UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOSÉ ROBERTTY DE FREITAS COSTA

ABORDAGENS HEURÍSTICAS E EXATAS PARA O PROBLEMA DA MÁXIMA INTERSEÇÃO DE K-SUBCONJUNTOS

QUIXADÁ

2021

JOSÉ ROBERTTY DE FREITAS COSTA

ABORDAGENS HEURÍSTICAS E EXATAS PARA O PROBLEMA DA MÁXIMA INTERSEÇÃO DE K-SUBCONJUNTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Fábio Carlos Sousa Dias

Coorientador: Prof. Dr. Wladimir Araújo Tavares

QUIXADÁ

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C873a Costa, José Robertty de Freitas.
Abordagens Heurísticas e Exatas para o Problema da Máxima Interseção de k-Subconjuntos / José Robertty de Freitas Costa. – 2021.
64 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Computação, Quixadá, 2021.
Orientação: Prof. Dr. Fábio Carlos Sousa Dias.
Coorientação: Prof. Dr. Wladimir Araújo Tavares.

1. Otimização Combinatória. 2. Heurística. 3. Branch and Bound. 4. Programação Linear Inteira. I. Título.
CDD 621.39

JOSÉ ROBERTTY DE FREITAS COSTA

ABORDAGENS HEURÍSTICAS E EXATAS PARA O PROBLEMA DA MÁXIMA INTERSEÇÃO DE K-SUBCONJUNTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: ___/___/_____

BANCA EXAMINADORA

Prof. Dr. Fábio Carlos Sousa Dias (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araújo Tavares (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Atílio Gomes Luiz
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Henrique Macêdo de Araújo
Universidade Federal do Ceará (UFC)

A Deus.

Aos meus pais, Francisco e Neide.

A minha esposa, Thyara.

AGRADECIMENTOS

Agradeço a Deus, por me dar forças e saúde durante toda esta caminhada.

Agradeço aos meus pais, Francisco e Neide, por todo sacrifício feito por eles para que eu pudesse ter acesso a uma educação de qualidade.

Agradeço a minha esposa, Thyara Araújo, por estar sempre ao meu lado me ajudando e me dando forças em todos momentos.

Agradeço ao Prof. Dr. Fábio Carlos Souza Dias, pela sua dedicação e paciência ao me orientar neste trabalho.

Agradeço ao Prof. Dr. Wladimir Araújo Tavares, pelas suas ótimas colaborações dadas a este trabalho.

Agradeço aos professores Atílio Gomes Luiz e Paulo Henrique Macêdo de Araújo, pela disponibilidade em participar da banca desse trabalho, e pelas suas colaborações e sugestões.

A todos os professores da Universidade Federal do Ceará - Campus Quixadá e, de maneira especial, aos professores Arthur Araruna, Atílio Gomes, Fábio Dias, Francicleber Ferreira, João Marcelo, Joel Castro, Paulo de Tarso, Paulo Henrique, Thiago Werlley e Wladimir Tavares, que, com seus exemplos de dedicação e empenho, foram muito importantes na minha formação acadêmica.

Agradeço aos meus colegas de graduação, pelas ajudas, críticas e reflexões recebidas. Em especial, gostaria de agradecer aos meus amigos Antônio Andson, Carlos Alberto, Claro Henrique, Gabriel Uchôa, João Mateus, Johnny Marcos, Paulo Miranda e Robert Almeida.

"An equation means nothing to me unless it expresses a thought of God."

(Srinivasa Ramanujan)

RESUMO

O Problema da Máxima Interseção de k-Subconjuntos pode ser definido como: dados dois conjuntos L e R, onde $L = \{S_1, S_2, \cdots, S_p\}$ é uma coleção de p subconjuntos de $R = \{e_1, e_2, \cdots, e_q\}$, e um número inteiro positivo k, temos que encontrar k subconjuntos pertencentes a L, tal que a interseção deles seja máxima. Este problema já foi demonstrado pertencer à classe $\mathscr{NP}$-difícil e possui aplicações na área de privacidade de dados e redes sociais. O objetivo deste trabalho foi propor novas abordagens, exatas e heurísticas, para o Problema da Máxima Interseção de k-Subconjuntos. Neste trabalho, foram propostas duas heurísticas gulosas, dois algoritmos Meta-heuríticos, um algoritmo *Branch and Bound* e uma formulação de Programação Linear Inteira. Além disso, foi proposto um novo procedimento de Teste de Redução, que visa diminuir a dimensão do problema. Foram geradas instâncias aleatórias e realizados diversos testes computacionais. Os resultados computacionais evidenciaram que os algoritmos Meta-heurísticos propostos superam o algoritmo heurístico de estado da arte em qualidade de solução ou em tempo de execução. Em relação às abordagens exatas, o algoritmo *Branch and Bound* proposto se mostrou mais eficiente que o método do estado da arte em instâncias onde o grafo possui uma densidade baixa ou média.

Palavras-chave: Otimização Combinatória. Heurística. Branch and Bound. Programação Linear Inteira.

ABSTRACT

The Maximum k-Subset Intersection Problem can be defined as follows: given two sets L and R, where $L = \{S_1, S_2, \cdots, S_p\}$ is a collection of p subsets of $R = \{e_1, e_2, \cdots, e_q\}$, and a positive integer k, we have to find k subsets belonging to L such that their intersection is maximum. This problem has already been shown to belong to the class $\mathscr{NP}$-hard and has applications in the area of data privacy and social networks. The objective of this work was to propose new approaches, exact and heuristic, for the Maximum k-Subset Intersection Problem. In this work, we propose two greedy heuristics, two Metaheuristics algorithms, a *Branch and Bound* algorithm and a formulation of Integer Linear Programming. In addition, a new Reduction Test procedure has been proposed, which aims to reduce the scale of the problem. Random instances were generated and several computational tests were performed. The computational results showed that the proposed Metaheuristic algorithms surpass the state-of-the-art heuristic algorithm in solution quality or in execution time. Regarding the exact approaches, the proposed *Branch and Bound* algorithm proved to be more efficient than the state-of-the-art method for instances where the graph has a low or medium density.

Keywords: Combinatorial Optimization. Heuristic. Branch and Bound. Integer Linear Programming.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A área de otimização combinatória busca modelar e resolver problemas com base em suas restrições e em seus objetivos. Alguns objetivos típicos dos problemas de otimização combinatória consistem em aproveitar melhor os materiais utilizados no processo de produção, otimizar o tempo para realização de ações e operações, transportar mais materiais pelas melhores rotas, diminuir chances de se obter prejuízos, aumentar lucros e diminuir gastos, etc (MIYAZAWA F. K; DE SOUZA, 2015).

Existem diversos problemas, descritos na literatura, com aplicações práticas nas áreas de privacidade dos dados (MANGASARIAN, 2011), alocação de recursos (CRUZ C. D. A.; NETO, 2019), bioinformática (ADI; FERREIRA, 2005) e entre outras, que são modelados e resolvidos por técnicas de otimização combinatória.

Uma aplicação recorrente da área de privacidade de dados visa determinar dados seguros para serem divulgados. Um conjunto de dados é considerado seguro para ser publicado quando não se consegue facilmente determinar de qual pessoa ou de qual instituição aqueles dados provêm. A segurança de um conjunto de dados pode ser obtida encontrando um conjunto de pessoas/instituições que compartilham esses dados. Este processo é conhecido como anonimização de dados. O problema da Máxima Interseção de k-Subconjuntos está relacionado com o problema de maximizar o número de informações em comum (interseção) a pelo menos k pessoas (cada pessoa pode ser modelada por um subconjuntos de informações). Note que resolvendo este problema garantimos um certo nível de anonimização dos dados.

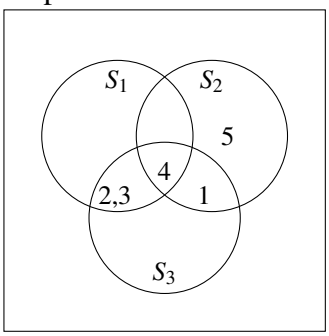
O problema da Máxima Interseção de k-Subconjuntos foi inicialmente proposto por Vinterbo (2002), onde o mesmo citou a aplicação deste problema na resolução do problema de privacidade dos dados de pacientes de um hospital. Uma aplicação para o problema na área de Redes Sociais, consiste na tarefa de encontrar um grupo de pessoas com o maior número de interesses em comum. Seja R um conjunto de interesses e L o conjunto de pessoas representadas por subconjuntos de R, o objetivo é encontrar um grupo de k pessoas (subconjuntos de interesses) em L com maior quantidade de interesses em comum (interseção máxima) (BOGUE *et al.*, 2013).

Formalmente, o Problema da Máxima Interseção de k-Subconjuntos, também conhecido como kMIS, pode ser definido como: dados dois conjuntos L e R, onde $L = \{S_1, S_2, \cdots, S_p\}$ é uma coleção de p subconjuntos de $R = \{e_1, e_2, \cdots, e_q\}$, e um número inteiro positivo k, temos que encontrar k subconjuntos pertencentes a L tal que à interseção deles seja máxima. Apesar de fácil definição, o problema já foi demonstrado ser $\mathscr{NP}$-Difícil (VINTERBO, 2002) através de

uma redução ao problema da Máxima Biclique Balanceada (GAREY; JOHNSON, 1979).

De forma ilustrativa, é apresentada uma instância do problema utilizando um diagrama de Venn na Figura 1. Suponha $R = \{1,2,3,4,5\}$, $L = \{S_1, S_2, S_3\}$, onde $S_1 = \{2,3,4\}$, $S_2 = \{1,4,5\}$ e $S_3 = \{1,2,3,4\}$. Para $k = 1$, basta encontrar o subconjunto pertencente a L, com a maior cardinalidade. Neste exemplo, a solução seria o subconjunto S_3. Para $k = 2$, devemos encontrar a maior interseção de subconjuntos pertencentes a L, tomados aos pares. Neste caso, temos que $|S_1 \cap S_2| = 1, |S_1 \cap S_3| = 3$ e $|S_2 \cap S_3| = 2$. Logo, a maior interseção para $k = 2$ é obtida escolhendo os subconjuntos S_1 e S_3.

Figura 1 – Exemplo de instância do problema kMIS.



Fonte: Próprio autor.

Um grafo é uma estrutura matemática abstrata que modela relações entre objetos. Um grafo simples G, consiste em um par ordenado $(V(G), E(G))$, onde $V(G)$ é um conjunto finito, não vazio, denominado de conjunto de vértices, e $E(G)$ é um conjunto finito chamado de conjunto de arestas. Uma aresta $e = uv \in E(G)$ é formada por dois vértices $u, v \in V(G)$, onde $u \neq v$. Dizemos que dois vértices u, v são adjacentes se $uv \in E(G)$. Neste caso, se u e v são adjacentes, dizemos também que estes vértices são vizinhos. Definimos grau de um vértice u, denotado por $d_G(u)$, como o número de vértices adjacentes a u. Usualmente, definimos $N_G(u)$, como o conjunto dos vértices vizinhos do vértice u.

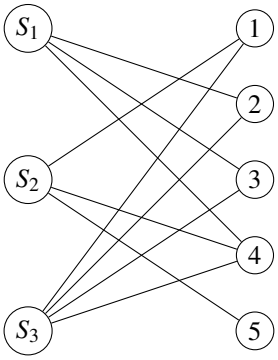
Um grafo G é dito bipartido, quando seu conjunto de vértices $V(G)$ pode ser particionado em dois conjuntos disjuntos A e B, tal que, para toda aresta $uv \in E(G)$, temos que $u \in A$ e $v \in B$, ou $u \in B$ e $v \in A$. Denotamos o grafo bipartido G, por $G = (A \cup B, E)$. Observe que uma instância do Problema da Máxima Interseção de k-Subconjuntos, representada pelos conjuntos L e R, pode ser modelada como um grafo bipartido $G = (A \cup B, E)$, onde A contém vértices i associados aos elementos S_i de L, e B contém os vértices j associados aos elementos e_j de R. Uma aresta ij, pertence ao conjunto $E(G)$ se, e somente se, $e_j \in S_i$. Por abuso de notação, iremos considerar que o grafo que modela uma instância do problema é um grafo bipartido

$G = (L \cup R, E)$.

Em um grafo bipartido, dado o vértice $u \in L$, definimos $\overline{N_G(u)}$ como o conjunto de vértices em R que não são vizinhos do vértice u, ou seja, $\overline{N_G(u)} = R \setminus N_G(u)$. De forma análoga, se $u \in R$ então definimos $\overline{N_G(u)} = L \setminus N_G(u)$. Por fim, definimos a interseção de vizinhanças de uma coleção de vértices C em um grafo G, como $\cap(C) = \bigcap_{u \in C} N_G(u)$.

Utilizando as terminologias de teoria dos grafos, podemos definir o Problema da Máxima Interseção de k-Subconjuntos como: dado um grafo bipartido $G = (L \cup R, E)$ e um número inteiro positivo k, temos como objetivo encontrar o conjunto $L' \subseteq L$, com $|L'| = k$, tal que, $|\cap(L')|$ seja máxima. A Figura 2, apresenta a instância exemplo representada por meio de um grafo bipartido.

Figura 2 – Instância exemplo representada como um grafo bipartido.



Fonte: Próprio autor.

Alguns trabalhos na literatura buscam propor abordagens exatas e heurísticas para o Problema da Máxima Interseção de k-Subconjuntos. Bogue *et al.* (2013) apresentam uma heurística gulosa, chamada kInter, e três formulações de Programação Linear Inteira (PLI). Duas dessas formulações foram obtidas a partir de adaptações de formulações para o problema da Biclique Máxima em Arestas (MEB do inglês *Maximum Edge Biclique Problem*) e uma nova formulação desenvolvida diretamente para o problema, denominada de modelo M_ARESTA.

Em Bogue *et al.* (2014a), é apresentada uma implementação da Meta-heurística GRASP para o problema, com o objetivo de encontrar uma solução inicial de qualidade. Além disso, Bogue *et al.* (2014a), apresentam uma nova formulação de Programação Linear Inteira, uma análise poliédrica do problema e procedimentos que visam a redução da dimensão do problema.

Apesar de existir diversas abordagens exatas para o problema, ainda não existe abordagem exata que resolva instâncias de tamanho grande em tempo não proibitivo. O que

faz com que seja necessário a investigação de novas abordagens exatas para o problema. Já as abordagens heurísticas existentes, se mostram promissoras devido à qualidade das soluções geradas pelas mesmas. Além disso, os algoritmos heurísticos mais sofisticados podem melhorar ainda mais a qualidade das soluções, e por conseguinte, gerar métodos exatos que consigam resolver instâncias cada vez maiores.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é propor abordagens exatas e heurísticas para o Problema da Máxima Interseção de k-Subconjuntos que visem resolver o problema de forma mais eficiente, seja em tempo ou em qualidade de solução, do que as abordagens algorítmicas existentes na literatura.

1.1.2 Objetivos Específicos

Os principais objetivos específicos deste trabalho são:

- Propor novas abordagens heurísticas e Meta-heurísticas;
- Propor novas técnicas de pré-processamento das instâncias;
- Propor uma nova formulação de Programação Linear Inteira (PLI);
- Apresentar um algoritmo *Branch and Bound* Combinatorial para o kMIS;
- Comparar as abordagens, exatas e heurísticas, propostas neste trabalho com as já existentes na literatura, a fim de determinar quais abordagens são mais eficientes para resolver o kMIS.

Neste capítulo, foi apresentado o Problema da Máxima Interseção de k-Subconjuntos e as suas principais aplicações já relatadas na literatura. No capítulo 2, são apresentados os principais conceitos utilizados neste trabalho. Os principais trabalhos relacionados são apresentados no capítulo 3. Já nos capítulos 4 e 5, são descritas, respectivamente, as abordagens heurísticas e exatas propostas neste trabalho. No capítulo 6, são descritos os resultados obtidos nos testes computacionais realizados. Por fim, no capítulo 7, são apresentadas as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos utilizados neste trabalho. Na seção 2.1 são abordados os conceitos de Algoritmos Heurísticos e Meta-heurísticos. Além disso, são descritas três Meta-heurísticas utilizadas neste trabalho. Na seção 2.2 são apresentados os conceitos de Programação Linear e Programação Linear Inteira. Por fim, na seção 2.3 são apresentados os conceitos principais de um algoritmo *Branch and Bound*.

2.1 Algoritmos Heurísticos e Meta-heurísticos

Os algoritmos computacionais enfrentam um gigantesco desafio quando o objetivo é fornecer soluções exatas para problemas $\mathscr{NP}$-Difíceis de grande porte (GOLDBARG *et al.*, 2017). De modo geral, não existem algoritmos que encontram soluções ótimas para estes problemas em tempo polinomial, exceto se $P = NP$. Em muitas situações, é mais vantajoso utilizar um algoritmo que forneça uma solução de boa qualidade em tempo polinomial, mesmo que não haja garantia de otimalidade. Neste contexto, os algoritmos heurísticos se apresentam como uma técnica eficiente para resolver tais problemas.

Um algoritmo é considerado heurístico quando não há garantias de soluções ótimas, o algoritmo objetiva resolver problemas complexos utilizando uma quantidade não muito grande de recursos, especialmente no que diz respeito ao consumo de tempo para encontrar soluções de boa qualidade (SUCUPIRA, 2004).

Neste trabalho, foram propostas heurísticas gulosas para o Problema da Máxima Interseção de k-Subconjuntos. Em um problema de maximização, uma heurística gulosa consiste em um método iterativo em que, a cada iteração, busca selecionar para a solução o elemento pertencente a lista de candidatos que maximiza a solução naquele instante.

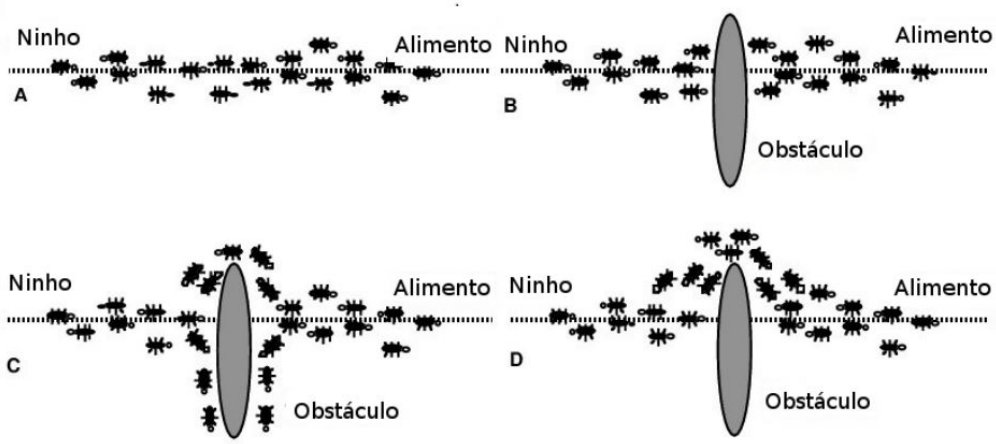
Uma Meta-heurística é um método heurístico que busca resolver, de forma genérica, diversos problemas de otimização combinatória. Para Dorigo *et al.* (2006), uma Meta-heurística pode ser vista como um método heurístico geral projetado para guiar uma heurística específica. Nas próximas subseções, iremos descrever as Meta-heurísticas trabalhadas.

2.1.1 Ant Colony Optimization

A computação, assim como outras áreas do conhecimento, se aproveita de observações feitas na natureza, em busca de padrões que podem ser úteis em seus respectivos campos

de pesquisa. Um comportamento específico pesquisado, assim como citado em Goldbarg *et al.* (2017), é o de *Collective Intelligence*, onde é investigado o fato dos indivíduos se comunicarem com o grupo e como agem em autonomia. Uma das populações estudadas, por apresentar uma característica peculiar de comportamento coletivo, é a população de formigas. Na Figura 3, é apresentado um exemplo do comportamento coletivo das formigas.

Figura 3 – Comportamento coletivo de uma colônia de formigas.



Fonte: (MULATI *et al.*, 2013).

A Figura 3, apresenta o comportamento coletivo das formigas em diferentes situações. No cenário A, não existe nenhum obstáculo entre o ninho e o alimento, fazendo com que todas as formigas sigam o mesmo caminho. No cenário B, é colocado um obstáculo no caminho. No cenário C, é visto que, inicialmente, as formigas não sabem por qual caminho seguir; logo, a probabilidade de escolha de um caminho é $\frac{1}{2}$. Porém, observou-se que, após um tempo, as formigas convergiam ao menor caminho (cenário D). Este fato deve-se a uma substância liberada pelas formigas, conhecida como feromônio, que marca o caminho pelo qual elas passam. Essa substância evapora com o passar do tempo, fazendo com que o menor trajeto contenha uma concentração maior de feromônio, indicando para as próximas formigas que aquele caminho é o melhor a ser seguido.

O comportamento coletivo das formigas, inspirou a Meta-heurística *Ant Colony Optimization* (ACO), também conhecida como Otimização por Colônia de Formigas, apresentada pela primeira vez em Dorigo e Caro (1999), e que se mostrou eficiente aplicada a problemas $\mathscr{NP}$-Difíceis. Segundo Mulati *et al.* (2013), a Meta-heurística ACO é comumente dividida em duas fases principais: construção e atualização de feromônios.

A fase de construção tem como objetivo gerar um conjunto de soluções viáveis. Essa fase é realizada pelas formigas artificiais, termo usado na literatura. Essas formigas nada

mais são do que soluções que serão construídas. Para a construção da solução, é utilizada a probabilidade de um objeto (vértice, aresta, conjunto e entre outros) j ser selecionado para a solução T. Com base na probabilidade, são selecionados os objetos que farão parte da solução. A Equação 2.1, apresenta uma fórmula genérica para o cálculo da probabilidade de um objeto j pertencer a uma solução T.

$$p_j^T = \frac{[\tau_j]^\alpha [\eta_j]^\beta}{\sum_{l \in N(T)} [\tau_l]^\alpha [\eta_l]^\beta} \forall j \in N(T) \tag{2.1}$$

A Equação 2.1 descreve a probabilidade de selecionar o objeto j, dada uma solução em construção T. O conjunto $N(T)$, é o conjunto de objetos ainda não selecionados na construção T. O valor τ_j é referente ao feromônio no objeto j. Já o η_j é um fator guloso para a seleção do objeto j. Os coeficientes α e β são constantes que tratam da importância dos parâmetros τ_j e η_j, para o cálculo de p_j^T.

Na fase de atualização, é realizada uma atualização dos feromônios. O parâmetro τ_j, é comumente atualizado com base na qualidade das soluções geradas, em que o objeto j pertence. O objetivo desta fase, é aumentar o valor do feromônio dos objetos que formam soluções de boa qualidade, semelhante ao que acontece com os caminhos das formigas. A Equação 2.2 mostra como é feita atualização dos feromônios, onde ρ é uma constante que determina o coeficiente de evaporação do feromônio, ou seja, a porcentagem de feromônio que ficará preservada de uma iteração a outra. A variável Δ_j é um valor que remete à qualidade das soluções, geradas naquela iteração, onde o objeto j foi selecionado.

$$\tau_j = (1-\rho)\tau_j + \Delta_j \tag{2.2}$$

2.1.2 Greedy Randomized Adaptive Search Procedure

A Meta-heurística *Greedy Randomized Adaptive Search Procedure*, também conhecida como GRASP, foi inicialmente proposta em Feo e Resende (1989). O GRASP, é composto por duas fases: uma fase de construção, baseada em uma heurística randomizada, e uma fase de busca local, aplicada na solução obtida na primeira fase. Uma heurística de busca local é uma heurística de melhoria que parte de uma solução inicial qualquer e procura sistematicamente por uma solução que seja melhor do que a solução corrente em uma vizinhança considerada (COSTA; URRUTIA, 2009).

Em algumas implementações da Meta-heurística GRASP, após a fase de busca local, também é aplicado um procedimento de intensificação, chamado de *path-relinking*, que procura adicionar atributos das melhores soluções encontradas até o momento na solução da busca local, com o objetivo de gerar soluções de boa qualidade.

No Algoritmo 1, é apresentado o pseudocódigo básico de uma implementação da Meta-heurística GRASP. O algoritmo apresentado possui $IterMax$ iterações, e retorna a melhor solução encontrada durante estas iterações. A cada iteração, uma solução viável é construída (Linha 5) utilizando uma heurística randomizada. Esta solução construída é passada como parâmetro para uma heurística de busca local (Linha 6), onde deve-se explorar a vizinhança da solução construída. Após isso, realiza-se, opcionalmente, um processo de intensificação da melhor solução retornada pela busca local (Linha 7). Por fim, se esta solução encontrada for mais vantajosa que a melhor solução encontrada até o momento, deve-se armazenar tal solução (Linha 8).

Algoritmo 1: Meta-heurística GRASP

```
1  início
2      i ← 1
3      MelhorSolucao ← ∅
4      Enquanto i ≤ IterMax faça
5          Solucao ← Construcao()
6          Solucao ← Busca_Local(Solucao)
7          Solucao ← Path_Relinking(Solucao)
8          Salvar_Solucao(Solucao, MelhorSolucao)
9          i ← i + 1
10     Retorna MelhorSolucao
```

Fonte: (BOGUE *et al.*, 2014b).

2.1.3 Variable Neighborhood Search

A Meta-heurística *Variable Neighborhood Search* (VNS), também conhecida como Busca em Vizinhança Variável, foi inicialmente proposta em Hansen *et al.* (2000). O VNS, consiste em um algoritmo que, dada uma solução inicial, ou a melhor a solução até o momento, realiza diversas buscas na vizinhança desta solução. Durante a realização destas buscas, a amplitude da vizinhança é alterada, a fim de evitar ótimos locais.

O VNS é uma Meta-heurística que propõe explorar o espaço de busca variando sistematicamente as estruturas de vizinhança. As vizinhanças consideradas mais eficientes para

o problema trabalhado, via de regra, possuem maior chance de utilização durante o desenvolvimento do algoritmo (GOLDBARG *et al.*, 2017).

Dada uma solução inicial S, definimos $N_q(S)$ como uma vizinhança que varia com o parâmetro q. A vizinhança $N_q(S)$, pode ser vista, por exemplo, como as soluções viáveis que diferem q elementos da solução S. O Algoritmo VNS, pode ser dividido em duas fases. A fase de diversificação, conhecida como *Shaking*, onde o objetivo é alterar o ponto focal da busca, permitindo, supostamente, a exploração de uma região (vizinhança) diferente (GOLDBARG *et al.*, 2017). A segunda fase do VNS, consiste em realizar uma busca local na solução gerada pela fase anterior.

O Algoritmo 2 apresenta uma implementação básica da Meta-heurística VNS. Inicialmente, define-se a vizinhança N_q para todo $1 \leq q \leq q_{max}$. O algoritmo parte de uma solução viável inicial (Linha 3), que é dada por uma heurística qualquer. O algoritmo possui $IterMax$ iterações, onde para cada iteração é realizado no mínimo $\frac{q_{max}}{q_{step}}$ iterações do laço mais interno. Para cada iteração do laço mais interno, realiza-se um processo de diversificação sobre a vizinhança N_q da melhor solução S encontrada até o momento (Linha 8).

Algoritmo 2: Meta-heurística VNS

```
1  início
2      Selecione vizinhanças N_q, onde 1 ≤ q ≤ q_max
3      S ← Solucao_Inicial()
4      i ← 1
5      Enquanto i ≤ IterMax faça
6          k ← 1
7          Enquanto q ≤ q_max faça
8              S^1 ← Shaking(N_q(S))
9              S^2 ← Busca_Local(N_q(S^1))
10             Se f(S^2) > f(S) então
11                 S ← S^2
12                 q ← 1
13             Senão
14                 q ← q + q_step
15         i ← i + 1
16     Retorna S
```

Fonte: (GOLDBARG *et al.*, 2017).

Observe que, a cada iteração do laço mais interno, o valor de q é modificado, alterando também, possivelmente, a vizinhança N_q de uma iteração para outra. Com a solução S^1, obtida na fase de diversificação, realiza-se um processo de busca local na vizinhança desta

solução (Linha 9). Por fim, utiliza-se a função objetivo f, para verificar se a solução retornada pela busca local é melhor que a melhor solução encontrada até o momento. Se for o caso, deve-se armazenar esta solução e reiniciar a vizinhança N_q, atribuindo a q o valor 1. Caso contrário, q é incrementado em q_{step}, que define o passo de crescimento de q.

2.2 Programação Linear

A Programação Linear (PL), utiliza um modelo matemático para descrever um determinado problema. O adjetivo linear significa que todas as funções matemáticas nesse modelo são funções lineares (HILLIER; LIEBERMAN, 2013). Um modelo de PL, busca descrever as restrições e a função objetivo do problema, utilizando funções lineares. Além disso, neste tipo de modelo, todas as variáveis utilizadas são contínuas.

Em um modelo de programação linear, as restrições lineares devem traduzir exatamente as especificações do problema. Desta forma, as restrições são atendidas se, e somente se, temos uma solução viável para o problema modelado. Além disso, em um problema de maximização, o maior valor obtido pela função objetivo, atendendo todas as restrições, deve corresponder a solução ótima do problema.

Utilizando a forma canônica, temos que todas as restrições são de desigualdade. Considerando um problema de maximização, temos que as restrições são do tipo $\leq$. Sejam $c \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$ vetores de constantes, $x \in \mathbb{R}^n$ o vetor das variáveis de decisão e $A \in \mathbb{R}^{m \times n}$ uma matriz de coeficientes, o modelo geral de PL na forma canônica é apresentado na Equação 2.3.

$$\begin{aligned} \max \quad & c^T x \\ \text{sujeito a} \quad & \\ & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.3}$$

Em 1947, George Dantzig criou um algoritmo eficiente, conhecido como Simplex, para a resolução de problemas de programação linear (NASH, 2000). O Simplex, é um método iterativo exato que busca resolver de forma ótima problemas de programação linear. O algoritmo possui um procedimento que permite identificar, a partir de qualquer solução viável, uma nova solução viável que melhore a solução e permite terminar a busca quando não for mais possível encontrar tal solução (GOLDBARG *et al.*, 2016).

2.2.1 Programação Linear Inteira

Modelos de Programação Linear Inteira (PLI), ou somente Programação Inteira, são modelos de PL onde algumas variáveis possuem restrições de integralidade. Um modelo de otimização constitui um problema de PLI se existe alguma variável do modelo que pode assumir somente valores inteiros (GOLDBARG *et al.*, 2016). Outro caso particular é quando as variáveis são inteiras e restritas aos valores 0 ou 1, esse caso é conhecido como Programação Linear Inteira Binária (WOLSEY, 1998).

Vale ressaltar que a resolução de um modelo de PLI é mais difícil que um modelo de PL, visto que uma instância do problema de PL pode ser resolvida por algoritmos polinomiais, como por exemplo o método dos pontos interiores (KARMARKAR, 1984).

Um conceito muito importante em PLI é o de relaxação linear. Dado um modelo de PLI, a relaxação linear consiste na resolução do problema descrito no modelo retirando as restrições de integralidade. Observe que a solução de uma relaxação linear pode ser obtida de forma rápida utilizando algoritmos como o Simplex. Além disso, a relaxação linear nos fornece limitantes para o problema original.

Comumente, utiliza-se *solvers* para a resolução de modelos de Programação Linear Inteira. Neste trabalho, foi utilizado o *solver* ILOG CPLEX Optimization Studio da IBM.

2.3 Branch and Bound

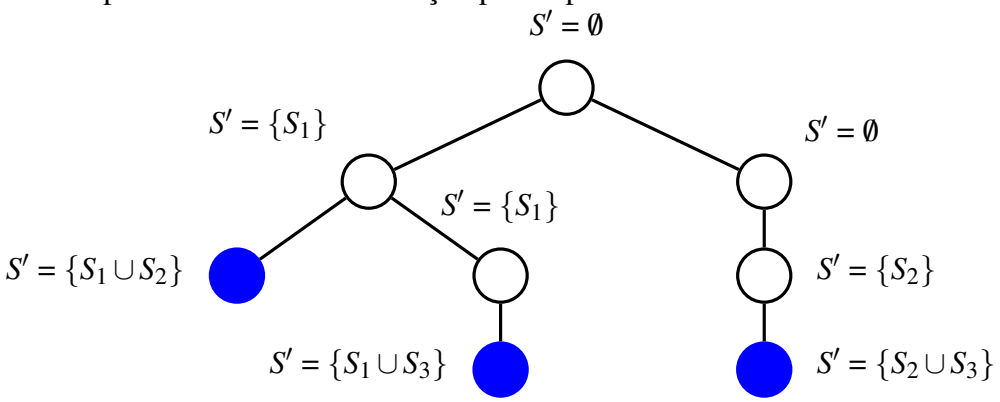
O método *Branch and Bound* (B&B) consiste em uma técnica de enumeração de soluções viáveis para um determinado problema, a fim de determinar a melhor solução. Esta técnica é amplamente utilizada na resolução de problemas, onde as variáveis são inteiras. *Branch and Bound* é de longe a técnica mais amplamente utilizada para resolver problemas de otimização combinatória da classe $\mathscr{NP}$-Difícil (CLAUSEN, 1999).

A árvore de busca, também conhecida como árvore de enumeração, consiste na árvore gerada a partir da ramificação feita para subdividir o problema original em diversos outros subproblemas. Os subespaços de buscas inexplorados são representados como nós da árvore de enumeração (CLAUSEN, 1999).

Um exemplo de ramificação em uma árvore de busca consiste na fixação de variáveis, onde buscamos definir quem deve estar na solução. Suponha um algoritmo B&B para o kMIS em que o processo de fixação de variáveis consiste em expandir para a esquerda, inserindo o

subconjunto S_i na solução parcial S', ou para a direita, garantindo que S_i jamais será inserido em uma solução parcial S'. De forma ilustrativa, suponha uma instância do problema kMIS com $L = \{S_1, S_2, S_3\}$, $R = S_1 \cup S_2 \cup S_3$ e $k = 2$. A Figura 4, ilustra a árvore de enumeração para esta instância.

Figura 4: Exemplo de árvore de enumeração para o problema kMIS.



Fonte: Próprio autor.

Na Figura 4, observe que alguns nós não são ramificados para a direita, pois, como o tamanho da solução precisa ser igual a 2, alguns nós não seriam capazes de gerar soluções deste tamanho. Os nós na cor azul são soluções viáveis para o problema, ou seja, onde o $|S'| = 2$. Observe que não é pertinente esses nós serem expandidos, já que procuramos uma solução que possua exatamente tamanho 2. Para determinar a melhor solução do problema, basta verificar dentre as soluções viáveis qual apresenta a maior interseção. Porém, dado que $|L| = p$, teríamos que investigar dentre $\binom{p}{k}$ possibilidades, qual a melhor solução. Para valores de p relativamente grandes, esta simples investigação pode se tornar totalmente inviável.

A ideia principal do algoritmo B&B consiste em minimizar o crescimento da árvore de enumeração, à medida que está sendo construída, através da realização de podas em nós em que se é possível saber previamente que estes não nos levarão a uma solução ótima do problema (BOGUE *et al.*, 2014b). Uma poda consiste na não ramificação de uma subárvore da árvore de enumeração, diminuindo o espaço de busca.

3 TRABALHOS RELACIONADOS

Na literatura, não existem muitos trabalhos que propõem soluções algorítmicas para o Problema da Máxima Interseção de k-Subconjuntos. Porém, existem dois trabalhos que propõem abordagens exatas e heurísticas para este problema. Nas próximas duas seções, estes trabalhos são descritos. Na seção 3.3 é apresentado um trabalho onde foram propostos algoritmos exatos para o problema da Máxima Biclique Balanceada em grafos bipartidos. Por fim, na seção 3.4 é apresentado uma comparação entre o presente trabalho e os trabalhos descritos neste capítulo.

3.1 O Problema da Máxima Interseção de k-Subconjuntos

O trabalho de Bogue *et al.* (2013) foi o primeiro da literatura a apresentar abordagens algorítmicas para o kMIS. O trabalho apresenta uma heurística gulosa e três formulações de Programação Linear Inteira (PLI). Nas próximas subseções, são descritas as técnicas propostas no trabalho de Bogue *et al.* (2013), bem como os resultados obtidos pelo mesmo.

3.1.1 Heurística kInter

A heurística gulosa apresentada em Bogue *et al.* (2013) é uma heurística que a cada iteração seleciona um vértice $u \in L \setminus L'$, tal que, $|\cap(L' \cup \{u\})|$ é máxima, onde L' é a solução parcial que está sendo construída. A solução parcial L', é inicializada com $\emptyset$. Observe que na primeira iteração, escolhemos o vértice $u \in L$, que possui a maior vizinhança. Essa escolha gulosa é repetida k vezes, conforme apresentado no Algoritmo 3.

Algoritmo 3: HEURÍSTICA KINTER

```
   Entrada: G(L∪R,E),k
1  início
2  |  L' ← ∅
3  |  Enquanto |L'| < k faça
4  |  |  Selecionar um vértice u ∈ L\L' tal que |∩(L'∪{u})| seja máxima.
5  |  |  L' ← L'∪{u}
6  |  Retorna L'
```

Fonte: (BOGUE *et al.*, 2013).

3.1.2 Formulações de Programação Linear Inteira

O problema *k*MIS possui uma estreita relação com o problema *Maximum Edge Biclique* (MEB). O problema MEB pode ser definido como: dado um grafo bipartido $G = (A \cup B, E)$ e um inteiro positivo k, deseja-se encontrar os conjuntos $A' \subseteq A$ e $B' \subseteq B$, onde para todo $u \in A'$ e para todo $v \in B'$, temos que $uv \in E(G)$, de forma que $|A'| \cdot |B'|$ seja máximo.

Em Xavier (2012), o problema *k*MIS é demonstrado ser $\mathscr{NP}$-Difícil, realizando uma redução a partir do problema MEB. Em Bogue *et al.* (2013), foram realizadas adaptações de duas formulações clássicas do problema MEB para o problema *k*MIS. O trabalho também propõe uma nova formulação, denominada de M_ARESTA (BOGUE *et al.*, 2014b). A seguir, cada uma destas formulações é descrita.

3.1.2.1 Formulação MEBV adaptada para o kMIS

Para esta formulação, consideramos as variáveis y_u, como a quantidade de vértices adjacentes de u que pertencem à solução, caso u também pertença a solução, e 0 caso contrário, sendo que $u \in L$ (BOGUE *et al.*, 2013). Já as variáveis z_u, são binárias e dizem se o vértice $u \in V(G)$ pertence ou não à solução. Considerando estas variáveis definidas, apresentamos esta formulação na Equação 3.1.

$$
\begin{aligned}
\max \quad & \frac{1}{k} \sum_{u \in L} y_u \\
\text{sujeito a} \quad & z_u + z_v \leq 1 \quad \forall u \in L \quad \forall v \in \overline{N_G(u)} \\
& y_u \leq |N_G(u)| z_u \quad \forall u \in L \\
& y_u \leq \sum_{v \in N_G(u)} z_v \quad \forall u \in L \\
& \sum_{u \in L} z_u = k \\
& y_u \geq 0 \quad \forall u \in L \\
& z_u \in \{0,1\} \quad \forall u \in V(G)
\end{aligned}
\tag{3.1}
$$

A primeira restrição desta formulação garante que dois vértices u e v não podem ser selecionados ao mesmo tempo para a solução, se $u \in L$ e $v \in \overline{N_G(u)}$. A segunda restrição impõe que y_u seja 0, caso o vértice u não esteja na solução. Caso o vértice u esteja na solução, temos que y_u é menor ou igual a $|N_G(u)|$. A terceira restrição, garante que exatamente k vértices do

conjunto L sejam selecionados para a solução. As duas últimas restrições, garantem o domínio das variáveis utilizadas no modelo. A função objetivo deste modelo visa maximizar a quantidade de elementos na interseção das vizinhanças dos subconjuntos selecionados (BOGUE *et al.*, 2014b). Na formulação MEB V, existem $O(|V|)$ variáveis e $O(|L| \cdot |R|)$ restrições (BOGUE *et al.*, 2013).

3.1.2.2 Formulação MEBL adaptada para o kMIS

Esta formulação utiliza-se das mesmas variáveis utilizadas na formulação MEB V. A única diferença é que as variáveis z_u são definidas somente para $u \in L$, e as variáveis y_v são definidas para $v \in R$. Dada esta descrição de variáveis utilizadas, a formulação MEB L é apresentada na Equação 3.2.

$$
\begin{aligned}
\max \quad & \frac{1}{k} \sum_{v \in R} y_v \\
\text{sujeito a} \quad & y_v \leq \sum_{u \in N_G(v)} z_u \quad \forall v \in R \\
& y_v \leq (1 - z_u)|N_G(u)| \quad \forall u \in L \quad \forall v \in \overline{N_G(u)} \\
& \sum_{u \in L} z_u = k \\
& y_u \geq 0 \quad \forall u \in R \\
& z_u \in \{0,1\} \quad \forall u \in L
\end{aligned}
\tag{3.2}
$$

A primeira restrição desta formulação exige que y_v não seja maior que a quantidade de vértices vizinhos de v que estão na solução. A segunda restrição assegura que, caso $u \in L$ seja escolhido, nenhum vértice $v \in R$ que não seja vizinho a u possa ser escolhido (BOGUE *et al.*, 2014b). A terceira restrição assegura que exatamente k vértices do conjunto L, sejam selecionados para a solução. As duas últimas restrições, garantem o domínio das variáveis utilizadas no modelo. Note que, nesta formulação, a quantidade de variáveis binárias do modelo é reduzida para $O(|L|)$.

3.1.2.3 Formulação M_ARESTA

Para esta formulação são utilizadas as variáveis binárias x_i e y_j. A variável x_i, indica se um vértice $i \in R$ foi ou não escolhido para a solução. Já variável y_j, indica se um vértice $j \in L$

foi ou não escolhido para a solução. Dada a definição das variáveis do modelo, a formulação é apresentada na Equação 3.3.

$$
\begin{aligned}
\max \quad & \sum_{i \in R} x_i \\
\text{sujeito a} \quad & \sum_{j \in L} y_j = k \\
& x_i + y_j \leq 1 \quad \forall j \in L \quad \forall i \in \overline{N_G(j)} \\
& x_i \in \{0,1\} \quad \forall i \in R \\
& y_j \in \{0,1\} \quad \forall j \in L
\end{aligned}
\tag{3.3}
$$

A primeira restrição garante que exatamente k vértices do conjunto L estarão na solução. A segunda restrição assegura que para quaisquer dois vértices i, j, se $j \in L$ e $i \in \overline{N_G(j)} \subseteq R$, então i, j não podem aparecer simultaneamente em uma solução. As duas últimas restrições garantem o domínio das variáveis utilizadas no modelo.

3.1.3 Análise dos Resultados

O trabalho de Bogue *et al.* (2013), gerou instâncias aleatórias e executou as formulações de PLI e a heurística *k*Inter, tendo como entrada essas instâncias. Para comparar as formulações foram utilizadas as métricas *gap* de integralidade e *gap* de dualidade. Foi verificado em Bogue *et al.* (2013), que a formulação M_ARESTA obteve melhores resultados que as demais. Além disso, a heurística *k*Inter gerou soluções com *gap* médio de 23% em relação a solução ótima.

3.2 *An Integer Programming Formulation for the Maximum k-Subset Intersection Problem*

No trabalho de Bogue *et al.* (2014a), é apresentada uma implementação da metaheurística GRASP, denominada de GRASP REATIVO, para resolução do Problema da Máxima Interseção de *k*-Subconjuntos. Foram propostas técnicas de pré-processamento das instâncias, também é apresentado um estudo poliédrico do problema, onde foi possível obter restrições lineares mais fortes para o problema. Utilizando estas restrições, Bogue *et al.* (2014a) propuseram um modelo de PLI, denominado de M_CLIQUE. Nas próximas subseções são descritas estas abordagens, bem como a análise dos resultados obtidos por Bogue *et al.* (2014a).

3.2.1 GRASP Reativo

Nesta subseção é apresentado o algoritmo GRASP REATIVO, desenvolvido por Bogue *et al.* (2014a). Como vimos na subseção 2.1.2, a meta-heurística GRASP é composta por duas fases: uma fase de construção, baseada em uma heurística aleatória, e uma fase de busca local, aplicada na solução obtida na primeira fase. Após a fase de busca local, também é aplicado um procedimento de intensificação, chamado de *path-relinking*, que procura adicionar atributos das melhores soluções encontradas até o momento na solução da busca local, na esperança de gerar soluções de boa qualidade. O que difere entre o algoritmo GRASP tradicional e o algoritmo GRASP REATIVO é a fase de construção. A seguir é descrita a fase de construção do GRASP REATIVO, proposto por Bogue *et al.* (2014a).

3.2.1.1 Fase de Construção

A fase de construção do algoritmo GRASP, consiste em construir iterativamente uma solução viável L' para o problema, sendo inicialmente $L' = \emptyset$. Em cada iteração, um vértice $u \in L$ é escolhido aleatoriamente na Lista Restrita de Candidatos (LRC), para ser adicionado em L'. Esse processo se repete até que $|L'| = k$.

Considerando uma solução parcial L', onde $|L'| < k$, a Lista Restrita de Candidatos é composta pelos vértices $u \in L \setminus L'$, com os maiores custos incrementais $c(u)$, onde $c(u) = |\cap(L' \cup \{u\})|$. O tamanho da LRC, depende de um parâmetro $\alpha \in [0,1]$, que indica o nível de ganância do algoritmo. Dado α, LRC é formado pelos $u \in L \setminus L'$, tal que, $c(u) \in [c^{min} + \alpha \times (c^{max} - c^{min}), c^{max}]$, onde $c^{max} = max_{u \in L \setminus L'} c(u)$ e $c^{min} = min_{u \in L \setminus L'} c(u)$, representando, respectivamente, o maior e o menor custo incremental dos elementos em $L \setminus L'$.

Prais e Ribeiro (2000) desenvolveram um método para que o algoritmo GRASP selecione o valor de α dentro de um conjunto de valores pré-determinado $A = \{\alpha_1, \alpha_2, \ldots, \alpha_v\}$. No GRASP REATIVO, o $\alpha_i \in A$, é selecionado com probabilidade P_i. Esta probabilidade P_i, associada a um valor α_i, é progressivamente alterada de acordo com a qualidade das soluções geradas, onde o valor α_i foi selecionado para a construção da Lista Restrita de Candidatos. Depois de algumas iterações, o GRASP REATIVO tende a utilizar o valor de α_i que obteve em média as melhores soluções.

3.2.2 *Pré-processamento de Instâncias*

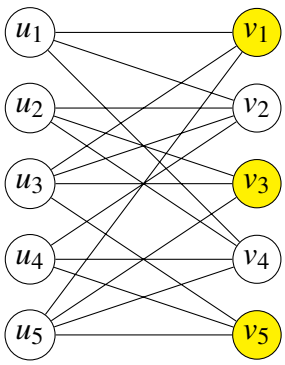
Em Bogue *et al.* (2014a), foi desenvolvido um procedimento de pré-processamento para a redução do tamanho das instâncias de entrada do problema. Seja $\mathscr{L}_u(\lambda) = \{v \in L \setminus \{u\} : |N_G(u) \cap N_G(v)| < \lambda\}, \forall u \in L$ e $\mathscr{R}_v(t) = \{u \in R \setminus \{v\} : |N_G(v) \cap N_G(u)| < t\}, \forall v \in R$, onde λ é o valor de uma solução viável para o problema e $t = k$. O pré-processamento dos vértices de G proposto por Bogue *et al.* (2014a), é realizado da seguinte forma:

1. Todo vértice $u \in L$ com $d_G(u) < \lambda$ ou com $|L| - |\mathscr{L}_u(\lambda)| < k$ pode ser removido de G;
2. Todo vértice $v \in R$ com $d_G(v) < k$ ou com $|R| - |\mathscr{R}_v(k)| < \lambda$ pode ser removido de G;
3. Repita os passos 1 e 2 enquanto for possível reduzir o tamanho da instância.

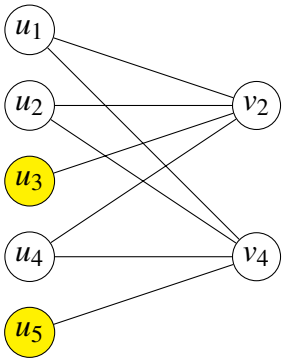
Este procedimento de teste de redução será denominado ao longo deste trabalho de *TR_LR*. De forma ilustrativa, suponha a seguinte instância: $L = \{u_1, u_2, u_3, u_4, u_5\}$, $R = \{v_1, v_2, v_3, v_4, v_5\}$, $N_G(u_1) = \{v_1, v_2, v_4\}$, $N_G(u_2) = \{v_2, v_3, v_4\}$, $N_G(u_3) = \{v_1, v_2, v_3, v_5\}$, $N_G(u_4) = \{v_2, v_4, v_5\}$, $N_G(u_5) = \{v_1, v_3, v_4, v_5\}$, $N_G(v_1) = \{u_1, u_3, u_5\}$, $N_G(v_2) = \{u_1, u_2, u_3, u_4\}$, $N_G(v_3) = \{u_2, u_3, u_5\}$, $N_G(v_4) = \{u_1, u_2, u_4, u_5\}$, $N_G(v_5) = \{u_3, u_4, u_5\}$, $k = 3$ e $\lambda = 2$. Esta instância, denominada Instância *TR_LR*, é apresentada na Figura 5a.

Note que nesta instância $d_G(u) \geq \lambda$ para todo $u \in L$ e $d_G(v) \geq k$ para todo $v \in R$. Além disso, $|L| - |\mathscr{L}_u(\lambda)| \geq k$ para todo $u \in L$. Porém, $|R| - |\mathscr{R}_{v_1}(k)| < \lambda$, $|R| - |\mathscr{R}_{v_3}(k)| < \lambda$ e $|R| - |\mathscr{R}_{v_5}(k)| < \lambda$, logo os vértices v_1, v_3 e v_5 podem ser removidos. A Figura 5b apresenta a instância após a remoção dos vértices v_1, v_3 e v_5. Observe que, após a remoção destes vértices, $d_G(u_3) < \lambda$ e $d_G(u_5) < \lambda$, logo os vértices u_3 e u_5 podem ser removidos. Por fim, na Figura 5c, é apresentado a instância final pré-processada. Note que no grafo resultante $|L| = k = 3$, logo a solução ótima da instância é selecionar todos os k vértices de L.

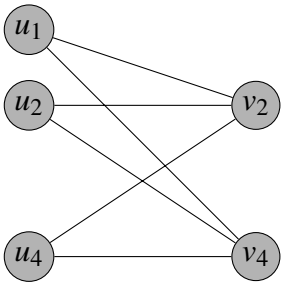
Figura 5: Pré-processamento da Instância *TR_LR*.



(a) Instância *TR_LR*.



(b) Remoção dos vértices v_1, v_3 e v_5.



(c) Instância *TR_LR* pré-processada.

Fonte: Próprio autor.

3.2.3 *Formulação M_CLIQUE*

No trabalho de Bogue *et al.* (2014a), é proposta uma formulação de Programação Linear Inteira mais sofisticada. Inicialmente considere o grafo bipartido $G = (L \cup R, E)$, um número inteiro k e $\lambda = |\cap(L')|$ o valor de uma solução viável $L' \subseteq L$ para o problema. Definimos os conjuntos $\overline{E(G)} = \{uv \notin E(G) \colon u \in L \text{ e } v \in R\}$, $\mathscr{L}(\lambda) = \{uv \in E(G) \colon u, v \in L \text{ e } |N_G(u) \cap N_G(v)| < \lambda\}$ e $\mathscr{R}(k) = \{uv \in E(G) \colon u, v \in R \text{ e } |N_G(u) \cap N_G(v)| < k\}$. Considere o grafo G', onde o conjunto $E(G')$ é dado por $E(G') = \overline{E(G)} \cup \mathscr{L}(\lambda) \cup \mathscr{R}(k)$. O objetivo deste passo é encontrar cliques no grafo G'. Uma clique no grafo G', é um conjunto de vértices $C \subseteq V(G')$, tal que, para quaisquer dois vértices $u, v \in C$, existe uma aresta $uv \in E(G')$.

Porém, encontrar todas as cliques de um grafo tem custo exponencial. Por conta disso, Bogue *et al.* (2014a) utilizaram um algoritmo apresentado em Nemhauser e Sigismondi (1992), que busca encontrar o maior número de cliques possíveis. Para esta formulação são utilizadas as mesmas variáveis binárias x_i e y_j da formulação M_ARESTA. A formulação M_CLIQUE é apresentada pela Equação 3.4.

$$
\begin{aligned}
\max \quad & \sum_{i \in R} x_i \\
\text{sujeito a} \quad & \sum_{j \in L} y_j = k \\
& \sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1, \text{ para toda clique } C \in G' \\
& x_i \in \{0,1\} \quad \forall i \in R \\
& y_j \in \{0,1\} \quad \forall j \in L
\end{aligned}
\tag{3.4}
$$

A primeira restrição garante que exatamente k subconjuntos estarão na solução. A segunda restrição assegura que no máximo um vértice pertencente a uma clique pode ser selecionado para entrar na solução. As duas últimas restrições garantem o domínio das variáveis utilizadas no modelo.

3.2.4 *Análise dos Resultados*

O trabalho de Bogue *et al.* (2014a) utiliza instâncias reais de um *dataset*, onde são obtidos dados de usuários e suas preferências musicais. Todas as instâncias foram submetidas ao pré-processamento, proposto neste mesmo trabalho. A solução inicial para realizar o pré-processamento foi a heurística *k*Inter. Por fim, foram executadas as formulações M_CLIQUE,

M_ARESTA e a meta-heurística GRASP REATIVO, tendo como entrada as instâncias pré-processadas.

Bogue *et al.* (2014a) concluíram, após a realização dos testes computacionais, que a formulação M_CLIQUE consegue encontrar a solução ótima em um maior conjunto de instâncias do que a formulação M_ARESTA. Além disso, observou-se que em 94 % das instâncias, a meta-heurística GRASP REATIVO encontrou a solução ótima, e para as demais instâncias obteve um *gap* médio de 13.5 %.

3.3 *Towards Effective Exact Methods for the Maximum Balanced Biclique Problem in Bipartite Graphs*

No trabalho de Zhou *et al.* (2018), foram propostas abordagens exatas para o problema da Máxima Biclique Balanceada em grafos bipartidos. Este problema, consiste em, dado um grafo bipartido $G(A \cup B, E)$, determinar os conjuntos $A' \subseteq A$ e $B' \subseteq B$, onde $|A'| = |B'|$ e para todo $a \in A'$ e para todo $b \in B'$, existe a aresta $ab \in E(G)$, de forma que a cardinalidade do conjunto A' seja máxima. Assim, como o kMIS, este problema também é uma variação do problema MEB.

Nas próximas subseções, é apresentado o algoritmo *Branch and Bound*, denominado por ExtBBClq, proposto por Zhou *et al.* (2018), além das formulações de programação linear inteira também propostas. Por fim, é apresentado um resumo dos resultados obtidos pelos autores.

3.3.1 ExtBBClq

No trabalho de McCreesh e Prosser (2014), é apresentado um algoritmo *Branch and Bound*, denominado BBClq, para o problema da Máxima Biclique Balanceada, para qualquer tipo de grafo. O algoritmo BBClq era até então o algoritmo exato de melhor desempenho para o problema da Máxima Biclique Balanceada (ZHOU *et al.*, 2018). O trabalho de Zhou *et al.* (2018), propôs uma variação deste algoritmo, denominado ExtBBClq.

O algoritmo ExtBBClq utiliza um procedimento, que também foi proposto por Zhou *et al.* (2018), de propagação do limite superior, onde para cada vértice i é calculado o limite superior ub_i, da solução que contém o vértice i. O Upper Bound Propagation (UBP) é independente do algoritmo *Branch and Bound*. Desta forma, ele executa somente uma vez, antes

da execução do ExtBBClq. Durante o processo de ramificação do algoritmo, o limite ub_i é utilizado como limite superior de um determinado nó, onde, o vértice i está sendo inserido na solução.

3.3.2 Formulações de Programação Linear Inteira

Baseada no algoritmo de propagação de limite superior, Zhou *et al.* (2018) propõem uma nova formulação de programação linear inteira, denominada NILP0, para o problema da Máxima Biclique Balanceada. Esta formulação, utiliza as variáveis x_i, que diz se o vértice i pertence ou não à solução, e a variável z que se trata do valor da solução. Além disso, utiliza-se o parâmetro l, que é igual o maior valor de limite superior para um determinado vértice, retornado por UBP, ou seja, $l = max_{i \in V(G)} ub_i$. A equação 3.5 apresenta a formulação NILP0.

$$
\begin{aligned}
\max \quad & z \\
\text{sujeito a} \quad & z = \sum_{i \in A} x_i \\
& z = \sum_{j \in B} x_j \\
& z - (1 - x_i) l \leq \sum_{j \in N_G(i)} x_j \quad \forall i \in A \cup B \\
& x_i \in \{0,1\} \quad \forall i \in A \cup B \\
& z \in \mathbb{Z}_+
\end{aligned}
\tag{3.5}
$$

As duas primeiras restrições deste modelo garantem que a biclique encontrada seja balanceada, ou seja, cada parte tenha o mesmo tamanho z. Já a terceira restrição garante que caso o vértice i pertença à solução, z seja menor que a quantidade de elementos contidos na vizinhança de i, selecionados para a solução. As duas últimas restrições garantem o domínio das variáveis utilizadas no modelo. Esta formulação tem como objetivo diminuir o número de restrições em relação ao modelo, denominado por ILP0, apresentado em Dawande *et al.* (2001).

Zhou *et al.* (2018) apresentam também um conjunto de restrições válidas, apresentado na Equação 3.6, que podem fortalecer ainda mais os modelos de Programação Linear. Dado um inteiro l, definimos $S^l \subseteq V(G)$, tal que, $i \in S^l$ se, e somente se, $ub_i \leq l$. Definimos também o conjunto T^l como o conjunto maximal de elementos disjuntos em S^l, onde para todo $i, j \in T^l$, temos $N_G(i) \cap N_G(j) = \emptyset$. Além disso, define-se Ω^l, como a coleção de todos os conjuntos

maximais de elementos disjuntos em S^l.

$$\sum_{j \in T^l} (l - ub_j + 1)x_j + \sum_{i \in S^l \setminus T^l} x_i \leq l \quad \forall T^l \subseteq \Omega^l \tag{3.6}$$

No entanto, dado um limite superior válido l, Ω^l pode ser constituído por um número exponencial de elementos disjuntos de S^l (ZHOU *et al.*, 2018). Desta forma, é utilizado um algoritmo de geração de restrições válidas definido em Rossi *et al.* (2014). Por fim, foram gerados mais dois modelos de programação linear inteira, a partir da adição das restrições válidas aos modelos ILP0 e NIPL0 gerando, respectivamente, os modelos denominados ILP1 e NIPL1.

3.3.3 Análise dos Resultados

A fim de verificar a qualidade das abordagens propostas, Zhou *et al.* (2018) realizam a geração de grafos bipartidos aleatórios. A comparação entre algoritmos *Branch and Bound* (BBClq e ExtBBClq) e formulações de programação linear inteira (ILP0, ILP1, NLP0 e NLP1), são realizados a partir do número de nós analisados e pelo tempo de execução. Ao realizar os testes computacionais, foi possível observar que o número de nós analisados pelo ExtBBClq é sempre menor ou igual ao número de nós analisados pelo BBClq. Por consequência, o tempo de execução do algoritmo ExtBBClq é menor. Já entre os modelos de programação linear, foi observado que nos grafos menos densos, os modelos NLP0 e NLP1 apresentam melhores resultados que o ILP0 e ILP1. Para grafos densos, ocorre o contrário.

3.4 Trabalho Proposto

Este trabalho, assim como Bogue *et al.* (2013) e Bogue *et al.* (2014a), propõe novas abordagens heurísticas e exatas para o Problema da Máxima Interseção de k-Subconjuntos. Inicialmente, foram propostas heurísticas simples que são utilizadas como solução inicial para os algoritmos propostos neste trabalho. Além disso, novos algoritmos Meta-heurísticos foram propostos para o problema, visando encontrar soluções de melhor qualidade do que as geradas pelo algoritmo GRASP REATIVO, proposto em Bogue *et al.* (2014a). Foi proposto também uma nova formulação de PLI, visando comparar e obter a melhor formulação para solucionar o kMIS, em um intervalo de tempo limitado. Além disso, foram propostos novos testes de redução, buscando comparar com o que foi proposto em Bogue *et al.* (2014a). Por fim, assim como o trabalho de Zhou *et al.* (2018), foi proposto um algoritmo B&B para o problema.

Vale ressaltar que, assim como os trabalhos Tavares *et al.* (2015) e Tavares *et al.* (2016), neste trabalho, foram utilizadas técnicas de bit paralelismo, para representar e realizar operações de interseção entre conjuntos. A Tabela 1, apresenta uma breve comparação, entre o presente trabalho e os citados durante este capítulo, em relação às técnicas utilizadas.

Tabela 1: Comparação entre os trabalhos relacionados e o presente trabalho.

Abordagem	Trabalhos			
	Bogue *et al.* (2013)	Bogue *et al.* (2014a)	Zhou *et al.* (2018)	Este Trabalho
Algoritmos Heurísticos	X			X
Algoritmos Meta-heurísticos		X		X
Técnicas de Pré-Processamento		X		X
Programação Linear Inteira	X	X	X	X
Branch and Bound			X	X

Fonte: Próprio autor.

4 ABORDAGENS HEURÍSTICAS

Neste capítulo são descritas as abordagens heurísticas desenvolvidas neste trabalho para o Problema da Máxima Interseção de k-Subconjuntos.

4.1 Heurísticas Gulosas

Nesta seção são descritas as heurísticas gulosas propostas neste trabalho. O objetivo destas heurísticas é fornecer uma solução inicial viável de boa qualidade.

4.1.1 kInter Estendido

Esta heurística proposta consiste na execução da heurística KINTER, apresentada na Subseção 3.1.1, $|L|$ vezes, onde a cada iteração, a solução inicial L' é inicializada com um vértice $u \in L$ distinto. Deste modo, selecionamos de forma gulosa apenas $k-1$ vértices em $L \setminus \{u\}$. O algoritmo deve retornar a melhor solução encontrada entre as $|L|$ soluções viáveis geradas. O Algoritmo 4, apresenta o pseudocódigo da heurística KINTER ESTENDIDO.

Algoritmo 4: HEURÍSTICA KINTER ESTENDIDO

Entrada: $G(L \cup R, E), k$

início
  $L^\star \leftarrow \emptyset$
  Para $u \in L$ **faça**
    $L' \leftarrow \{u\}$
    Enquanto $|L'| < k$ **faça**
      Selecionar um vértice $v \in L \setminus L'$ tal que $|\cap(L' \cup \{v\})|$ seja máxima.
      $L' \leftarrow L' \cup \{v\}$
    Se $|\cap(L')| > |\cap(L^\star)|$ **então**
      $L^\star \leftarrow L'$
  Retorna $L^\star$

Fonte: Próprio autor.

O objetivo da heurística KINTER ESTENDIDO é gerar diversas soluções. Vale ressaltar que o vértice u de maior vizinhança, que sempre estará presente em todas as soluções geradas pela heurística KINTER, nem sempre gerará uma solução de boa qualidade.

4.1.2 kInter Estendido Path-Relinking

O procedimento *Path-Relinking*, originalmente proposto em Glover (1997), é comumente utilizado como procedimento de intensificação no algoritmo GRASP. Durante esse procedimento, um conjunto das melhores soluções encontradas, denotado por $Elite$, é mantido. O *Path-relinking* constrói um caminho de soluções viáveis conectando a solução de entrada L_i até uma solução $L_a \in Elite$, gerando várias soluções intermediárias. Para geração de soluções intermediárias é aplicada uma operação de movimento, denotada por $\oplus$, onde busca-se modificar a solução atual, denominada solução origem L_{origem}, a fim de que ela fique mais semelhante à solução final $L_{destino}$, denominada solução de destino.

Um movimento m consiste na remoção de um vértice u, onde $u \in L_{origem}$ e $u \notin L_{destino}$, e na inserção do vértice v, onde $v \notin L_{origem}$ e $v \in L_{destino}$. Quando $L_{origem} = L_{destino}$, não podemos mais realizar nenhum movimento. Dizemos que m é um melhor movimento, quando não existe um movimento m' tal que, $|\cap(L_{origem} \oplus m')| > |\cap(L_{origem} \oplus m)|$. Desta forma, para computar um melhor movimento m deve-se verificar todas as possibilidades de remover um vértice u, onde $u \in L_{origem}$ e $u \notin L_{destino}$, e inserir um vértice v, onde $v \notin L_{origem}$ e $v \in L_{destino}$.

Algoritmo 5: Path-Relinking

```
   Entrada: Solução L_i.
1  início
2      Selecione aleatoriamente L_a ∈ Elite
3      Se |∩(L_i)| > |∩(L_a)| então
4          L_origem ← L_i; L_destino ← L_a; L* ← L_i
5      Senão
6          L_origem ← L_a; L_destino ← L_i; L* ← L_a
7      Enquanto L_origem ≠ L_destino faça
8          Compute o melhor movimento m
9          L_origem ← L_origem ⊕ m
10         Se |∩(L_origem)| > |∩(L*)| então
11             L* ← L_origem
12     Retorna L*
```

Fonte: Próprio autor.

Ribeiro *et al.* (2002) observaram que o procedimento *Path-relinking* atinge melhores resultados quando a solução origem é melhor que a solução destino, pois a vizinhança da solução origem é mais cuidadosamente explorada. Desta forma, dada uma solução de entrada L_i e uma solução L_a selecionada arbitrariamente do conjunto $Elite$, vamos escolher a melhor solução

para ser a solução de origem L_{origem} e a outra será a solução de destino $L_{destino}$. Após isso, vamos efetuar uma sequencia de movimentos em L_{origem}, até que $L_{origem} = L_{destino}$. A melhor solução encontrada durante todo o procedimento será retornada. No Algoritmo 5, é apresentado o pseudocódigo do procedimento *Path-Relinking*.

A heurística kINTER ESTENDIDO PATH-RELINKING consiste na execução do procedimento de intesificação *Path-Relinking* a cada solução encontrada pela heurística kINTER ESTENDIDO. O objetivo desta heurística é encontrar soluções de boa qualidade que possivelmente não seriam geradas pela heurística kINTER ESTENDIDO.

4.2 Algoritmos Meta-heurísticos

Nesta seção, serão descritos dois algoritmos Meta-heurísticos desenvolvidos. Na subseção 4.2.1, é descrito um algoritmo *Variable Neighborhood Search* (VNS), denominado VNS REATIVO. Já na subseção 4.2.2, é descrito um algoritmo *Ant Colony Optimization*, denominado ACO kMIS. Esses algoritmos foram desenvolvidos com o objetivo de melhorar a qualidade da solução em relação ao algoritmo GRASP REATIVO, descrito na Subseção 3.2.1.

4.2.1 VNS Reativo

Na Subseção 2.1.3, é apresentado o algoritmo *Variable Neighborhood Search* (VNS) básico. Comumente são inseridos no algoritmo VNS básico outros procedimentos que visam melhorar seu desempenho. O pseudocódigo do VNS REATIVO é apresentado no Algoritmo 6. Além do grafo e o parâmetro k, o algoritmo recebe como entrada uma solução inicial. A solução inicial $L_{inicial}$ do Algoritmo 6 é dada pela heurística kINTER ESTENDIDO PATH-RELINKING, apresentada na Subseção 4.1.2. Esta heurística foi escolhida pelo seu bom desempenho em testes computacionais iniciais.

No algoritmo proposto, é utilizada a Lista Restrita de Candidatos (LRC) na fase de agitação. Esta lista é utilizada na fase de construção do algoritmo GRASP REATIVO, descrito em 3.2.1.1. A construção desta lista depende do parâmetro $0 \leq \alpha \leq 1$. Este parâmetro define a ganância do algoritmo. Se α assume valores próximos de 1, o algoritmo tenderá a tomar decisões gulosas. Já quando α assume valores próximos de 0, as decisões serão mais aleatórias. Nos algoritmos GRASP REATIVO e VNS REATIVO, o parâmetro α é reativo, ou seja, ele será escolhido com base em uma probabilidade, e esta probabilidade será modificada com base na

qualidade da solução gerada.

Assim como no GRASP REATIVO, o parâmetro α será escolhido dentro de uma lista de possíveis $\alpha_i = 0.1 \cdot i$, onde $1 \leq i \leq 10$. Todo α_i possui uma probabilidade P_i de ser selecionado. Inicialmente, todos os possíveis α_i recebem a mesma probabilidade P_i de serem selecionados (Linhas 3-5). Porém, esta probabilidade é atualizada com base na qualidade das soluções geradas (Linhas 18-22). O parâmetro α é selecionado (Linha 12) com base nas probabilidades dos α_i. Desta forma, o algoritmo irá convergir a selecionar α_i que gere as melhores soluções.

Algoritmo 6: VNS Reativo

```
   Entrada: G(L∪R,E), k, L_inicial
1  início
2      Z_max ← 0;
3      Para i ← 1 to 10 faça
4          Inicialize cont(i) e values(i) com 0;
5          P_i ← 1/10;
6      L', L* ← L_inicial;
7      t ← 1;
8      Para j ← 1 to seqs faça
9          q ← 1;
10         iter_sem_melhoria ← 0;
11         Enquanto q ≤ q_max faça
12             α_i ← selectionAlpha();
13             L'' ← shakingGrasp(L', q, α_i);
14             L'' ← BuscaLocal(L'', k)
15             CriterioAceitacao(L', L'');
16             CriterioParada(L', L'');
17             Z_max ← |∩(L*)|
18             cont(i) ← cont(i) + 1
19             values(i) ← values(i) + c(L')
20             Se t ≡ 0 (mod 10) e cont(j) ≠ 0, ∀j = 1...10 então
21                 Calcule media(j) e Q_j para todo j = 1...10
22                 Calcule σ = Σ_{j=1}^{v} Q_j e atualize P_j = Q_j/σ para todo j = 1...10
23             t ← t + 1;
24     Retorna L*
```

Fonte: Próprio autor.

Após a seleção do parâmetro α, será realizado, respectivamente, os procedimentos de agitação da solução, busca local, critério de aceitação e o critério de parada. A seguir é descrito cada um destes procedimentos.

4.2.1.1 Fase de Agitação

A fase de agitação do algoritmo proposto consiste em remover q vértices da solução inicial e utilizar o procedimento da fase de construção do algoritmo GRASP REATIVO, que consiste em inserir q vértices aleatórios contidos na Lista Restrita de Candidatos (LRC). O procedimento utilizado, denominado *ShakingGrasp*, tem como entrada uma solução L', um inteiro $q < k$ e um valor real $0 \leq \alpha \leq 1$. O parâmetro α define o tamanho da LRC. Quanto maior o valor de α, mais a escolha do vértice a ser inserido na solução construída se aproxima da escolha gulosa. Além disso, o tamanho da LRC decresce.

A cada iteração do algoritmo é atualizado os custos incrementais de cada vértice que não pertence a solução, ou seja, para todo vértice $u \in L \setminus L_s$ calcula-se $c(u) = |\cap (L_s \cup \{u\})|$. Após esta etapa, é construído a LRC utilizando o parâmetro α e a solução em construção L_s. Após as construção da LRC, seleciona-se aleatoriamente um vértice $u \in LRC$. Após isso, inserimos o vértice u selecionado na solução L_s. O algoritmo finaliza quando $|L_s| = k$. O pseudocódigo do algoritmo *ShakingGrasp*, é descrito no Algoritmo 7.

Algoritmo 7: ShakingGrasp

Entrada: L', q, α
1 $L_s \leftarrow L'$;
2 Remova q vértices aleatórios de L_s;
3 **Enquanto** $|L_s| < k$ **faça**
4 Atualize $c(u)$ para todo vértice $u \in L \setminus L_s$;
5 Constrói *LRC* utilizando α e L_s;
6 Secione aleatoriamente $u \in LRC$;
7 $L_s \leftarrow L_s \cup \{u\}$;
8 **Retorna** L_s;

Fonte: Próprio autor.

4.2.1.2 Busca Local

A busca local no algoritmo VNS REATIVO consiste na remoção de t vértices aleatórios da solução gerada pela fase de agitação. Após isso, será inserido t vértices de forma gulosa, seguindo a estratégia da heurística KINTER, apresentada na Subseção 3.1.1. Após testes empíricos realizados, definimos o parâmetro $t = 0.3 \cdot k$. O pseudocódigo da busca local é apresentada no Algoritmo 8.

Algoritmo 8: Busca Local

Entrada: L'', k
1 $t \leftarrow 0.3 \cdot k$;
2 $L_s \leftarrow L''$;
3 Remova t vértices aleatórios de L_s;
4 **Enquanto** $|L_s| < k$ **faça**
5 | Selecione $u \in L \setminus L_s$, tal que $|\cap(L_s \cup \{u\})|$ seja máximo;
6 | $L_s \leftarrow L_s \cup \{u\}$
7 **Retorna** L_s;

Fonte: Próprio autor.

4.2.1.3 Critério de Aceitação de Solução

Em um algoritmo VNS, comumente, a solução que possui sua vizinhança explorada é a melhor solução encontrada até o momento. Desta forma, armazenamos/aceitamos a solução retornada pela busca local se esta solução for melhor que a melhor solução encontrada até o momento. Em Hansen *et al.* (2000), é apresentado um novo critério de aceitação da solução retornada pela busca local no VNS, chamado de Skewed VNS. A ideia deste procedimento é fazer com que o algoritmo VNS não fique preso em um ótimo local, e passará a aceitar soluções piores, desde que sejam significativamente distintas da melhor solução encontrada. Para o cálculo da diferença de estrutura entre duas soluções viáveis, iremos utilizar a diferença simétrica de Bogue *et al.* (2014a). A diferença simétrica, denotada por δ, é definida como a quantidade de vértices diferentes que existem entre as soluções.

Algoritmo 9: Critério de Aceitação

Entrada: L', L''
1 **Se** $|\cap(L'')| > |\cap(L')|$ **então**
2 | $L' \leftarrow L''$;
3 **Senão**
4 | $\varepsilon = k * get\_quartil(q_{max}, q)$;
5 | **Se** $|\cap(L'')| = |\cap(L')|$ *e* $\delta(L', L'') > \varepsilon$ **então**
6 | | $L' \leftarrow L''$;

Fonte: Próprio autor.

Uma solução que não seja melhor que a melhor solução encontrada será mais facilmente aceita se q for mais próximo de q_{max}, pois isto nos diz que a vizinhança atual foi mais vezes explorada. Para nos auxiliar neste processo, definimos a função *get_erro_quartil*. Esta função recebe como parâmetro os valores *q_max* e q, e calcula em qual quartil de *q_max*

está q. Dependendo do quartil, a função retorna 0.2, 0.3, 0.4 e 0.5 para o quartil 1, 2, 3 e 4, respectivamente. O pseudocódigo do Critério de Aceitação está descrito no Algoritmo 9.

4.2.1.4 Critério de Parada

Em um algoritmo *Variable Neighborhood Search* (VNS), o parâmetro q define o nível de agitação na vizinhança. Este parâmetro é incrementado por q_{step}, ou seja, a cada iteração do algoritmo $q = q + q_{step}$. Na literatura, normalmente se utiliza um valor constante para o q_step, escolhido depois de testes computacionais. Porém, no algoritmo proposto, este parâmetro é alterado com o objetivo de, quando o algoritmo não conseguir encontrar nenhuma melhor solução na vizinhança, o parâmetro q cresça mais rapidamente e, por consequência, o tempo de execução seja reduzido.

Algoritmo 10: Critério de Parada

```
   Entrada: L'', L*
1  Se |∩(L'')| > |∩(L*)| então
2  |  L* ← L'';
3  |  q ← 1;
4  |  iter_sem_melhoria ← 0;
5  |  Se q_step > 1 então
6  |  |  q_step ← q_step − 1;
7  Senão
8  |  Se q_step < q_step_max então
9  |  |  iter_sem_melhoria ← iter_sem_melhoria + 1;
10 |  |  Se iter_sem_melhoria ≥ β * q_max então
11 |  |  |  q_step ← q_step + 1;
12 |  |  |  iter_sem_melhoria ← 0;
13 |  Se q < q_max e q + q_step > q_max então
14 |  |  q ← q_max;
15 |  Senão
16 |  |  q ← q + q_step;
```

Fonte: Próprio autor.

No algoritmo proposto, o valor do q_step é inicializado com 1, e será alterado depois de certa quantidade de iterações sem melhora na solução, denotada por $iter\_sem\_melhoria$. Quando $iter\_sem\_melhoria$ supera $\beta \cdot q\_max$, então o q_step é incrementado por 1, onde $0 < \beta \leq 1$ representa a proporção que desejamos manter o valor do q_step inalterado sem que tenha ocorrido melhora. Após alguns testes computacionais, escolhemos o valor $\beta = 0.4$. Quando o algoritmo encontra uma solução melhor que a melhor solução já encontrada, o parâmetro q

é reinicializado como 1 e, se $q_{step} > 1$, então q_{step} é decrementado em 1. O pseudocódigo de atualização q_{step} é apresentado no Algoritmo 10.

4.2.2 ACO kMIS

Neste trabalho, foi desenvolvido um algoritmo *Ant Colony Optimization* (ACO), para o Problema da Máxima Interseção de k-SUbconjuntos, denominado ACO KMIS. No Algoritmo 11, é apresentado o pseudocódigo do ACO KMIS. Como foi visto na Subseção 2.1.1, um algoritmo ACO se utiliza de formigas artificiais, que são soluções que estão sendo construídas. O algoritmo proposto é executado $IterMax$ iterações, onde a cada iteração do algoritmo serão construídas $|L|$ soluções. Inicialmente, toda solução L_u é inicializada com um vértice $u \in L$ distinto (Linhas 8-9).

Para a construção da solução L_u obtida pela formiga u, o algoritmo ACO KMIS utiliza a probabilidade p^u_{ij}, que significa a probabilidade de selecionar um vértice $j \in L \setminus L_u$ para solução L_u, dado que $i \in L$ foi o último vértice a ser inserido na solução L_u. Para o cálculo da probabilidade p^u_{ij}, descrito a Equação 4.1, utiliza-se o feromônio τ_{ij} e o fator guloso η^u_j.

$$p^u_{ij} = \frac{[\tau_{ij}]^\alpha [\eta^u_j]^\beta}{\sum_{l \in L \setminus L_u} [\tau_{il}]^\alpha [\eta^u_l]^\beta} \quad \forall j \in L \setminus L_u \tag{4.1}$$

O feromônio é um fator dinâmico, que é inicializado com τ_0 (Linhas 2-4) e modificado com base na qualidade das soluções geradas. O fator guloso η^u_j nos fornece a qualidade da solução parcial $L_u \cup \{j\}$ em relação a solução parcial L_u, o cálculo do fator $\eta^{L_u}_j$ é descrito na Equação 4.2.

$$\eta^u_j = \frac{|\cap(L_u \cup \{j\})|}{|\cap(L_u)|} \quad \forall j \in L \setminus L_u \tag{4.2}$$

Na construção da solução, o vértice $j \in L \setminus L_u$ com maior probabilidade p^u_{ij} será inserido na solução (Linhas 16-17). A construção da solução L_u finaliza quando $|L_u| = k$. Neste ponto, verificamos se $|\cap(L_u)| > |\cap(L^*)|$, onde L^* armazena a melhor solução encontrada. Caso isso aconteça, atualizamos L^* (Linhas 19-20).

Na fase de atualização, iremos atualizar o feromônio τ_{ij} (Linhas 36-38). O cálculo de atualização, é apresentado na Equação 4.3, onde ρ é o coeficiente de evaporação do feromônio. Além disso, Δ_{ij} mede a qualidade média das soluções L_u geradas por alguma formiga u, onde

$i \in L_u$ e $j \in L_u$, em relação a qualidade da melhor solução encontrada L^*. Caso não exista solução L_u, tal que, $i \in L_u$ e $j \in L_u$, então $\Delta_{ij} = 0$. Desta forma, se os vértices $i, j \in L$ pertencem a soluções de boa qualidade, então o parâmetro τ_{ij} tende a aumentar, e consequentemente o vértice j terá maior probabilidade de ser selecionado em uma solução que o vértice i já pertence.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \Delta_{ij} \quad \forall i \in L \quad \forall j \in L \setminus \{i\} \tag{4.3}$$

Por fim, a melhor solução encontrada L^* é retornada ao fim do algoritmo (Linha 40).

Algoritmo 11: ACO kMIS

Entrada: $G(L \cup R, E)$, k

Para $u \in L$ **faça**
  Para $v \in L \setminus \{u\}$ **faça**
    $\tau_{ij} \leftarrow \tau_0$
$L^* \leftarrow \emptyset$, $iter \leftarrow 1$
Enquanto $iter \leq IterMax$ **faça**
  Para $u \in L$ **faça**
    $L_u \leftarrow \{u\}$
  Para $u \in L$ **faça**
    $i \leftarrow u$
    Enquanto $|L_u| < k$ **faça**
      Para $j \in L \setminus L_u$ **faça**
        $\eta_j^{L_u} \leftarrow \frac{|\cap(L_u \cup \{j\})|}{|\cap(L_u)|}$
      Para $j \in L \setminus L_u$ **faça**
        $p_{ij}^{L_u} \leftarrow \frac{[\tau_{ij}]^\alpha [\eta_j^{L_u}]^\beta}{\sum_{l \in L \setminus L_u} [\tau_{il}]^\alpha [\eta_l^{L_u}]^\beta}$
      Selecione j, tal que, $p_{ij}^{L_u}$ seja máximo.
      $L_u \leftarrow L_u \cup \{j\}$
      $i \leftarrow j$
    Se $|\cap(L_u)| > |\cap(L^*)|$ **então**
      $L^* \leftarrow L_u$
  Para $i \in L$ **faça**
    Para $j \in L \setminus \{i\}$ **faça**
      $\Delta_{ij}, Q_{ij} \leftarrow 0$
  Para $u \in L$ **faça**
    Para $i \in L$ **faça**
      Para $j \in L \setminus \{i\}$ **faça**
        Se $i \in L_u$ *e* $j \in L_u$ **então**
          $\Delta_{ij} \leftarrow \Delta_{ij} + |\cap(L_u)|$
          $Q_{ij} \leftarrow Q_{ij} + 1$
  Para $i \in L$ **faça**
    Para $j \in L \setminus \{i\}$ **faça**
      Se $Q_{ij} > 0$ **então**
        $\Delta_{ij} \leftarrow \frac{\Delta_{ij}}{Q_{ij}}$
        $\Delta_{ij} \leftarrow \frac{\Delta_{ij}}{|\cap(L^*)|}$
  Para $i \in L$ **faça**
    Para $j \in L \setminus \{i\}$ **faça**
      $\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \Delta_{ij}$
  $iter \leftarrow iter + 1$
Retorna L^*

Fonte: Próprio autor.

5 ABORDAGENS EXATAS

Neste capítulo, são descritas as abordagens exatas desenvolvidas neste trabalho para o Problema da Máxima Interseção de k-Subconjuntos.

5.1 Testes de Redução

Neste trabalho, assim como no de Bogue *et al.* (2014a), são propostos novos testes de redução do tamanho da dimensão do problema, denominados *TR_Strong*. Estes testes de redução podem ser utilizados para realização de um pré-processamento das instâncias de teste, visando diminuir o número de vértices do grafo, facilitando a resolução do problema, ou ainda, como critério de fixação em algoritmo de *Branch and Bound*. Os testes de redução propostos dominam os testes de redução propostos por Bogue *et al.* (2014a).

Dado os conjuntos $\mathscr{L}_u(\lambda) = \{v \in L \setminus \{u\} \colon |N_G(u) \cap N_G(v)| < \lambda\}$, $\forall u \in L$ e $\mathscr{R}_v(t) = \{u \in R \setminus \{v\} \colon |N_G(v) \cap N_G(u)| < t\}$, $\forall v \in R$, definimos $\overline{\mathscr{L}_u}(\lambda) = L \setminus (\mathscr{L}_u(\lambda) \cup \{u\})$, $\forall u \in L$ e $\overline{\mathscr{R}_v}(t) = R \setminus (\mathscr{R}_v(t) \cup \{v\})$, $\forall v \in R$. Definimos o conjunto $C_L(u) = \{v \in N_G(u) \colon |\overline{\mathscr{L}_u}(\lambda) \cap N_G(v)| \geq k-1\}$, como o subconjunto de vértices em $N_G(u) \subseteq R$, tal que pertença também a vizinhança de pelo menos $k-1$ vértices de $\overline{\mathscr{L}_u}(\lambda)$. De forma análoga, definimos o conjunto $C_R(v) = \{u \in N_G(v) \colon |\overline{\mathscr{R}_v}(t) \cap N_G(u)| \geq \lambda - 1\}$, como o subconjunto de vértices em $N_G(v) \subseteq L$, tal que pertença também a vizinhança de pelo menos $k-1$ vértices de $\overline{\mathscr{R}_v}(t)$. O pré-processamento dos vértices de G proposto neste trabalho, é realizado da seguinte forma:

1. Todo vértice $u \in L$ com $|C_L(u)| < \lambda$ pode ser removido de G;
2. Todo vértice $v \in R$ com $|C_R(v)| < k$ pode ser removido de G;
3. Repita os passos 1 e 2 enquanto for possível reduzir o tamanho da instância.

No passo 1, se $|C_L(u)| < \lambda$, então não existem λ vértices na vizinhança de u em G que também pertençam a vizinhança de pelo menos $k-1$ vértices de $\overline{\mathscr{L}_u}(\lambda)$. Os vértices em $\overline{\mathscr{L}_u}(\lambda)$ são os únicos vértices em L que podem estar em uma solução viável L', tal que $u \in L'$ e $|\cap(L')| \geq \lambda$. Portanto, se $|C_L(u)| < \lambda$, então $|\cap(L')| < \lambda$ para qualquer solução viável L' contendo u. Logo, o vértice u pode ser removido do grafo G.

No passo 2, se $|C_R(v)| < k$, então não existem k vértices na vizinhança de v em G que também pertençam a vizinhança de pelo menos $\lambda - 1$ vértices de $\overline{\mathscr{R}_v}(t)$. Como os vértices em $\overline{\mathscr{R}_v}(t)$ são os únicos vértices em R, que dado uma solução viável L' com $v \in \cap(L')$ e $\cap(L') \subseteq \overline{\mathscr{R}_v}(t) \cup \{v\}$. Portanto, se $|C_R(v)| < k$, então não existe uma solução viável L' com

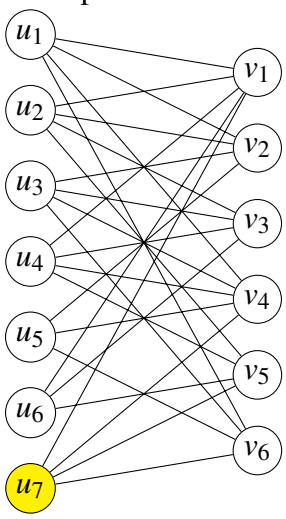
exatamente k vértices, tal que $v \in \cap(L')$. Logo, o vértice v pode ser removido do grafo G.

Assim como nos testes de redução de Bogue *et al.* (2014a), a remoção de qualquer vértice pode afetar outro vértice que até então não atendia as restrições do passo 1 ou 2. Dessa forma, repetimos os passos 1 e 2, até que não seja mais possível remover nenhum vértice de G.

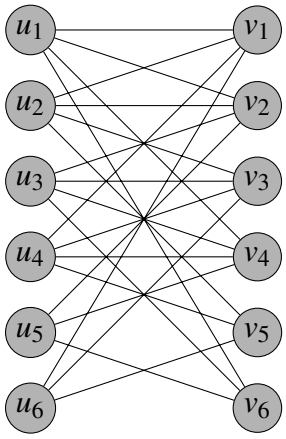
De forma ilustrativa, suponha a seguinte instância: $L = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$, $R = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, $N_G(u_1) = \{v_1, v_2, v_4, v_6\}$, $N_G(u_2) = \{v_1, v_2, v_3, v_5\}$, $N_G(u_3) = \{v_2, v_3, v_4, v_6\}$, $N_G(u_4) = \{v_1, v_3, v_4, v_5\}$, $N_G(u_5) = \{v_2, v_4, v_6\}$, $N_G(u_6) = \{v_1, v_3, v_5\}$, $N_G(u_7) = \{v_1, v_4, v_5, v_6\}$, $N_G(v_1) = \{u_1, u_2, u_4, u_6, u_7\}$, $N_G(v_2) = \{u_1, u_2, u_3, u_5\}$, $N_G(v_3) = \{u_2, u_3, u_4, u_6\}$, $N_G(v_4) = \{u_1, u_3, u_4, u_5, u_7\}$, $N_G(v_5) = \{u_2, u_4, u_6, u_7\}$, $N_G(v_6) = \{u_1, u_3, u_5, u_7\}$, $k = 3$ e $\lambda = 3$. Esta instância, denominada Instância TR_Strong, é apresentada na Figura 6a.

Note que $d_G(u) \geq \lambda$ e $|L| - |\mathscr{L}_u(\lambda)| \geq k$, para todo $u \in L$. Além disso, $d_G(v) \geq k$ e $|R| - |\mathscr{R}_v(k)| \geq \lambda$, para todo $v \in R$. Desta forma, nenhum vértice seria removido no teste proposto em Bogue *et al.* (2014a). Porém, note que $|C_L(u_7)| < \lambda$, logo u_7 pode ser removido. A Figura 6b, apresenta a instância após a remoção do vértice u_7.

Figura 6: Pré-processamento da Instância TR_Strong.



(a) Instância TR_Strong.



(b) Instância TR_Strong pré-processada.

Fonte: Próprio autor.

A seguir, são apresentadas as demonstrações da dominância dos testes de redução apresentados aqui, em relação aos testes de redução apresentados em Bogue *et al.* (2014a).

Proposição 1. *Seja u um vértice do conjunto L e inteiros $k, \lambda > 0$. Se $d_G(u) < \lambda$ ou $|L| - |\mathscr{L}_u(\lambda)| < k$ então $|C_L(u)| < \lambda$.*

Demonstração. Observe inicialmente que $C_L(u) \subseteq N_G(u)$, logo $|C_L(u)| \leq |N_G(u)|$. Se $d_G(u) =$

$|N_G(u)| < \lambda$ então $|C_L(u)| \leq |N_G(u)| < \lambda$. Observe também que $|L| = |\mathscr{L}_u(\lambda)| + |\overline{\mathscr{L}_u}(\lambda)| + 1$, logo $|\overline{\mathscr{L}_u}(\lambda)| = |L| - |\mathscr{L}_u(\lambda)| - 1$. Se $|L| - |\mathscr{L}_u(\lambda)| < k$, então $|\overline{\mathscr{L}_u}(\lambda)| = |L| - |\mathscr{L}_u(\lambda)| - 1 < k-1$. Como $|\overline{\mathscr{L}_u}(\lambda)| < k-1$, para qualquer vértice $v \in N_G(u)$, temos que $|\overline{\mathscr{L}_u}(\lambda) \cap N_G(v)| < k-1$. Portanto, se $|L| - |\mathscr{L}_u(\lambda)| < k$, temos que $C_L(u) = \emptyset$ e $|C_L(u)| = 0 < \lambda$. □

Proposição 2. *Seja v um vértice do conjunto R e inteiros $k, \lambda > 0$. Se $d_G(v) < k$ ou $|R| - |\mathscr{R}_v(k)| < \lambda$ então $|C_R(v)| < k$.*

A demonstração da Proposição 2 é análoga a demonstração da Proposição 1. Como solução inicial utilizamos a solução da heurística gulosa kINTER ESTENDIDO. Vale ressaltar que, dada a solução L' gerada pela heurística, utilizamos o parâmetro $\lambda = |\cap(L')| + 1$, pois em abordagens exatas buscaremos encontrar uma solução de valor pelo menos $\lambda + 1$, e caso não exista sabemos que a solução L' é uma solução ótima para aquela instância.

5.2 Branch and Bound

Nesta seção, apresentamos um algoritmo de *Branch and Bound*, denominado BBK-MIS, para o Problema da Máxima Interseção de k-Subconjuntos. De maneira geral, um algoritmo de *Branch and Bound* realiza uma enumeração implícita de todas as soluções viáveis de um problema. Na modelagem do Problema da Máxima Interseção de k-Subconjuntos, como um grafo bipartido $G = (L \cup R, E)$, uma solução viável L', é um conjunto com k vértices de L, ou seja, $L' \subseteq L$ e $|L'| = k$. Definimos a função $opt(G,k)$ como o valor da solução ótima para uma instância do problema trabalhado. A seguir, são apresentadas duas proposições e suas respectivas demonstrações. Estas proposições serão utilizadas ao longo desta subseção.

Proposição 3. *Seja um grafo bipartido $G = (L \cup R, E)$ e um inteiro k. Considere uma ordenação dos vértices de L, tal que $d_G(v_1) \geq d_G(v_2) \geq \cdots \geq d_G(v_{|L|})$. Se $UB = d_G(v_k)$, então $opt(G,k) \leq UB$.*

Demonstração. Suponha uma solução L' ótima viável da instância G,k, ou seja, $opt(G,k) = |\cap(L')|$. Sabemos que $|\cap(L')| \leq d_G(v_i)$ para todo $v_i \in L'$. Além disso, sabemos que só podem existir no máximo $k-1$ vértices de grau estritamente maior do que $d_G(v_k)$. Desta forma, como $|L'| = k$, pelo princípio da casa dos pombos, existe pelo menos um vértice $v_j \in L'$, tal que $d_G(v_j) \leq d_G(v_k)$. Portanto, como $opt(G,k) = |\cap(L')| \leq d_G(v_j) \leq d_G(v_k) = UB$, temos que $opt(G,k) \leq UB$. □

Proposição 4. *Seja um grafo bipartido $G = (L \cup R, E)$ e um inteiro k. Considere $L' \subset L$, com $0 < |L'| < k$, uma solução ótima parcial. Para todo $u \in L \setminus L'$, se $\cap(L') = \cap(L' \cup \{u\})$, então $L' \cup \{u\}$ é uma solução ótima parcial.*

Demonstração. Seja L^* uma solução ótima, tal que, $u \notin L^*$. Sem perda de generalidade, suponha que $L' \subset L^*$. Como $\cap(L') = \cap(L' \cup \{u\})$, então $|\cap(L^*)| = |\cap(L^* \cup \{u\})|$. Seja $v \in L^* \setminus L'$ e $L'' = (L^* \cup \{u\}) \setminus \{v\}$, então $|\cap(L^*)| \leq |\cap(L'')|$. Como L^* é ótimo, então $|\cap(L^*)| \geq |\cap(L'')|$, portanto $|\cap(L^*)| = |\cap(L'')|$ e L'' é ótimo. □

Dado um grafo bipartido $G = (L \cup R, E)$ e um inteiro positivo k, um subproblema do Problema da Máxima Interseção de k-Subconjuntos é definido por uma tupla (S, P, λ), onde $S \subseteq L$ é a solução parcial que está sendo construída, $P \subseteq L$ é o conjunto de vértices candidatos, pois é formado pelos vértices que podem entrar em S, e λ é a cardinalidade da melhor solução encontrada até o momento. No Algoritmo 12, é exibido o algoritmo de *Branch and Bound* proposto para o problema kMIS. Na chamada inicial do algoritmo BBKMIS, consideramos $S = \emptyset$, $P = L$ e λ com o valor de uma solução inicial. No algoritmo proposto, a solução inicial é obtida pela heurística gulosa kINTER ESTENDIDO, apresentada na Subseção 4.1.1.

Algoritmo 12: BBKMIS

Entrada: S, P, λ
Se $|S| \geq k$ **então**
  Retorna $S' \subseteq S$ *tal que* $|S'| = k$
$LS \leftarrow LimiteSuperior(S, P)$
Se $|S| + |P| < k$ *ou* $LS \leq \lambda$ **então**
  Retorna $\emptyset$
$S_I \leftarrow LimiteInferior(S, P)$
Se $|\cap(S_I)| = LS$ **então**
  Retorna S_I
$u \leftarrow CriterioRamificacao(S, P)$
$P \leftarrow P \setminus \{u\}$
$P' \leftarrow \{v \in P : |\cap(S) \cap N_G(u) \cap N_G(v)| > \lambda\}$
$S' \leftarrow S \cup \{u\}$
$D \leftarrow \{v \in P : \cap(S') = \cap(S' \cup \{v\})\}$
$S_e \leftarrow BBkMIS(S' \cup D, P' \setminus D, \lambda)$
$\lambda \leftarrow \max(\lambda, |\cap(S_e)|)$
$S_d \leftarrow BBkMIS(S, P \setminus \{i\}, \lambda)$
Se $|\cap(S_e)| > |\cap(S_d)|$ **então**
  Retorna S_e
Retorna S_d

Fonte: Próprio autor.

Consideramos que um subproblema pode ser podado quando temos a garantia que a solução em construção não pode gerar uma solução melhor do que a melhor solução já encontrada. Um subproblema (S,P,λ), pode ser podado por limitante, quando um limite superior para qualquer solução viável S', onde $S \subseteq S'$, $S' \subseteq S \cup P$ e $|S'| = k$, não supere λ. Na Linha 5, a poda por limitante é realizada. No algoritmo proposto, o procedimento de limite superior é baseado na Proposição 3. Também na Linha 5, é aplicado uma poda por inviabilidade, pois se $|S| + |P| < k$, não é possível construir uma solução viável com k vértices.

Um subproblema (S,P,λ), pode ser podado por otimalidade, quando encontramos uma solução viável S' com $S \subseteq S'$, $S' \subseteq S \cup P$, tal que, $|\cap(S')|$ seja igual ao limite dado pelo procedimento de limite superior. Na Linha 8, a poda por otimalidade é realizada. O procedimento para encontrar uma solução viável é uma adaptação da heurística KINTER, descrita na Subseção 3.1.1. Nesta adaptação, a solução construída S' é inicializada com S. Em seguida, a solução é estendida de maneira gulosa escolhendo o vértice $v \in P \setminus S'$, tal que, $|\cup(S' \cup \{v\})|$ seja máximo. Repetimos o procedimento até que $|S'| = k$.

Cada subproblema não podado (S,P,λ), é subdividido em dois novos subproblemas, de acordo com o vértice u escolhido a partir de P. Na Linha 10, o processo de seleção do vértice de P é realizado. No BBKMIS, o vértice u é selecionado, tal que, $|\cap(S \cup \{u\})| \leq |\cap(S \cup \{v\})|$, para todo $v \in P$, ou seja, é selecionado o vértice com a menor cardinalidade da interseção de sua vizinhança com a interseção da vizinhança dos vértices já escolhidos.

Quando um vértice $u \in P$ é selecionado para a ramificação, podemos remover todos os vértices $v \in P$, onde $|\cap(S) \cap N_G(u) \cap N_G(v)| \leq \lambda$, pois os vértices u e v não podem estar presentes ao mesmo tempo em uma solução L', tal que $S \subset L'$ e $|\cap(L')| > \lambda$. Na Linha 12, é construído o novo conjunto de vértices candidatos P', removendo os vértices v, tal que $|\cap(S) \cap N_G(u) \cap N_G(v)| \leq \lambda$.

Na Linha 14, quando um vértice $u \in P$ é adicionado na solução atual, podemos descobrir outros vértices de P que podem ser selecionados de maneira ótima, conforme a Proposição 4. O conjunto D, representa este conjunto de vértices selecionados.

Neste trabalho, adaptamos o teste de redução proposto por (BOGUE *et al.*, 2014a), para encontrar os vértices que podem ser removidos de G. Fazendo uma análise mais profunda, propomos um segundo teste de redução, com um poder de redução ainda mais efetivo, conforme descrito na Seção 5.1. Esses procedimentos de redução, podem ser empregados tanto para a redução do problema inicial, quanto para redução de cada subproblema na árvore de *Branch and*

Bound.

Dado o grafo de entrada $G(L \cup R, E)$, um inteiro positivo k e um subproblema (S, P, λ), podemos aplicar os testes de redução na instância G' e $k' = k - |S|$, onde $V(G') = L' \cup R'$, $L' = P$, $R' = \cap(S)$ e $N_{G'}(u) = N_G(u) \cap R'$. Caso $u \in V(G')$ for removido por alguns dos testes de redução, então podemos remover u do conjunto de candidatos P. No capítulo 6, são apresentadas quatro versões do algoritmo *Branch and Bound* proposto, variando a utilização dos procedimentos de redução nos subproblemas da árvore de enumeração.

5.3 Programação Linear Inteira

Neste trabalho, foi desenvolvida uma nova formulação para o Problema da Máxima Interseção de k-Subconjuntos. A formulação proposta aqui, denominada *M_NEIGHB*, é uma variação da formulação *M_ARESTA*, descrita em 3.1.2.3.

Esta formulação se utiliza das variáveis binárias x_i e y_j. A variável x_i, indica se um vértice $i \in R$ foi ou não escolhido para a solução. Já variável y_j, indica se um vértice $j \in L$ foi ou não escolhido para a solução. Dada a definição das variáveis do modelo, a formulação é apresentada na Equação 5.1.

$$
\begin{aligned}
\max \quad & \sum_{i \in R} x_i \\
\text{sujeito a} \quad & \sum_{j \in L} y_j = k \\
& \sum_{i \in \overline{N_G(j)}} x_i \leq (1 - y_j) \cdot |\overline{N_G(j)}| \quad \forall j \in L \\
& \sum_{j \in \overline{N_G(i)}} y_j \leq (1 - x_i) \cdot |\overline{N_G(i)}| \quad \forall i \in R \\
& x_i \in \{0,1\} \quad \forall i \in R \\
& y_j \in \{0,1\} \quad \forall j \in L
\end{aligned}
\tag{5.1}
$$

Assim como as demais formulações para o Problema da Máxima Interseção de k-Subconjuntos, buscamos maximizar $\sum_{i \in R} x_i$, ou seja, o número de vértices do conjunto R na solução. A primeira restrição garante que exatamente k vértices do conjunto L estarão na solução. A segunda restrição garante que se $j \in L$ estiver na solução, então $x_i = 0$ para todo $i \in \overline{N_G(j)}$. A terceira restrição garante que se $i \in R$ estiver na solução, então $y_j = 0$ para todo $j \in \overline{N_G(i)}$. As

duas últimas restrições garantem o domínio das variáveis utilizadas no modelo. Esta formulação foi elaborada com o objetivo de reduzir o número de restrições da formulação M_ARESTA.

6 RESULTADOS COMPUTACIONAIS

Neste capítulo, são apresentados os resultados computacionais obtidos. Na seção 6.1, é descrita a geração das instâncias utilizadas neste trabalho. Na seção 6.2, é descrita a comparação entre os testes de redução propostos neste trabalho com os já existentes na literatura. Nas seções 6.3 e 6.4, são descritos os testes computacionais realizados, respectivamente, com os algoritmos heurísticos e exatos propostos neste trabalho.

Todos os testes computacionais descritos neste capítulo, foram realizados em um computador Intel Xeon, com 2.30 GHz, 64 GBytes de memória RAM e usando o CentOS 7.0. Todos os algoritmos foram executados com 1 thread, com limite de memória de 8 GBytes e limitação de tempo execução de 1800 segundos. Os algoritmos comparados neste capítulo foram implementados utilizando a linguagem C++. Além disso, as formulações comparadas nesta seção foram executadas utilizando o solver matemático CPLEX na versão 12.7.

Em todos os algoritmos implementados, as vizinhanças de vértices são representadas por meio de strings de bits, desta forma, todas as operações de interseção em nossos algoritmos são feitas por meio de paralelismo de bits. O paralelismo de bits já foi empregado com sucesso em diversos problemas da literatura: clique máxima (SEGUNDO *et al.*, 2011), clique máxima ponderada (TAVARES *et al.*, 2015; TAVARES *et al.*, 2016) e k-plex máximo (SILVA *et al.*, 2017).

6.1 Instâncias de Teste Aleatórias

Neste trabalho, foram geradas instâncias de grafos de maneira aleatória, usando o modelo de grafos descrito em Gilbert (1959). Estas instâncias foram baseadas no trabalho de Bogue *et al.* (2013), onde os grafos diferenciam-se pela densidade e valor de k, conforme Tabela 2. Para instâncias de mesma densidade e tamanho, foram geradas um só grafo e foi variando o parâmetro k, de acordo com a classe da instância. Acredita-se que desta forma pode-se avaliar melhor o comportamento dos algoritmos em relação a essas duas propriedades.

Geramos 9 classes de instâncias de acordo com a Tabela 3. Para cada classe, geramos instâncias de tamanho 40, 60, 80, 100, 140, 180, 200, 240, 280 e 300 vértices. Em relação ao balanceamento do grafo, geramos três grupos de instâncias. O primeiro grupo (G_1) são os grafos bipartidos balanceados, ou seja, $|L| = |R|$, o segundo grupo (G_2) os grafos com $|L| > |R|$ e o terceiro (G_3) com $|L| < |R|$. Para as instâncias desbalanceadas, usamos a proporção de 80% do

tamanho da menor em relação a maior.

Tabela 2: Classificação das propriedades das instâncias.

Classificação	Probabilidade Aresta	k
Baixo	0.3	$[0.1\|L\|, 0.3\|L\|]$
Média	0.6	$[0.4\|L\|, 0.6\|L\|]$
Alta	0.9	$[0.7\|L\|, 0.9\|L\|]$

Fonte: Próprio autor.

Tabela 3: Classes de Instâncias.

k \ Densi.	Baixa	Média	Alta
Baixo	C_1	C_4	C_7
Médio	C_2	C_5	C_8
Alto	C_3	C_6	C_9

Fonte: Próprio autor.

Após testes computacionais preliminares, o valor de k utilizado foi dado pelo limitante inferior de cada classificação. Assim como no trabalho de Bogue *et al.* (2013), não foi possível gerar instâncias viáveis para a classe C_3. Uma instância é dita viável para o Problema da Máxima Interseção de k-Subconjuntos quando dado a solução ótima L^*, temos que $|\cap(L^*)| \geq 1$. Além disso, não foi possível gerar algumas instâncias viáveis das classes C_2 e C_6. Das 270 instâncias geradas, somente 216 instâncias são viáveis e foram utilizadas em nossos testes.

6.2 Pré-processamento

No trabalho de Bogue *et al.* (2014a) foi proposto um conjunto de testes de redução como procedimento de pré-processamento das instâncias, visando a diminuição da dimensão do problema. Neste trabalho, foi proposto um novo conjunto de testes de redução. Foram realizados testes computacionais visando comparar a qualidade dos testes de redução.

Nesta seção, os testes de redução propostos em Bogue *et al.* (2014a), serão denominados por *TR_LR*, e os propostos neste trabalho, serão denominados *TR_Strong*. Tendo como entrada as instâncias aleatórias geradas, executamos os testes de redução e analisamos o número de vértices removidos.

Nas classes C_2 e C_6, ambos os testes de redução removeram todos os vértices do grafo. Já nas classes C_8 e C_9 ambos os testes não removeram nenhum vértice do grafo. Removemos da comparação tais instâncias para facilitar a visualização e a comparação dos testes de redução. Na Tabela 4, é apresentado o número de vértices definidos por classe e por grupo de instância.

Analisando a Tabela 4, pode-se notar que os testes de redução propostos conseguem definir mais vértices que os testes de redução existentes na literatura. Vale destacar que nas classes C_1 e C_5 o *TR_Strong* consegue definir um número muito maior de vértices que o *TR_LR*. Já na classe C_4 e C_7 o *TR_Strong* define um número pouco maior de vértices. No geral, o *TR_Strong* definiu 26.71% mais vértices que o *TR_LR*.

Tabela 4: Resultados dos testes de redução para as classes C_1, C_4, C_5 e C_7.

Classe	*TR_LR*				*TR_Strong*			
	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total
C_1	18	76	25	119	44	87	87	**218**
C_4	5	10	16	31	5	13	26	**44**
C_5	199	214	173	586	260	273	230	**763**
C_7	96	94	182	372	100	97	182	**379**
Total	318	394	396	1108	**409**	**470**	**525**	**1404**

Fonte: Próprio autor.

Em relação ao tempo de execução, em média o *TR_LR* executou em 7.26 ms e o *TR_Strong* executou em 204.02 ms. Apesar da grande diferença de tempo, acredita-se que o tempo de execução do *TR_Strong* ainda seja baixo comparado ao tempo 1800 segundos fornecidos a abordagens exatas do problema na literatura. Desta forma, os testes de redução propostos foram selecionados e utilizados como procedimento de pré-processamento.

6.3 Testes Computacionais com Algoritmos Heurísticos

Nesta seção, são descritos os resultados computacionais dos testes realizados com algoritmos meta-heurísticos. Até então, o GRASP REATIVO era o melhor algoritmo para o Problema da Máxima Interseção de k-Subconjuntos existente na literatura. Portanto, os algoritmos VNS REATIVO e ACO KMIS, propostos neste trabalho, foram comparados com o algoritmo GRASP REATIVO.

O algoritmo GRASP REATIVO foi implementado conforme indicado no trabalho de Bogue *et al.* (2014a). Os parâmetros utilizados no algoritmo VNS REATIVO, foram obtidos de forma empírica. O parâmetro $seqs = 500$, o q_{max} igual a $|L| - k - 4$ se $k \geq |L|/2$, ou $q_{max} = k - 4$ se $k \leq |L|/2$ e $k + 4 \geq |L|/2$. Os parâmetros utilizados no algoritmo ACO KMIS foram obtidos também de forma empírica e são eles: $\alpha = 0.5$, $\beta = 2$, $\tau_0 = 1$, $\rho = 0.7$ e $IterMax = 50$. Vale destacar que, pelo fato dos algoritmos GRASP REATIVO e VNS REATIVO possuírem componentes de aleatoriedade, estes algoritmos foram executados 10 vezes por instância, e a solução e o tempo de execução considerados, foram obtidos através da média das 10 execuções.

Com o intuito de comparar os algoritmos, utilizamos a métrica GAP, descrita na Equação 6.1, onde UB é o tamanho da melhor solução daquela instância entre os algoritmos comparados e BUB é o tamanho da solução do algoritmo avaliado. Note que quanto menor o

GAP, maior é a qualidade da solução retornada pelo algoritmo analisado.

$$GAP = \frac{UB - BUB}{UB} \tag{6.1}$$

Na Tabela 5, é apresentada a comparação entre os algoritmos por grupo de instância. A coluna "Qtd" significa a quantidade de soluções onde o algoritmo obteve a melhor solução entre os algoritmos comparados e a coluna $\bar{x}$ mostra a média dos GAPs obtidos naquela classe de instância.

Tabela 5: Resultados dos testes computacionais com algoritmos Meta-heurísticos por classe de instância.

Classe	GRASP REATIVO			VNS REATIVO			ACO KMIS	
	Qtd	$\bar{x}$		Qtd	$\bar{x}$		Qtd	$\bar{x}$
C_1	21	0.10111		26	0.02638		**30**	**0.00000**
C_2	**17**	**0.00000**		**17**	**0.00000**		**17**	**0.00000**
C_4	11	0.06122		**24**	0.01260		**24**	**0.00827**
C_5	22	0.07000		28	0.01444		**29**	**0.00833**
C_6	**19**	**0.00000**		**19**	**0.00000**		**19**	**0.00000**
C_7	16	0.00873		25	**0.00132**		**26**	0.00149
C_8	6	0.05879		17	0.03270		**27**	**0.00325**
C_9	10	0.06256		18	0.05634		**30**	**0.00000**
Total	122	0.05033		174	0.01997		**202**	**0.00296**

Fonte: Próprio autor.

Analisando a Tabela 5, pode-se notar que os algoritmos conseguiram soluções de mesma qualidade nas classes C_2 e C_6. Na classe C_7, o algoritmo VNS REATIVO apresentou o melhor GAP médio, porém o ACO KMIS foi superior em relação a quantidade de instâncias onde o algoritmo encontrou a melhor solução. Nas demais classes, o algoritmo ACO KMIS se mostrou superior aos demais. Vale destacar, que nas classes C_8 e C_9, o ACO KMIS conseguiu encontrar as melhores soluções em uma grande quantidade instâncias, principalmente quando comparado aos demais algoritmos.

Os algoritmos propostos VNS REATIVO e ACO KMIS, se mostraram superiores em qualidade de solução em relação ao algoritmo do estado da arte GRASP REATIVO. O algoritmo ACO KMIS se mostrou superior em qualidade de solução, e conseguiu encontrar a melhor solução em cerca de 93.51% das instâncias de teste. Na Tabela 6, é apresentado o tempo médio de execução, em segundos, dos algoritmos meta-heurísticos comparados por grupo de instância.

Tabela 6: Média dos Tempos de execução dos algoritmos Meta-heurísticos por grupo de instância.

Grupo	GRASP REATIVO	VNS REATIVO	ACO KMIS
G_1	24.831	**10.253**	36.570
G_2	22.691	**10.871**	35.230
G_3	11.566	**6.320**	18.360
Total	19.644	**9.127**	29.976

Fonte: Próprio autor.

Observando a Tabela 6, pode-se verificar que o algoritmo VNS REATIVO foi o que obteve o menor tempo de execução médio em relação aos algoritmos analisados em todos os grupos de instâncias. Já o algoritmo ACO KMIS, foi o que obteve sempre o maior tempo de execução médio. Desta forma, o algoritmo VNS REATIVO se mostrou o mais eficiente em tempo de execução.

6.4 Testes Computacionais com Algoritmos Exatos

Nesta seção, são apresentados os principais resultados obtidos em testes computacionais realizados com algoritmos exatos para o Problema da Máxima Interseção de k-Subconjuntos.

As instâncias das classes C_2 e C_4 não serão desconsideradas nos testes computacionais desta seção, pois os testes de redução já conseguem definir todos os vértices destas classes. Desta forma, das 216 instâncias geradas, somente 180 instâncias serão utilizadas nos testes computacionais descritos nesta seção. Vale destacar que todos os algoritmos exatos tiveram uma limitação de tempo de execução de 1800 segundos.

Inicialmente foram realizados testes em relação ao algoritmo *Branch and Bound* proposto. Foram geradas quatro versões do Algoritmo 12, baseado na utilização de um dos testes de redução e no momento em que são aplicados. O BBKMIS_TRLR e BBKMIS_TRS aplicam o teste *TR_LR* e *TR_Strong*, respectivamente, em todas as chamadas do algoritmo. Como a inserção de um vértice na solução parcial S tem mais chances de causar mudança no grafo, e consequentemente, os testes de redução terão mais chances de remover vértices, foram criadas duas versões, onde os testes de redução são aplicados apenas quando ocorre a inserção de um novo vértice em S, BBKMIS_TRLRE e o BBKMIS_TRSE.

A fim de verificar quais das versões obtiveram os melhores resultados, foi implementada e executada as quatro versões do algoritmo. A Tabela 7, apresenta a quantidade de instâncias que cada algoritmo conseguiu determinar a solução ótima.

Tabela 7: Comparação entre as quatro versões do Algoritmo BBKMIS.

Classe	BBKMIS_TRLR				BBKMIS_TRS				BBKMIS_TRLRE				BBKMIS_TRSE			
	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total
C_1	10	10	10	**30**	10	10	10	**30**	10	10	10	**30**	10	10	10	**30**
C_4	4	4	5	13	4	4	4	12	4	5	5	**14**	4	4	5	13
C_5	10	10	10	**30**	6	8	6	20	8	10	8	26	6	8	6	20
C_7	4	4	4	**12**	3	4	4	11	4	4	4	**12**	4	4	4	**12**
C_8	1	2	2	**5**	1	1	1	3	1	2	2	**5**	1	2	1	**5**
C_9	2	2	2	**6**	2	2	2	**6**	2	2	2	**6**	2	2	2	**6**
Total	**31**	32	**33**	**96**	26	29	27	82	29	**33**	31	93	27	30	28	85

Fonte: Próprio autor.

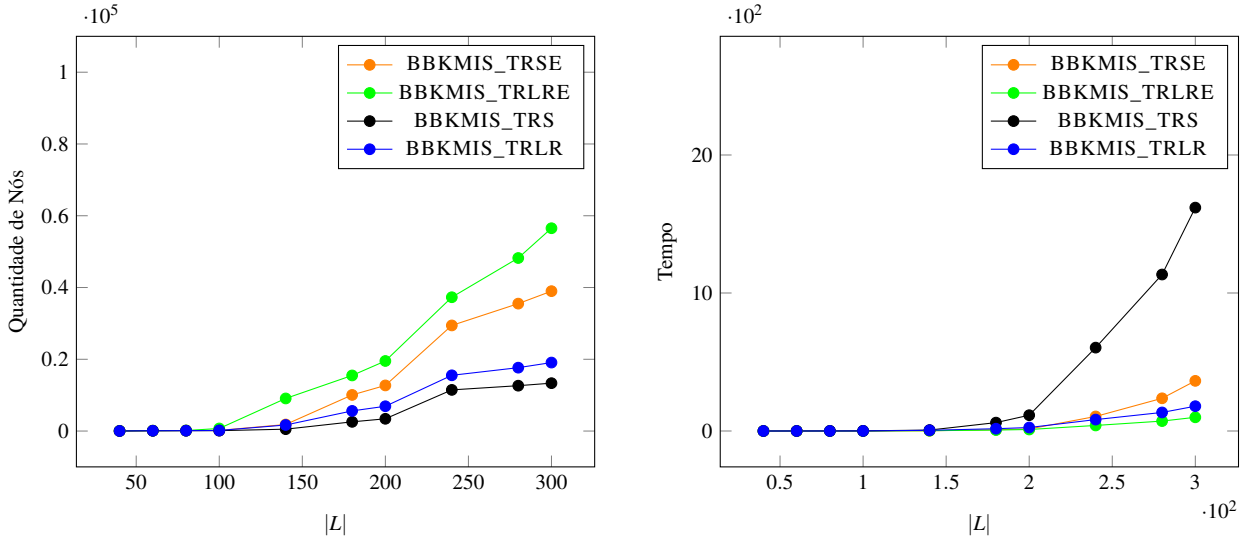
Das quatros versões do *Branch and Bound*, o BBKMIS_TRLR foi a que obteve melhores resultados gerais, destacando-se principalmente na classe C_5. A versão BBKMIS_TRLRE, obteve bons resultados e foi superior na classe C_4. Acredita-se que as versões BBKMIS_TRLR e BBKMIS_TRLRE se mostram superiores às versões BBKMIS_TRS e BBKMIS_TRSE devido ao fato do teste de redução *TR_Strong*, apesar de ser mais eficiente, é mais custoso em tempo de execução.

Desta forma, as versões BBKMIS_TRS e BBKMIS_TRSE exploram uma menor quantidade de nós, mas o tempo de execução fica mais custoso que as demais versões. Com a finalidade de analisar esta propriedade, foram gerados os gráficos apresentados nas Figuras 7a e 7b. Estes gráficos foram gerados a partir da análise de tempo e quantidade de nós gerados pelas quatro versões na classe C_1 do grupo G_1, em relação ao tamanho do conjunto de vértices L. Esta classe foi escolhida pois todas versões conseguiram garantir a solução ótima em todas as instâncias desta classe em um tempo de execução relativamente não baixo.

Podemos notar que existe uma grande disparidade de nós gerados, quando comparamos as versões que aplicam os testes de redução em todos os nós com as versões que realizam esses testes somente quando um novo vértice é inserido na solução parcial. Observe que apesar da redução da quantidade de nós gerados na versão BBKMIS_TRS ser inferior às demais versões, o seu tempo de execução é bem superior. Note que isso se deve ao fato do teste de redução *TR_Strong* ser bastante custoso. De maneira geral, a versão BBKMIS_TRLR foi a que apresentou os melhores resultados e será considerada nas próximas comparações.

Neste trabalho, propomos um algoritmo *Branch and Bound*, denominado BBKMIS_TRLR, e uma formulação de Programação Linear Inteira, denominada M_NEIGHB. Foi realizado um conjunto de testes computacionais visando comparar as abordagens propostas com a formulação M_CLIQUE, que é o método exato mais eficiente existente na literatura até então. Os algoritmos foram executados com as instâncias de teste com o objetivo de comparar a qualidade dos métodos exatos em relação a quantidade de instâncias onde foi possível encontrar

Figura 7: Comparação entre as versões *Branch and Bound* propostas em relação ao número de nós e ao tempo de execução.



(a) Número de nós gerados nas instâncias da classe C_1 do grupo G_1.

(b) Tempo de execução nas instâncias da classe C_1 do grupo G_1.

Fonte: Próprio autor.

a solução ótima e o tempo de execução. A Tabela 8, apresenta a quantidade de instâncias, por classe e por grupo, que os algoritmos comparados conseguiram encontrar e garantir a solução ótima.

Tabela 8: Número de instâncias ótimas encontradas pelos algoritmos exatos.

Classe	M_CLIQUE				BBKMIS_TRLR				M_NEIGHB			
	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total
C_1	6	6	6	18	10	10	10	**30**	8	10	9	27
C_4	3	3	4	10	4	4	5	**13**	2	3	3	8
C_5	6	6	7	19	10	10	10	**30**	8	9	8	25
C_7	4	4	4	**12**	4	4	4	**12**	4	4	4	**12**
C_8	3	4	4	**11**	1	2	2	5	2	2	2	6
C_9	4	4	4	**12**	2	2	2	6	4	4	4	**12**
Total	26	27	29	82	**31**	**32**	**33**	**96**	28	**32**	30	90

Fonte: Próprio autor.

Podemos observar na Tabela 8 que o algoritmo BBKMIS_TRLR consegue encontrar a solução ótima em um maior número de instâncias (53,33 %), comparado aos demais métodos exatos. Este algoritmo se mostra mais eficiente em instâncias de densidade baixa e média (classes C_1, C_4 e C_5), destacando-se principalmente nas classes C_1 e C_5. Já a formulação M_CLIQUE, se mostra mais eficientes em classes com a densidade alta e k médio e alto (classes C_8 e C_9), destacando-se na classe C_8. A formulação M_NEIGHB, se mostra mais eficiente que a

formulação M_CLIQUE e menos eficiente que o algoritmo BBKMIS_TRLR, nas classes C_1, C_4 e C_5. Já nas classes C_8 e C_9, ocorre o contrário. Na classe C_7 (densidade alta e k baixo), todos os métodos exatos encontraram a mesma quantidade de soluções ótimas.

Visando analisar melhor o comportamento dos métodos exatos, foi realizado uma comparação de tempo de execução nas instâncias, onde os três métodos exatos conseguiram encontrar e garantir a solução ótima. Neste nova comparação, apenas 67 instâncias serão consideradas. A Tabela 9, apresenta o tempo de execução médio, em segundos, dos algoritmos por classe instância.

Tabela 9: Média de tempo por classe.

Classe	M_CLIQUE	BBKMIS_TRLR	M_NEIGHB
C_1	236.6105	**3.0250**	47.12611
C_4	69.72875	**0.42250**	296.22625
C_5	212.69944	33.86111	**31.55388**
C_7	34.12500	**32.94166**	159.05416
C_8	**0.14200**	279.94200	51.26400
C_9	**0.08000**	64.49833	2.27666

Fonte: Próprio autor.

Analisando os resultados apresentados na Tabela 9, pode-se notar que o tempo de execução médio do algoritmo BBKMIS_TRLR é menor nas classes C_1, C_4 e C_7. A formulação M_CLIQUE, possui tempo médio de execução menor nas classes C_8 e C_9. Já a formulação M_NEIGHB, possui menor tempo na classe C_5. De maneira geral, o algoritmo BBKMIS_TRLR possui menor tempo de execução nas classes de densidade baixa e média, e a formulação M_CLIQUE possui menores tempos de execução em classes de densidade alta.

Comparando os resultados obtidos e apresentados nas Tabelas 8 e 9, pode-se notar que, em termos gerais, o algoritmo BBKMIS_TRLR se mostra mais eficiente que os demais métodos exatos nas classes de densidade baixa e média. Além disso, a formulação M_CLIQUE se mostra superior nas classes de densidade alta.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi proposto um conjunto de abordagens, exatas e heurísticas, para o Problema da Máxima Interseção de k-Subconjuntos. Em relação às abordagens heurísticas, foram propostas duas novas heurísticas gulosas e dois algoritmos meta-heurísticos. Já em relação às abordagens exatas, foram propostos um algoritmo *Branch and Bound* e uma formulação de Programação Linear Inteira. Além disso, foi proposto um conjunto de testes de redução, que foi utilizado como pré-processamento das instâncias de teste.

Inicialmente, foram geradas instâncias de teste aleatórias, com base na densidade do grafo e no parâmetro k. Após isso, foram realizados experimentos computacionais com os algoritmos meta-heurísticos propostos. Estes experimentos evidenciaram que os algoritmos propostos, VNS REATIVO e ACO KMIS, superam o algoritmo do estado da arte GRASP REATIVO, em relação à qualidade das soluções retornadas. Vale destacar, que o algoritmo ACO KMIS se mostrou mais eficiente em qualidade de solução que os demais algoritmos. Em relação ao tempo de execução, apenas o algoritmo VNS REATIVO se mostrou mais eficiente que o GRASP REATIVO.

Também foram realizados testes computacionais com os métodos exatos propostos. Em termos gerais, os métodos exatos propostos, BBKMIS_TRLR e M_NEIGHB, se mostraram mais eficientes que o método exato do estado arte M_CLIQUE. Nas classes de densidade baixa e média, o algoritmo BBKMIS_TRLR conseguiu encontrar mais soluções ótimas, enquanto que nas classes de densidade alta, a formulação M_CLIQUE se mostrou mais eficiente.

Vale destacar que alguns resultados apresentados neste trabalho já foram publicados no Simpósio Brasileiro de Pesquisa Operacional (COSTA *et al.*, 2018; COSTA *et al.*, 2020) e no Journal of Heuristics (DIAS *et al.*, 2020).

Como trabalhos futuros, listamos as seguintes possibilidades:

- Propor testes de redução que visem a inserção de vértices;
- Aplicar o procedimento de Critério de Parada, do VNS REATIVO, no algoritmo ACO KMIS, visando diminuir seu tempo de execução;
- Propor uma versão do *Branch and Bound* inversa, onde a cada nó iremos decidir se devemos ou não remover um vértice u da solução.

REFERÊNCIAS

ADI, S. S.; FERREIRA, C. E. **Identificação de genes por comparação de sequências**. Dissertação (Mestrado) – Universidade de São Paulo, 2005.

BOGUE, E. T.; SOUZA, C. C. de; XAVIER, E. C.; FREIRE, A. S. O problema da máxima interseção de k-subconjuntos. In: **Anais do XLV SBPO**. Natal: Sobrapo, 2013. p. 2416–2425.

BOGUE, E. T.; SOUZA, C. C. de; XAVIER, E. C.; FREIRE, A. S. An integer programming formulation for the maximum k-subset intersection problem. In: **Combinatorial Optimization**. Cham: Springer International Publishing, 2014. p. 87–99.

BOGUE, E. T.; SOUZA, C. C. de; XAVIER, E. C.; FREIRE, A. S. **O problema da Máxima Interseção de k-Subconjuntos**. Dissertação (Mestrado) – Universidade Estadual de Campinas, 2014.

CLAUSEN, J. Branch and bound algorithms-principles and examples. **Department of Computer Science, University of Copenhagen**, [*S. l.*: *s. n.*], p. 1–30, 1999.

COSTA, F. N.; URRUTIA, S. A. **Programação de tabelas para torneios round robin simples com estádios predefinidos**. Dissertação (Mestrado) – Universidade Federal de Minas Gerais, 2009.

COSTA, J. R. d. F.; DIAS, F. C. S.; TAVARES, W. de A. Um algoritmo vns para o problema da máxima interseção de k-subconjuntos. In: **Anais do L SBPO**. Rio de Janeiro, RJ: Sobrapo, 2018.

COSTA, J. R. d. F.; DIAS, F. C. S.; TAVARES, W. de A. Um algoritmo branch-and-bound para o problema da máxima interseção de k-subconjuntos. In: **Anais do LII SBPO**. João Pessoa, PB: Sobrapo, 2020.

CRUZ C. D. A.; NETO, R. M. F. M. C. S. H. P. H. D. B. Heurística lagrangiana para o problema de alocação de veículos. **Anais do LI SBPO**, Limeira: Sobrapo, Sobrapo, v. 2, 2019.

DAWANDE, M.; KESKINOCAK, P.; SWAMINATHAN, J. M.; TAYUR, S. On bipartite and multipartite clique problems. **Journal of Algorithms**, [*S. l.*]: Elsevier, Elsevier, v. 41, n. 2, p. 388–403, 2001.

DIAS, F. C.; TAVARES, W. A.; COSTA, J. R. d. F. Reactive vns algorithm for the maximum k-subset intersection problem. **Journal of Heuristics**, [*S. l.*]: Springer, Springer, v. 26, n. 6, p. 913–941, 2020.

DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. **IEEE computational intelligence magazine**, [*S. l.*]: IEEE, IEEE, v. 1, n. 4, p. 28–39, 2006.

DORIGO, M.; CARO, G. D. Ant colony optimization: a new meta-heuristic. In: IEEE. **Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 congress on**. [*S. l.*], 1999. v. 2, p. 1470–1477.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. **Operations research letters**, [*S. l.*]: Elsevier, Elsevier, v. 8, n. 2, p. 67–71, 1989.

GAREY, M. R.; JOHNSON, D. S. **Computers and intractability**. New York, NY: W. H. Freeman and Company, 1979.

GILBERT, E. N. Random graphs. **The Annals of Mathematical Statistics**, [*S. l.*]: The Institute of Mathematical Statistics, The Institute of Mathematical Statistics, v. 30, n. 4, p. 1141–1144, 12 1959.

GLOVER, F. Tabu search and adaptive memory programming - advances, applications and challenges. In: **Interfaces in computer science and operations research**. Boston, MA: Springer, 1997. p. 1–75.

GOLDBARG, E.; GOLDBARG, M.; LUNA, H. **Otimização Combinatória e Metaheurísticas**: Algoritmos e apliacações. Rio de Janeiro, RJ: Elsevier, 2017.

GOLDBARG, M.; LUNA, H. P.; GOLDBARG, E. **Programação linear e fluxos em redes**. Rio de Janeiro, RJ: Elsevier, 2016.

HANSEN, P.; JAUMARD, B.; MLADENOVIC, N.; PEREIRA, A. Variable neighborhood search for weight satisfiability problem. **Les Cahiers du GERARD - G-2000-62**, [*S. l.*: *s. n.*], 2000.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. Porto Alegre, RS: AMGH Editora Ltda, 2013.

KARMARKAR, N. A new polynomial-time algorithm for linear programming. In: **Proceedings of the sixteenth annual ACM symposium on Theory of computing**. [*S. l.*: *s. n.*], 1984. p. 302–311.

MANGASARIAN, O. L. Privacy-preserving linear programming. **Optimization Letters**, [*S. l.*]: Springer, Springer, v. 5, n. 1, p. 165–172, 2011.

MCCREESH, C.; PROSSER, P. An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In: SPRINGER. **International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems**. [*S. l.*], 2014. p. 226–234.

MIYAZAWA F. K; DE SOUZA, C. C. Introdução à otimização combinatória. In: **Jornadas de Atualizaçao em Informática-Congresso da Sociedade Brasileira de Computaçao-JAI-SBC**. [*S. l.*: *s. n.*], 2015.

MULATI, M. H.; CONSTANTINO, A. A.; SILVA, A. F. Otimização por colônia de formigas. In: OMINIPAX. **Meta-Heuristicas em Pesquisa Operacional**. Curitiba, PR, 2013. p. 53–68.

NASH, J. C. The (dantzig) simplex method for linear programming. **Computing in Science & Engineering**, [*S. l.*]: IEEE, IEEE, v. 2, n. 1, p. 29–31, 2000.

NEMHAUSER, G. L.; SIGISMONDI, G. A strong cutting plane/branch-and-bound algorithm for node packing. **Journal of the Operational Research Society**, [*S. l.*]: Taylor & Francis, Taylor & Francis, v. 43, n. 5, p. 443–457, 1992.

PRAIS, M.; RIBEIRO, C. C. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. **INFORMS Journal on Computing**, [*S. l.*]: INFORMS, INFORMS, v. 12, n. 3, p. 164–176, 2000.

RIBEIRO, C. C.; UCHOA, E.; WERNECK, R. F. A hybrid grasp with perturbations for the steiner problem in graphs. **INFORMS Journal on Computing**, [*S. l.*]: INFORMS, INFORMS, v. 14, n. 3, p. 228–246, 2002.

ROSSI, R. A.; GLEICH, D. F.; GEBREMEDHIN, A. H.; PATWARY, M. M. A. Fast maximum clique algorithms for large graphs. In: **Proceedings of the 23rd International Conference on World Wide Web**. [*S. l.*: *s. n.*], 2014. p. 365–366.

SEGUNDO, P. S.; RODRÍGUEZ-LOSADA, D.; JIMÉNEZ, A. An exact bit-parallel algorithm for the maximum clique problem. **Computers & Operations Research**, [*S. l.*]: Elsevier, Elsevier, v. 38, n. 2, p. 571–581, 2011.

SILVA, M. R. C. D.; TAVARES, W. A.; DIAS, F. C. S.; NETO, M. B. C. Algoritmo branch-and-bound para o problema do k-plex máximo. **Anais do XLIX SBPO**, Blumenau, SC: [*S. n.*], 2017.

SUCUPIRA, I. R. **Métodos heurísticos genéricos**: metaheurísticas e hiper-heurísticas. Dissertação (Mestrado) – Universidade de São Paulo, São Paulo, SP, 2004.

TAVARES, W. A.; NETO, M. B. C.; RODRIGUES, C. D.; MICHELON, P. Um algoritmo de branch and bound para o problema da clique máxima ponderada. **Anais do XLVII SBPO**, Porto de Galinhas, PE: [*S. n.*], v. 1, 2015.

TAVARES, W. A.; NETO, M. B. C.; RODRIGUES, C. D.; MICHELON, P. Bitclique um algoritmo de branch-and-bound para o problema da clique máxima ponderada. **Anais do XLVIII SBPO**, Vitória, ES: [*S. n.*], v. 1, 2016.

VINTERBO, S. A. A note on the hardness of the k-ambiguity problem. **Technical Report DSG-T R-2002-006**, [*S. l.*: *s. n.*], 2002.

WOLSEY, L. A. **Integer programming**. New York: Wiley Online Library, 1998. v. 42.

XAVIER, E. C. A note on a maximum k-subset intersection problem. **Information Processing Letters**, [*S. l.*]: Elsevier, Elsevier, v. 112, n. 12, p. 471–472, 2012.

ZHOU, Y.; ROSSI, A.; HAO, J.-K. Towards effective exact methods for the maximum balanced biclique problem in bipartite graphs. **European Journal of Operational Research**, [*S. l.*]: Elsevier, Elsevier, v. 269, n. 3, p. 834–843, 2018.