LII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2020

Um Algoritmo Branch-and-Bound para o Problema da Máxima Interseção de k-Subconjuntos

José Robertty de Freitas Costa
Campus Quixadá – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003, Cedro – 63902-580 – Quixadá – CE – Brazil
robertty@alu.ufc.br

Fábio Carlos Sousa Dias, Wladimir Araújo Tavares
Campus Quixadá – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003, Cedro – 63902-580 – Quixadá – CE – Brazil
{fabiocsd, wladimir}@lia.ufc.br

RESUMO

Neste artigo, apresentamos um algoritmo *Branch-and-Bound*, chamado BBKMIS, para o problema da Máxima Interseção de k-Subconjuntos (kMIS). Dado uma coleção $\mathcal{S}$ de subconjuntos do conjunto finito B e um inteiro k, deseja-se selecionar k subconjuntos $S_1, S_2, \ldots, S_k \in \mathcal{S}$, tal que $|S_1 \cap \ldots \cap S_k|$ seja máxima. Para este algoritmo, propomos um procedimento de limite superior, um critério de ramificação e analisamos dois critérios de redução, que podem ser empregados para redução do problema inicial e/ou de cada subproblemas no árvore de *Branch-and-Bound*. O primeiro procedimento de redução foi proposto por Bogue et al. [2014], e o segundo, proposto neste trabalho, estende o anterior. Nos testes computacionais realizados, o algoritmo proposto supera o algoritmo do estado da arte na maior parte das classes de instâncias avaliadas.

PALAVRAS CHAVE. Branch-and-Bound, Testes de Redução, Problema da Máxima Interseção de k-Subconjuntos.

Tópicos: Otimização Combinatória, Programação Matemática.

ABSTRACT

In this article, we present a *Branch-and-Bound* algorithm, called BBKMIS, for Maximum k-Subset Intersection Problem(kMIS) . Given a collection $\mathcal{S}$ of subsets of the finite set B and an integer k, we want to select k subsets $S_1, S_2, \ldots, S_k \in \mathcal{S}$ such that $|S_1 \cap \ldots \cap S_k|$ is maximum. For this algorithm, we propose an upper bound procedure, a branching strategy and we analyze two reduction procedures, which can be used to reduce the initial problem and/or each sub-problem in the *Branch-and-Bound* tree. The first reduction procedure was proposed by Bogue et al. [2014] and the second, proposed in this work, extends the previous one. In the computational tests performed, the proposed algorithm overcomes the state-of-the-art algorithm in most of the classes of instances evaluated.

KEYWORDS. Branch-and-Bound, Reduction Tests, k-SubSet Maximum Intersection Problem.

Topics: Combinatorial Optimization, Math Programming.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

1. Introdução

O Problema da Máxima Interseção de k-Subconjunto (kMIS), pode ser definido como: Dado uma coleção $\mathcal{S}$ de subconjuntos do conjunto finito B e um inteiro k, deseja-se selecionar k subconjuntos $S_1, S_2, \ldots, S_k \in \mathcal{S}$, tal que $|S_1 \cap \ldots \cap S_k|$ seja máxima. O problema pode ser modelado com um grafo bipartido $G = (L \cup R, E)$, sendo que cada vértice $v \in L$ está associado a um subconjunto $S \in \mathcal{S}$ e cada vértice $u \in R$ está associado a um elemento de B. Existe uma aresta entre um vértice i associado a um subconjunto $S_i \in \mathcal{S}$ e um vértice j relacionado a um $b_j \in B$, se e somente se, $b_j \in S_i$. Por exemplo, seja $B = \{1, 2, 3, 4, 5\}$, $\mathcal{S} = \{S_1, S_2, S_3\}$, onde $S_1 = \{2, 3, 4\}$, $S_2 = \{1, 4, 5\}$ e $S_3 = \{1, 2, 3, 4\}$. Para $k = 2$, nós temos que selecionar a máxima interseção entre dois subconjuntos. A Figura 1 ilustra este exemplo, temos que $|S_1 \cap S_2| = 1, |S_1 \cap S_3| = 3$ e $|S_2 \cap S_3| = 2$. Então a solução ótima é $L' = \{S_1, S_3\}$. Observe que o subgrafo induzido pelos vértices associados com S_1 e S_3 e todos os vértices associados com os elementos em $S_1 \cap S_3$ definem um grafo bipartido completo.

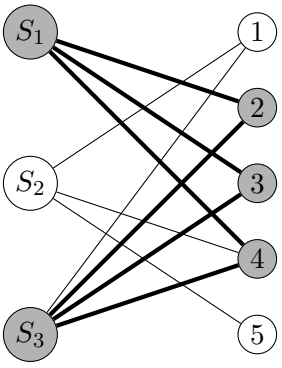


Figura 1: Para $k = 2$, a máxima interseção de k subconjuntos será obtido com os subconjuntos S_1 e S_3. O tamanho dessa interseção será 3.

O problema kMIS possui aplicações em privacidade de dados [Vinterbo, 2002], análise de dados em bioinformática [Nussbaum et al., 2012] e no contexto de redes sociais [Bogue et al., 2014]. No último trabalho citado, podemos encontrar a seguinte aplicação: Devemos encontrar k usuários de uma rede social que curtam um maior número de páginas em comum.

Em Vinterbo [2002] e Xavier [2012], o problema kMIS foi mostrado ser $\mathcal{NP}$-Difícil. Xavier [2012], provou que kMIS não pode ser aproximado dentro um fator constante $\frac{1}{n^{\varepsilon}}$, onde n é o número de subconjuntos e $\varepsilon > 0$, considerando que $NP \not\subset BPTIME(2^{n^{\varepsilon}})$. Shieh et al. [2012], mostrou que kMIS não pode ser aproximado dentro um erro absoluto $\frac{1}{2}n^{1-2\varepsilon} + O(n^{1-3\varepsilon})$ a menos que $P = NP$.

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 da formulação da Biclique Máxima em Arestas (MEB, do inglês Maximum Edge Biclique Problem) e uma nova formulação desenvolvida para o problema. Os autores mostraram que a nova formulação conseguiu resultados computacionais superiores as outras duas formulações adaptadas.

Em Bogue et al. [2014], os autores apresentam um procedimento de redução do tamanho da entrada do problema, uma meta-heurística do tipo GRASP para encontrar soluções mais próximas das ótimas e um modelo de programação linear inteira denominado M_CLIQUE. Os resultados computacionais mostraram que a formulação M_CLIQUE consegue resolver de maneira ótima mais instâncias do que a formulação M_EDGE.

Costa et al. [2018], apresentam duas novas heurísticas gulosas, obtidas a partir heurística gulosa kInter [Bogue et al., 2013], denominada *kInter Estendida* e *kInter Estendida com path-reliking*, e um algoritmo de busca em vizinhança variável reativo (VNS Reativo). Os resultados computacionais mostraram que o algoritmo VNS superou o algoritmo GRASP, apresentado em Bogue et al. [2014].

Neste trabalho, apresentamos um algoritmo *Branch-and-Bound* para o `kMIS`. Além disso, propomos novos testes de redução do tamanho da entrada. Os testes propostos se mostraram ser mais efetivos que as já existentes na literatura. Vale ressaltar que o algoritmo *Branch-and-Bound* também faz uso destes testes de redução como procedimento de fixação de variáveis. Na seção 2, apresentamos os conceitos iniciais e notações que serão utilizadas neste trabalho. Na seção 3, apresentamos os testes de redução existentes na literatura e os propostos por este trabalho. Já na seção 4, apresentamos o algoritmo *Branch-and-Bound* proposto. Na seção 5, apresentamos os testes computacionais realizados. Por fim, na seção 6, apresentamos as conclusões deste trabalho e os trabalhos futuros.

2. Preliminares

Ao longo deste artigo, vamos considerar que o Problema da Máxima Intersecção de k-subconjuntos (`kMIS`) está representado como um grafo bipartido $G = (L \cup R, E)$, sendo que cada vértice $v \in L$ está associado a um subconjunto $S \in \mathcal{S}$ e cada vértice $u \in R$ está associado a um elemento de B. Existe uma aresta entre um vértice i associado a um subconjunto $S_i \in \mathcal{S}$ e um vértice j relacionado a um $b_j \in B$, se e somente se, $b_j \in S_i$.

As seguintes notações serão usadas neste artigo. Seja G um grafo (L, R)-bipartido. Então,

- $N_G(v) = \{u \in L \cup R : uv \in E\}$ é a vizinhança do vértice v no grafo G.
- $d_G(v) = |N_G(v)|$ é o grau do vértice v no grafo G.
- Seja $L' \subseteq L$, denotamos $\cap(L') = \bigcap_{v \in L'} N(v)$. Vale ressaltar que se $L' = \emptyset$, então $\cap(L') = \emptyset$.
- $opt(G, k)$ é o valor máximo de $|\cap(L')|$ tal que $L' \subseteq L$ e $|L'| = k$.

Propomos neste trabalho um limite superior para o `kMIS`. Este limite pode ser obtido utilizando o lema descrito na Proposição 1. Além disso, apresentamos um critério de inserção de vértice em uma solução parcial apresentado na Proposição 2.

Proposição 1. *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 2. *Seja um grafo bipartido $G = (L \cup R, E)$ e um inteiro k. Considere $S \subset L$, com $0 < |S| < k$, uma solução ótima parcial. Para todo $i \in L \setminus S$, se $\cap(S) = \cap(S \cup \{i\})$ então $S \cup \{i\}$ é uma solução ótima parcial.*

Demonstração. Seja L' uma solução ótima tal que $i \notin L'$. Sem perda de generalidade, suponha que $S \subset L'$. Como $\cap(S) = \cap(S \cup \{i\})$, então $|\cap(L')| = |\cap(L' \cup \{i\})|$. Seja $j \in L' \setminus S$ e $L'' = L' \setminus \{i, j\}$, 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. □

3. Testes de Redução

Em Bogue et al. [2014], foi desenvolvido um procedimento de pré-processamento para a redução do tamanho das instâncias de entrada do problema. Seja $\mathcal{L}_u(\lambda) = \{v \in L \setminus \{u\} : |N_G(u) \cap N_G(v)| < \lambda\}, \forall u \in L$ e $\mathcal{R}_u(t) = \{v \in R \setminus \{u\} : |N_G(u) \cap N_G(v)| < t\}, \forall u \in R$, 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. [2014] é realizado da seguinte forma:

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

Propomos outros testes de redução que dominam os testes proposto por Bogue et al. [2014]. Seja $\overline{\mathcal{L}_u}(\lambda) = L \setminus (\mathcal{L}_u(\lambda) \cup \{u\}), \forall u \in L$ e $\overline{\mathcal{R}_v}(t) = R \setminus (\mathcal{R}_v(t) \cup \{v\}), \forall v \in R$. Definimos o conjunto $C_L(u) = \{v \in N_G(u) : |\overline{\mathcal{L}_u}(\lambda) \cap N_G(v)| \geq k - 1\}$ como o subconjunto de vértices em $N_G(u) \subseteq R$, tal que pertença também a vizinhança de pelo menos $k - 1$ vértices de $\overline{\mathcal{L}_u}(\lambda)$. De forma análoga, definimos o conjunto $C_R(v) = \{u \in N_G(v) : |\overline{\mathcal{R}_v}(t) \cap N_G(u)| \geq \lambda - 1\}$ como o subconjunto de vértices em $N_G(v) \subseteq L$, tal que pertença também a vizinhança de pelo menos $k - 1$ vértices de $\overline{\mathcal{R}_v}(t)$. O pré-processamento dos vértices de G proposto neste trabalho é realizado da seguinte forma:

1. Todo vértice $u \in L$, onde $|C_L(u)| < \lambda$, pode ser removido de G.
2. Todo vértice $v \in R$, onde $|C_R(v)| < k$, 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 $\overline{\mathcal{L}_u}(\lambda)$. Como os vértices em $\overline{\mathcal{L}_u}(\lambda)$ 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 v em G que também pertençam a vizinhança de pelo menos $\lambda - 1$ vértices de $\overline{\mathcal{R}_v}(t)$. Como os vértices em $\overline{\mathcal{R}_v}(t)$ são os únicos vértices em R que dado uma solução viável L' com $v \in \cap(L')$ e $\cap(L') \subseteq \overline{\mathcal{R}_v}(t) \cup \{v\}$. 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. [2014], a remoção de qualquer vértice pode afetar outro vértice, que até então não atendia as restrições do passo 1 ou 2. Vale destacar que a ordem com que os passos 1 e 2 são executados não importa. Dessa forma, repetimos os passos 1 e 2 até que não seja mais possível remover nenhum vértice de G. A seguir, é exibido a prova da dominância dos testes de redução apresentados aqui em relação aos testes de redução apresentados em Bogue et al. [2014].

Proposição 3. *Seja u um vértice do conjunto L e inteiros $k, \lambda > 0$. Se $d_G(u) < \lambda$ ou $|L| - |\mathcal{L}_u(\lambda)| < k$, 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 $|L| = |\mathcal{L}_u(\lambda)| + |\overline{\mathcal{L}_u}(\lambda)| + 1$, logo $|\overline{\mathcal{L}_u}(\lambda)| = |L| - |\mathcal{L}_u(\lambda)| - 1$. Se $|L| - |\mathcal{L}_u(\lambda)| < k$ então $|\overline{\mathcal{L}_u}(\lambda)| = |L| - |\mathcal{L}_u(\lambda)| - 1 < k - 1$. Como $|\overline{\mathcal{L}_u}(\lambda)| < k - 1$, para qualquer vértice $v \in N_G(u)$ temos que $|\overline{\mathcal{L}_u}(\lambda) \cap N_G(v)| < k - 1$. Portanto, se $|L| - |\mathcal{L}_u(\lambda)| < k$ temos que $C_L(u) = \emptyset$ e $|C_L(u)| = 0 < \lambda$. □

Proposição 4. *Seja v um vértice do conjunto R e inteiros $k, \lambda > 0$. Se $d_G(v) < k$ ou $|R| - |\mathcal{R}_v(k)| < \lambda$ então $|C_R(v)| < k$.*

A demonstração da Proposição 4 é análoga a demonstração da Proposição 3. Na seção 5, descrevemos os resultados obtidos quando comparamos a eficiência dos testes de redução propostos e os testes de redução apresentados em Bogue et al. [2014].

4. Algoritmo *Branch-and-Bound* `BBKMIS`

Nesta seção, apresentamos um algoritmo de branch-and-bound, denominado `BBKMIS`, para o problema `kMIS`. De maneira geral, um algoritmo de branch-and-bound realiza uma enumeração implicíta de todas as soluções viáveis de um problema. Na modelagem do `kMIS`, como um grafo bipartido $G = (L \cup R, E)$, uma solução viável é um conjunto com k vértices de L. A seguir, vamos apresentar a nossa estratégia de enumeração implícita dessas soluções.

Dado um grafo bipartido $G = (L \cup R, E)$ e um inteiro k, um subproblema do `kMIS` é definido por uma tupla (S, P, λ), onde $S \subseteq L$ representa um conjunto de vértices escolhidos a partir de L; $P \subseteq L$ é o conjunto de vértices candidatos, pois é formado pelos vértices que podem entrar em S; λ é a cardinalidade da maior intersecção de k vértices de L encontrada até o momento. No Algoritmo 1, é exibido o algoritmo de branch-and-bound proposto para o problema `kMIS`. Na chamada do algoritmo `BBKMIS`, consideramos $S = \emptyset$, $P = L$ e λ com o valor de uma solução inicial. No nosso algoritmo, a solução inicial é obtida pela heurística gulosa `kINTER ESTENDIDA` proposta em Costa et al. [2018].

Um subproblema (S, P, λ), pode ser podado por limitante, quando um limite superior para qualquer solução viável S', tal que $S \subseteq S'$, $S' \subseteq S \cup P$ e $|S'| = k$, não supere o λ. Na Linha 5, a poda por limitante é realizada. No nosso algoritmo, o procedimento de limite superior é baseado na Proposição 1. Também na linha 5, aplicamos uma poda por inviabilidade, pois quando $|S| + |P| < k$, 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. No nosso algoritmo, o procedimento para encontrar uma solução viável é uma adaptação da heurística `kINTER` apresentada em Bogue et al. [2013]. 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 a intersecção da vizinhança de v com $\cup(S')$ seja máxima.

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

Quando um vértice $i \in P$ é selecionado para a ramificação, podemos remover todos os vértices $j \in P$, onde $|\cap(S) \cap N_G(i) \cap N_G(j)| \leq \lambda$, pois os vértices i e j não podem estar presentes ao

mesmo tempo em uma solução L, tal que $S \subset L$ e $|\cap(L)| \geq \lambda+1$. Na Linha 12, é construído o novo conjunto de vértices candidatos P' removendo os vértices j, tal que $|\cap(S) \cap N_G(i) \cap N_G(j)| \leq \lambda$.

Na Linha 14, quando um vértice $i \in P$ é adicionado na solução atual, podemos descobrir outros vértices de P que podem ser selecionados de maneira ótima, conforme a Proposição 2. O conjunto D representa este conjunto de vértices selecionados.

Em Bogue et al. [2014], um procedimento de pré-processamento efetivo é proposto para reduzir o grafo de entrada. Esse procedimento permite a construção de um modelo de programação com número menor de restrições, comparado com o modelo kInter [Bogue et al., 2013].

Neste trabalho, adaptamos o teste de redução proposto por Bogue et al. [2014], 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 efetiva, conforme descrito na Seção 3.

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. Na seção 5.2, avaliamos os dois procedimentos de redução como procedimento de pré-processamento, ou seja, para a redução do problema inicial. Na seção 5.3, apresentamos qautro versões do algoritmo de branch-and-bound, variando a utilização dos procedimentos de redução nos subproblemas da árvore de enumeração.

Algoritmo 1: ALGORITMO BRANCH-AND-BOUND PROPOSTO

```
1  BBkMIS(S, P, λ):
2      if |S| ≥ k then
3          return S' ⊆ S tal que |S'| = k
4      LS ← LimiteSuperior(S, P)
5      if |S| + |P| < k ou LS ≤ λ then
6          return ∅
7      S_I ← LimiteInferior(S, P)
8      if |∩(S_I)| = LS then
9          return S_I
10     i ← CriterioRamificacao(S, P)
11     P ← P \ {i}
12     P' ← {j ∈ P : |∩(S) ∩ N_G(i) ∩ N_G(j)| > λ}
13     S' ← S ∪ {i}
14     D ← {j ∈ P : ∩(S') = ∩(S' ∪ {j})}
15     S_e ← BBkMIS(S' ∪ D, P' \ D, λ)
16     λ ← max(λ, |∩(S_e)|)
17     S_d ← BBkMIS(S, P \ {i}, λ)
18     if |∩(S_e)| > |∩(S_d)| then
19         return S_e
20     return S_d
```

5. Testes Computacionais

Nesta seção, apresentamos as classes de instâncias utilizadas nos testes computacionais e descrevemos o ambiente computacional. Em seguida, na Seção 5.2, comparamos os dois testes de redução como procedimento de pré-processamento. Na seção 5.3, comparamos quatro versões do algoritmo de branch-and-bound e o algoritmo M_CLIQUE.

5.1. Instâncias e Ambiente Computacional

Foram geradas instâncias de grafos de maneira aleatória, usando o modelo de grafos do Gilbert [1959], similares as instâncias geradas em Costa et al. [2018], que foram instâncias baseadas no trabalho de Bogue et al. [2013], onde os grafos diferenciam-se pela densidade e valor de k, conforme Tabela 1. Em Costa et al. [2018], para as instâncias com valores de k diferentes para uma mesma densidade, são geradas instâncias distintas. Optamos que para valores diferentes de k em uma mesma densidade, gerar apenas um grafo. Acreditamos que assim podemos avaliar melhor o comportamento dos algoritmos em relação a essas duas propriedades.

Tabela 1: 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	$[0.7\|L\|, 0.9\|L\|]$

Tabela 2: 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

Geramos 9 classes de instâncias de acordo com a Tabela 2. 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.

Após testes computacionais preliminares, decidimos escolher como valor de k o limite inferior em cada classificação e, assim como nos trabalhos de Bogue et al. [2013] e Costa et al. [2018], não foi possível gerar instâncias viáveis para a classe C_3. 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.

Neste trabalho, os subconjuntos são representados por meio de strings de bits, desta forma, todas as operações sobre os subconjuntos 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 [San Segundo et al., 2011], clique máxima ponderada [Tavares et al., 2015, 2016], k-plex máximo [Da Silva et al., 2017], intersecção máxima de k-subconjuntos [Costa et al., 2018].

Os algoritmos foram implementados na linguagem C++ e executados 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.

5.2. Pré-processamento

Iremos comparar os testes de redução propostos neste trabalho (TR1), com os testes de redução apresentados em Bogue et al. [2014] (TR0), em relação a quantidade de vértices que são removidos. Utilizamos como solução inicial para os testes a solução gerada pela heurística kINTER ESTENDIDA [Costa et al., 2018]. Além disso, iremos somar uma unidade a essa solução inicial, desta forma, os testes de redução irão remover os vértices que não poderão estar em uma solução viável melhor do que a solução encontrada pela heurística kINTER ESTENDIDA.

Destacamos inicialmente que nenhum dos testes conseguiu remover vértices nas classes C_8 e C_9 (classes de grafos mais densas). Além disso, nas classes C_2 e C_6, o TR1 não conseguiu remover nenhum vértice a mais em relação ao TR0, isso porque, para essas classes de densidades

baixa e média, respectivamente, e com k médio e alto, respectivamente, o TR0 consegue remover todos os vértices de um dos lados do grafo, indicando que a solução da heurística é ótima, ou removeu boa parte dos vértices até deixar o grafo com apenas uma solução viável, que é a ótima. Portanto, nessas classes o TR1 foi idêntico ao TR0.

Já nas classes C_1, C_4, C_5 e C_7, o teste de redução TR1 conseguiu remover uma quantidade de vértices superior a quantidade de vértices removidos pelo TR0, demonstrando na prática sua eficiência. Na Tabela 3, apresentamos os resultados obtidos para as instâncias viáveis dessas classes, onde as colunas indicam a quantidade de vértices removidos por classes e grupos.

Tabela 3: Resultados dos testes de redução para as classes C_1, C_4, C_5 e C_7

Classe	TR0				TR1			
	G_1	G_2	G_3	Total	G_1	G_2	G_3	Total
C_1	18	76	76	170	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	447	1159	**409**	**470**	**525**	**1404**

Podemos observar que nas classes de densidade baixa (C_1) e média (C_4, C_5) com k até médio, o TR1 é bem mais eficiente do que o TR0, conseguindo remover cerca de 29%, 42% e 31% a mais do que o TR0, respectivamente. Para densidade alta e k baixo (C_7), a eficiência do TR1, em relação ao TR0, diminuiu, conseguindo somente remover cerca de 2% a mais nessa classe. No total, o TR1 conseguiu remover 245 (21,14%) vértices a mais que o TR0.

Em relação ao tempo de execução o TR0 executou bem mais rápido, em média executou no tempo de 6.57 ms. Já o TR1, executou em média em 206.75 ms. Apesar disso, o tempo computacional do TR1 foi baixo, não chegando a 1.5 s em nenhuma instância de teste. Por fim, decidimos utilizar o TR1 como pré-processamento inicial das instâncias de teste.

5.3. Branch and Bound

Criamos quatro versões do Algoritmo 1, baseado na utilização de um dos testes de redução e no momento em que são aplicados. O BBKMIS_TR0 e BBKMIS_TR1 aplicam o teste TR0 e TR1, respectivamente, em todos os nós da árvore. 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, criamos duas versões, onde os testes de redução são aplicados apenas no ramo da árvore que ocorre a inserção do vértice em S, BBKMIS_TR0E e o BBKMIS_TR1E.

Nesta seção, vamos realizar uma comparação das quatro versões do algoritmo de *branch-and-bound* com o algoritmo do estado da arte apresentado em Bogue et al. [2014], que é baseado em uma formulação de Programação Linear Inteira M_CLIQUE, utilizando o Solver Matemático CPLEX na versão 12.7.

Observamos que os dois testes de redução resolvem completamente as instâncias geradas das classes C_2 e C_6, então essas duas classes de instâncias serão omitidas nas comparações realizadas nesta subseção.

A Tabela 4, apresenta a quantidade de instâncias que cada algoritmo conseguiu determinar a solução ótima. De modo geral, as versões do algoritmo *Branch-and-Bound* proposto, consegue garantir solução ótima para um maior número de instâncias nas classes C_1, C_4 e C_5, instâncias de densidade baixa e média. Enquanto que a formulação M_CLIQUE obtém resultados melhores nas classes C_7, C_8 e C_9, instâncias de densidade alta, sendo que para a classe C_7 a maior parte das versões *Branch-and-Bound* propostas, obteve o mesmo resultado que a formulação M_CLIQUE.

Das quatros versões do *Branch-and-Bound*, o BBKMIS_TR0 foi a que obteve melhores resultados gerais. Quando comparamos o BBKMIS_TR0 com o M_CLIQUE, usando a densidade do grafo como parâmetro, temos que, para as instâncias de densidade baixa (C_1), o BBKMIS_TR0 foi cerca de 67% melhor que o M_CLIQUE. Para as instâncias de densidades médias (C_4 e C_5), BBKMIS_TR0 foi cerca 49% melhor. Já para as instâncias de densidade alta (C_7, C_8 e C_9), o M_CLIQUE foi melhor, definindo cerca de 53% a mais de instâncias. Usando o valor do k como parâmetro de comparação, temos que, para k baixo (C_1, C_4 e C_7), o BBKMIS_TR0 foi cerca de 38% melhor, para k médio (C_5 e C_8), foi cerca de 17% melhor e para k alto (C_9), o M_CLIQUE foi cerca 100% melhor.

Tabela 4: Quantidade de soluções ótimas encontradas

Classe	BBKMIS_TR0				BBKMIS_TR1				BBKMIS_TR0E				BBKMIS_TR1E				M_CLIQUE			
	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	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**	6	6	6	18
C_4	4	4	5	13	4	4	4	12	4	5	5	**14**	4	4	5	13	3	3	4	10
C_5	10	10	10	**30**	6	8	6	20	8	10	8	26	6	8	6	20	6	6	7	19
C_7	4	4	4	**12**	3	4	4	11	4	4	4	**12**	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	3	4	4	**11**
C_9	2	2	2	6	2	2	2	6	2	2	2	6	2	2	2	6	4	4	4	**12**
Total	**31**	32	**33**	**96**	26	29	27	82	29	**33**	31	93	27	30	28	85	26	27	29	82

Analisando a quantidade de instâncias totais definidas, todas as versões do algoritmo *Branch-and-Bound* proposto obteve resultados superiores ou iguais os obtidos pela M_CLIQUE. O BBKMIS_TR0, foi a que obteve melhores resultados gerais, definindo 14 instâncias a mais que a formulação M_CLIQUE, destacando-se principalmente nas classes C_1 e C_5.

A fim de analisar o comportamento dos algoritmos avaliados neste trabalho, resolvemos observar o comportamento dos mesmo no conjunto de instâncias em que todos os algoritmos conseguiram garantir a solução ótima no tempo delimitado. Este conjunto tem um total de 101 instâncias. A Tabela 5, apresenta a comparação de tempo de execução médio, em segundos, por classe de instância dos algoritmos neste conjunto. Já a Tabela 6, apresenta a comparação de quantidade de nós médio gerados pelas árvores de enumeração por classe de instâncias dos algoritmos neste mesmo conjunto.

Tabela 5: Média de tempo por classe

Classe	BBKMIS_TR0	BBKMIS_TR1	BBKMIS_TR0E	BBKMIS_TR1E	M_CLIQUE
C_1	3.0250	11.5572	**1.2722**	1.6761	236.6105
C_4	2.2160	11.3370	**1.0960**	2.3930	318.143
C_5	**21.6835**	138.3623	42.2241	122.5394	162.3447
C_7	15.7927	160.0754	8.5130	38.3790	**5.7772**
C_8	1.9133	7.6433	1.4500	3.9166	**0.01666**
C_9	64.4983	288.7050	92.6466	238.195	**0.0800**

Analisando a Tabela 5, podemos notar que o tempo de execução médio das versões do algoritmo *Branch-and-Bound* proposto é sempre menor que o da formulação M_CLIQUE nas classes C_1, C_4 e C_5. Já nas classes C_7, C_8 e C_9 o M_CLIQUE, apresenta um menor tempo médio de execução. O mesmo acontece quando analisamos na Tabela 6 o número de nós médio gerados pela árvore de enumeração.

Vale destacar que a versão BBKMIS_TR0, sempre obteve um tempo médio de execução menor que a versão BBKMIS_TR1, e que BBKMIS_TR0E sempre obteve um tempo médio de execução menor que a versão BBKMIS_TR1E. Acreditamos que isso deve-se ao fato do tempo de

Tabela 6: Quantidade média de nós por classe

Classe	BBKMIS_TR0	BBKMIS_TR1	BBKMIS_TR0E	BBKMIS_TR1E	M_CLIQUE
C_1	1310.11	**715.22**	3419.66	2081.77	19628.00
C_4	6943.80	**4765.60**	11117.60	8119.80	403175.70
C_5	7053.58	**3414.05**	44569.35	22708.41	46125.70
C_7	26504.63	22985.90	37342.09	32196.27	**13316.45**
C_8	11015.00	8985.00	20755.66	17133.66	**12.33**
C_9	164672.55	132577.44	785673.33	634919.66	**168.00**

execução dos testes de redução TR1 ser superior ao tempo de execução dos testes de redução TR0. Além disso, com exceção da classe C_5, as versões que aplicam os testes de redução apenas quando inserimos um novo vértice na solução parcial se mostraram mais rápidas que as versões que aplicam os testes de redução em todos os nós da árvore de enumeração. Exatamente o contrário acontece quando analisamos na Tabela 6 o número de nós médio analisados na árvore.

Com a finalidade de analisar esta propriedade, geramos os gráficos apresentados nas Figuras 2 e 3. 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, e por não se tratar de uma classe onde os algoritmos garantem a solução em tempo relativamente pequeno.

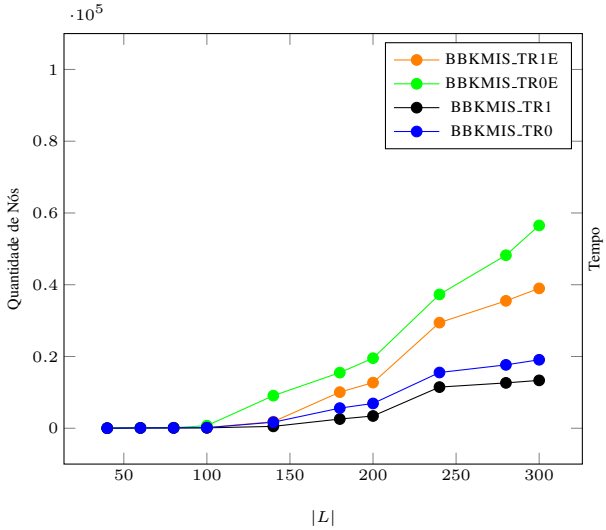


Figura 2: Número de nós gerados nas instâncias da classe C_1 do grupo G_1

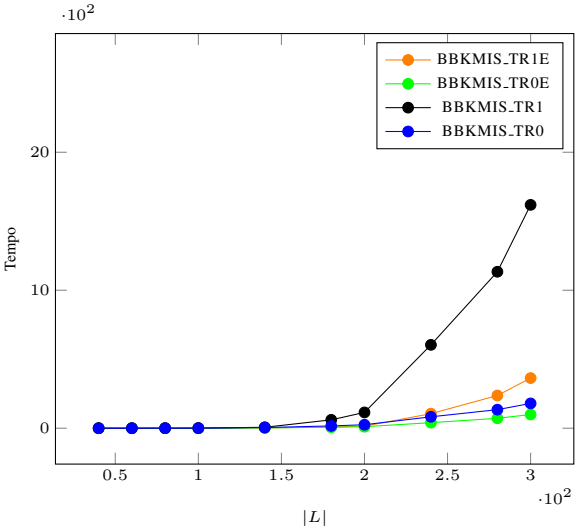


Figura 3: Tempo de execução nas instâncias da classe C_1 do grupo G_1

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. Esta disparidade do número de nós faz com que o tempo de execução da versão BBKMIS_TR0 seja quase o mesmo tempo de execução da versão BBKMIS_TR0E. Acreditamos que o fato da disparidade de nós seja bastante significativa entre essas duas versões, faz com que a versão BBKMIS_TR0 consiga garantir a solução ótima em um maior número de instâncias. O mesmo não acontece quando comparamos as versões BBKMIS_TR1 e BBKMIS_TR1E.

Observe que apesar da redução da quantidade de nós gerados na versão BBKMIS_TR1

ser inferior as demais versões, o seu tempo de execução é bem superior. Notamos que isso se deve ao fato do teste de redução TR1 ser bastante custoso. De maneira geral, a versão BBKMIS_TR0 foi a que apresentou os melhores resultados.

6. Conclusão e Trabalhos Futuros

Neste trabalho, propomos um algoritmo de branch-and-bound para o `kMIS`. Para este algoritmo, elaboramos procedimentos de limite superior e inferior, uma estratégia de ramificação e um novo critério de redução do tamanho do problema.

Os testes computacionais, mostraram que os testes de redução propostos são mais efetivos, como procedimento de pré-processamento, que os testes de redução apresentados em Bogue et al. [2014].

Avaliamos quatro versões do algoritmo branch-and-bound, a fim de verificar o desempenho dos testes de redução aplicados aos subproblemas gerados pelo algoritmo. De maneira geral, os testes computacionais indicaram que a versão BBKMIS_TR0, resolve um número maior de instâncias dentro do limite de tempo estabelecido do que as demais versões e o algoritmo de estado da arte M_CLIQUE, apresentado em Bogue et al. [2014]. Apesar de reduzir melhor o tamanho dos subproblemas gerados, acreditamos que a complexidade de tempo dos testes de redução propostos aplicados ao algoritmo branch-and-bound, torne o tempo de execução do algoritmo proibitivo.

Nas classes de instâncias com a densidade mais alta (C_7, C_8 e C_9), o algoritmo M_CLIQUE mostrou-se um pouco mais eficiente que todos os outros testados, exceto na classe C_7 de k baixo, onde praticamente todos os algoritmos obtiveram o mesmo resultado. Conjecturamos que a proposição de uma estratégia de ramificação baseada em cliques possa ser mais eficiente para essas classes de instâncias. Além disso, podemos investigar estruturas de dados que levem a uma redução do tempo de execução do algoritmo.

Referências

Bogue, E. T., de Souza, C. C., Xavier, E. C., e Freire, A. S. (2013). O problema da máxima interseção de k-subconjuntos. In *Anais do XLV SBPO, September 16-19, 2013, Natal, RN, Brazil*, p. 2416–2425. Sobrapo.

Bogue, E. T., de Souza, C. C., Xavier, E. C., e Freire, A. S. (2014). An integer programming formulation for the maximum k-subset intersection problem. In *Combinatorial Optimization*, p. 87–99, Cham. Springer International Publishing. ISBN 978-3-319-09174-7.

Costa, J. R. d. F., Dias, F. C. S., e de A. Tavares, W. (2018). Um algoritmo vns para o problema da máxima interseção de k-subconjuntos. In *Anais do L SBPO, August 06-09, 2018, Rio de Janeiro, RJ, Brazil*. Sobrapo.

Da Silva, M. R. C., Tavares, W. A., Dias, F. C. S., e Neto, M. B. C. (2017). Algoritmo branch-and-bound para o problema do k-plex máximo. In *Anais do XLIX SBPO, August 27-30, 2017, Blumenau, SC, Brazil*. Sobrapo.

Gilbert, E. N. (1959). Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144.

Nussbaum, D., Pu, S., Sack, J.-R., Uno, T., e Zarrabi-Zadeh, H. (2012). Finding maximum edge bicliques in convex bipartite graphs. *Algorithmica*, 64(2):311–325.

San Segundo, P., Rodríguez-Losada, D., e Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

Shieh, M.-Z., Tsai, S.-C., e Yang, M.-C. (2012). On the inapproximability of maximum intersection problems. *Information Processing Letters*, 112(19):723–727.

Tavares, W. A., Neto, M. B. C., Rodrigues, C. D., e Michelon, P. (2015). Um algoritmo de branch and bound para o problema da clique maxima ponderada. In *Anais do XLVII SBPO, September 25-28, 2015, Porto de Galinhas, PE, Brazil*, volume 1, p. 2655–2666. Sobrapo.

Tavares, W. A., Neto, M. B. C., Rodrigues, C. D., e Michelon, P. (2016). Bitclique: um algoritmo de branch-and-bound para o problema da clique máxima ponderada. In *Anais XLVIII SBPO, September 27-30, 2016, Vitória, ES, Brazil*, volume 1, p. 2429–2440. Sobrapo.

Vinterbo, S. (2002). A note on the hardness of the k-ambiguity problem. Technical report, Harvard Medical School, Boston, MA, USA.

Xavier, E. C. (2012). A note on a maximum k-subset intersection problem. *Information Processing Letters*, 112(12):471–472.