	14.08	24.95	7.73	14.17	34.07	16.65	7.24	9.79	1.99	11.87	19.03	4.62
g_29_50	33	50	-	1.52	2.72	0.94	12.66	29.06	10.38	14.94	36.0	11.97	7.05	9.66	1.81	13.55	20.44	5.03
g_30_50	34	50	-	2.27	3.83	1.2	17.69	33.71	13.98	17.58	45.09	24.98	8.43	11.58	2.34	14.8	26.79	6.93
g_31_50	35	50	-	1.55	3.94	1.94	17.58	38.17	16.17	32.48	70.94	36.57	7.95	14.67	9.77	17.03	27.6	5.48
g_32_50	36	50	-	2.4	4.41	1.54	22.4	46.16	15.55	29.06	66.59	30.21	8.83	12.46	2.42	22.52	32.46	6.6
g_33_50	37	50	-	2.72	6.89	2.28	27.09	54.65	25.09	31.83	90.23	51.12	10.62	14.99	2.99	22.18	36.18	7.31
g_34_50	38	50	-	3.0	6.95	2.4	27.6	57.64	21.78	33.55	85.0	43.3	10.68	15.34	3.13	23.86	38.13	6.87
g_35_50	39	50	-	3.05	6.52	2.02	30.84	66.3	25.23	40.46	88.77	47.44	12.21	17.55	3.75	27.68	43.97	9.93
g_36_50	40	50	-	3.17	9.46	2.63	43.13	85.8	28.43	56.42	150.73	65.35	12.97	18.18	3.81	36.41	57.26	14.32
g_37_50	41	50	-	4.79	8.47	2.27	37.77	95.28	52.98	78.19	226.32	109.66	15.79	22.25	3.37	45.99	67.94	14.8

Apêndice F – Resultados do tamanho do corte das instâncias com 75% de densidade do Experimento II

Problema			Valores dos cortes														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_75	5	75	6	6	6	0.0	6	6	0.0	6	6	0.0	6	6	0.0	6	6	0.0
g_2_75	6	75	9	9	9	0.0	9	9	0.0	9	9	0.0	9	9	0.0	9	9	0.0
g_3_75	7	75	11	11	11	0.0	11	11	0.0	11	11	0.0	11	11	0.0	11	11	0.0
g_4_75	8	75	15	15	14	0.18	15	15	0.0	15	15	0.0	15	15	0.0	15	15	0.0
g_5_75	9	75	18	18	17	0.3	18	18	0.0	18	18	0.0	18	18	0.0	18	17	0.18
g_6_75	10	75	23	23	22	0.5	23	23	0.0	23	23	0.0	23	22	0.18	23	23	0.0
g_7_75	11	75	25	25	24	0.55	25	25	0.0	25	24	0.37	25	24	0.4	25	24	0.18
g_8_75	12	75	34	34	33	0.5	34	34	0.0	34	33	0.37	34	33	0.37	34	34	0.0
g_9_75	13	75	34	34	33	0.3	34	34	0.0	34	34	0.0	34	34	0.0	34	34	0.0
g_10_75	14	75	42	42	41	0.25	42	42	0.0	42	42	0.0	42	42	0.0	42	42	0.0
g_11_75	15	75	52	52	51	0.71	52	52	0.0	52	52	0.0	52	51	0.76	52	51	0.18
g_12_75	16	75	57	57	56	0.63	57	57	0.0	57	57	0.0	57	56	0.48	57	57	0.0
g_13_75	17	75	62	62	61	0.76	62	62	0.0	62	62	0.0	62	60	0.83	62	61	0.18
g_14_75	18	75	-	71	70	0.34	71	70	0.18	71	70	0.25	71	70	0.52	71	70	0.25
g_15_75	19	75	-	82	80	1.61	82	82	0.0	82	81	1.28	82	80	0.95	82	82	0.0
g_16_75	20	75	-	88	87	0.96	88	87	0.18	88	87	0.18	88	87	0.86	88	87	0.18
g_17_75	21	75	-	93	92	1.12	93	92	0.25	93	92	0.4	93	91	1.11	93	92	0.4
g_18_75	22	75	-	108	107	1.41	108	107	0.3	108	107	0.48	108	106	1.31	108	107	0.37
g_19_75	23	75	-	113	112	0.98	113	112	0.18	113	112	0.89	113	110	1.2	113	112	0.99
g_20_75	24	75	-	130	126	1.11	130	129	0.98	128	127	0.48	130	127	1.02	130	128	0.98
g_21_75	25	75	-	133	132	1.27	134	133	0.48	134	132	1.06	133	130	1.32	134	133	0.46
g_22_75	26	75	-	145	144	1.03	145	145	0.0	145	144	0.18	144	140	1.64	145	145	0.0
g_23_75	27	75	-	159	158	0.55	159	158	0.25	159	158	0.18	158	154	1.8	159	159	0.0
g_24_75	28	75	-	165	162	1.74	165	164	0.56	165	164	1.33	162	159	1.36	165	164	1.38
g_25_75	29	75	-	172	170	2.12	172	171	0.9	172	171	1.17	171	167	2.08	172	171	1.11
g_26_75	30	75	-	196	192	2.45	196	195	1.45	196	194	1.85	196	189	2.13	196	195	1.22
g_27_75	31	75	-	205	203	2.36	205	204	1.09	205	203	1.66	205	198	1.82	205	203	1.7
g_28_75	32	75	-	221	218	1.69	221	220	1.2	221	220	1.44	218	214	1.94	221	221	0.0
g_29_75	33	75	-	227	223	1.81	227	226	1.0	227	225	1.09	222	219	1.81	227	225	0.93
g_30_75	34	75	-	238	234	2.86	238	236	2.17	238	236	2.24	233	228	2.09	238	237	1.77
g_31_75	35	75	-	251	249	2.09	251	250	0.52	251	250	1.44	246	242	1.71	251	250	1.35
g_32_75	36	75	-	267	263	2.3	267	266	1.44	267	264	2.18	262	257	2.25	267	265	1.58
g_33_75	37	75	-	281	275	2.12	281	278	1.99	281	278	2.09	277	269	2.73	281	279	1.89
g_34_75	38	75	-	302	299	1.7	302	301	1.06	302	300	1.28	297	290	2.47	302	301	1.22
g_35_75	39	75	-	313	310	2.78	313	312	1.09	313	312	1.0	306	301	2.89	313	312	0.64
g_36_75	40	75	-	338	335	3.61	338	337	0.8	338	337	1.8	332	322	3.05	338	337	0.37
g_37_75	41	75	-	350	346	2.98	351	349	1.69	351	348	1.96	341	336	2.03	351	349	2.17

Apêndice G – Resultados do tempo computacional das instâncias com 75% de densidade do Experimento II

Problema			Tempo computacional														
			ABB	GRASP-VND			GRASP-VNS			BT			AG			SA		
Nome	V	Dens		Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp	Melhor	Média	Dp
g_1_75	5	75	0.99	0.0	0.0	0.0	0.0	0.01	0.01	0.0	0.0	0.0	0.13	0.16	0.02	0.01	0.02	0.0
g_2_75	6	75	4.87	0.0	0.0	0.0	0.01	0.02	0.01	0.0	0.01	0.01	0.2	0.23	0.02	0.02	0.03	0.01
g_3_75	7	75	29.84	0.0	0.0	0.01	0.01	0.04	0.01	0.0	0.02	0.01	0.24	0.31	0.03	0.03	0.05	0.01
g_4_75	8	75	26.63	0.0	0.0	0.01	0.03	0.07	0.02	0.02	0.03	0.01	0.37	0.48	0.13	0.05	0.08	0.02
g_5_75	9	75	30.19	0.0	0.01	0.01	0.03	0.11	0.03	0.03	0.05	0.01	0.43	0.47	0.03	0.08	0.12	0.02
g_6_75	10	75	120.9	0.0	0.02	0.01	0.16	0.19	0.02	0.06	0.1	0.03	0.59	0.71	0.15	0.15	0.19	0.02
g_7_75	11	75	514.05	0.01	0.02	0.01	0.23	0.3	0.06	0.09	0.18	0.08	0.69	0.76	0.03	0.21	0.29	0.06
g_8_75	12	75	337.25	0.02	0.05	0.03	0.32	0.46	0.09	0.17	0.27	0.06	0.97	1.03	0.04	0.3	0.43	0.12
g_9_75	13	75	688.75	0.02	0.03	0.01	0.39	0.49	0.06	0.18	0.3	0.08	0.99	1.12	0.19	0.37	0.42	0.02
g_10_75	14	75	4467.39	0.03	0.06	0.02	0.56	0.65	0.05	0.27	0.45	0.09	1.36	1.5	0.19	0.54	0.64	0.06
g_11_75	15	75	21831.22	0.06	0.15	0.05	0.82	1.1	0.21	0.49	0.75	0.15	1.65	1.82	0.24	0.78	0.94	0.19
g_12_75	16	75	6573.29	0.05	0.1	0.03	0.99	1.37	0.26	0.54	0.99	0.38	1.92	2.08	0.24	0.97	1.14	0.19
g_13_75	17	75	37922.93	0.07	0.14	0.03	1.36	1.8	0.46	0.67	1.3	0.38	2.12	2.32	0.28	1.26	1.53	0.21
g_14_75	18	75	-	0.09	0.2	0.05	1.89	2.4	0.44	1.2	1.8	0.46	2.82	3.07	0.3	1.57	2.01	0.5
g_15_75	19	75	-	0.16	0.28	0.08	2.24	3.05	0.76	1.63	3.08	0.85	3.05	3.33	0.37	2.15	2.52	0.26
g_16_75	20	75	-	0.21	0.35	0.09	2.85	4.12	1.0	1.84	2.95	1.32	3.53	3.86	0.41	2.39	2.93	0.41
g_17_75	21	75	-	0.21	0.4	0.08	3.23	4.56	1.32	2.4	3.34	0.64	3.73	4.04	0.35	2.95	3.33	0.39
g_18_75	22	75	-	0.37	0.63	0.17	4.34	6.2	1.5	3.88	5.84	1.79	4.64	5.03	0.55	3.94	4.73	0.6
g_19_75	23	75	-	0.5	0.73	0.22	4.83	7.87	2.65	4.51	6.99	2.08	4.93	5.47	0.43	4.82	5.67	0.76
g_20_75	24	75	-	0.41	0.65	0.23	7.25	14.23	5.34	6.24	14.43	5.22	6.65	9.82	2.58	4.33	9.16	3.69
g_21_75	25	75	-	0.33	0.61	0.14	5.39	8.42	2.92	3.42	7.27	2.37	4.47	5.76	1.54	5.82	6.89	0.79
g_22_75	26	75	-	0.8	1.62	0.47	7.07	10.72	3.94	5.33	9.06	2.15	5.18	6.62	0.94	6.94	8.02	1.28
g_23_75	27	75	-	0.93	1.6	0.5	9.73	12.37	2.49	8.55	17.23	7.02	7.02	7.87	0.91	9.3	11.57	1.72
g_24_75	28	75	-	0.93	1.58	0.45	9.94	18.85	6.75	10.87	22.54	10.28	8.56	11.02	2.25	9.66	12.81	1.6
g_25_75	29	75	-	1.24	2.23	0.77	14.53	20.86	6.99	8.85	23.7	7.76	6.78	7.16	0.42	8.86	11.77	2.55
g_26_75	30	75	-	1.21	2.83	1.01	14.74	30.72	10.08	15.7	49.9	25.88	9.43	11.86	2.21	12.43	19.84	4.81
g_27_75	31	75	-	1.78	3.45	1.28	19.01	33.42	12.85	15.56	39.19	13.51	10.14	13.45	2.79	15.48	22.5	4.78
g_28_75	32	75	-	1.91	3.55	1.4	20.3	49.42	22.89	17.44	64.82	39.31	11.6	24.65	9.23	16.7	41.15	16.77
g_29_75	33	75	-	2.26	7.1	3.09	24.29	88.4	41.65	29.14	74.22	36.63	12.15	16.12	3.98	18.86	28.6	5.76
g_30_75	34	75	-	2.86	5.21	1.68	28.75	58.12	24.34	34.0	81.89	47.84	13.67	15.98	1.96	24.3	33.35	5.54
g_31_75	35	75	-	2.6	5.04	1.63	32.86	53.03	23.63	38.02	61.55	25.15	15.0	18.49	2.1	26.93	36.9	6.1
g_32_75	36	75	-	4.94	7.71	2.26	42.59	71.44	20.96	44.09	95.46	28.13	16.29	19.75	1.86	32.23	47.0	8.68
g_33_75	37	75	-	4.38	8.33	2.8	49.85	85.59	25.13	66.69	114.07	41.64	19.74	23.64	4.0	40.52	53.15	7.71
g_34_75	38	75	-	4.99	8.63	2.21	53.47	85.53	26.61	61.45	123.71	51.11	22.46	25.27	2.55	46.66	61.47	10.45
g_35_75	39	75	-	5.73	8.29	1.22	50.64	82.96	27.07	64.71	136.87	74.58	19.91	20.19	0.18	43.54	53.92	7.92
g_36_75	40	75	-	6.09	10.92	3.19	58.52	101.51	33.37	84.12	223.32	144.73	19.55	25.61	7.78	50.93	85.97	29.52
g_37_75	41	75	-	7.79	14.16	3.6	65.86	156.56	58.88	106.33	226.22	97.95	24.31	28.97	3.9	57.36	84.9	14.33