Algoritmos Exatos e Heurísticos para o Problema da Detecção de Cliques com Peso acima de um Limiar

Matheus Guedes Vilas Boas
Universidade Federal de Ouro Preto (UFOP)
Ouro Preto - Brasil
matheusgueedes91@gmail.com

Haroldo Gambini Santos
Universidade Federal de Ouro Preto (UFOP)
Ouro Preto - Brasil
haroldosantos@gmail.com

Samuel Souza Brito
Universidade Federal de Ouro Preto (UFOP)
Ouro Preto - Brasil
samuelufop@gmail.com

RESUMO

O presente trabalho trata do projeto, implementação e avaliação de algoritmos exatos e heurísticas para a resolução do Problema da Detecção de Cliques com Peso acima de um Limiar. Foram usados 3 algoritmos para a resolução do problema: Algoritmo de Bron-Kerbosch, Algoritmo de Ostergard e a Heurística Busca Tabu com Multi-Vizinhanças. Todos esses foram adaptados para resolver o problema abordado neste trabalho. Nos experimentos computacionais, foram utilizadas 7292 instâncias, oriundas de quatro conjuntos referentes a separação de cortes em problemas de Programação Inteira. Os resultados obtidos comprovam a eficiência do Algoritmo de Bron-Kerbosch adaptado (BKv4), o qual encontrou a solução ótima para todas as instâncias e em um tempo consideravelmente menor que as outras técnicas. Uma tentativa, sem sucesso, de paralelizar o algoritmo de Bron-Kerbosch (BKv4), através da divisão do grafo em subgrafos conexos, foi realizada. Nesse sentido, são sugeridos como trabalhos futuros a paralelização das técnicas implementadas neste trabalho.

PALAVRAS CHAVE. Problema da Detecção de Cliques com Peso acima de um Limiar. Algoritmos exatos. Heurística Busca Tabu. Área Principal: Otimização Combinatória.

ABSTRACT

This paper deals with the design, implementation and evaluation of exact and heuristic algorithms for solving Clique Detection Problem with Weight above a Threshold. We used 3 algorithms for solving the problem: Bron-Kerbosch Algorithm, Ostergard Algorithm and Multi-neighborhood Tabu Search Heuristic. All these have been adapted to solve the problem presented in this work. In the computational experiments, we used 7292 instances, coming from four sets for the separation cuts in Integer Programming problems. The results prove the efficiency of Bron-Kerbosch algorithm adapted (BKv4), which found the optimal solution for all instances and in considerably less time than the other techniques. An attempt, unsuccessfully, to parallelize the Bron-Kerbosch algorithm (BKv4) by graph division in related subgraphs, was performed. In this sense, are suggested as future work parallelization techniques implemented in this work.

KEYWORDS. Clique Detection Problem with Weight above a Threshold. Exact Algorithms. Tabu Search Heuristic. Main Area: Combinatorial Optimization.

1. Introdução

O problema do Clique consiste em encontrar subgrafos completos em um grafo. Um subgrafo é completo quando para cada par de vértices há uma aresta de ligação. Formalmente, dado um grafo não direcionado $G = (V, A)$, tal que V representa o conjunto de vértices e A representa o conjunto de arestas, então $G_1 = (V_1, A_1)$ denomina-se subgrafo de G se $V_1 \subseteq V$ e $A_1 \subseteq A$, onde cada aresta de A_1 incide nos vértices de V_1. Existem vários problemas relacionados à detecção de cliques. Os principais são apresentados a seguir.

1. Problema do Clique Máximo: Consiste em determinar o clique com a maior cardinalidade em um grafo (Ostergard, 2002) e (Debroni *et al*., 2011).

2. Problema do Clique com Peso Máximo: Dado um grafo com vértices ponderados, o objetivo do problema é encontrar o clique de maior peso. (Ostergard, 2001) e (Wu *et al*., 2012).

3. Problema do *k*-Clique: Consiste em encontrar todos os cliques de tamanho *k* (Goldschmidt *et al*., 1996) e (Vassilevska, 2008).

O Problema da Detecção de Cliques modela diversas áreas de aplicações, tais como: Problemas de otimização em pesquisa operacional, Atribuição de frequências em comunicação a rádio, criptografia, determinação de trilhas em circuitos impressos, emparelhamento de sequências de aminoácidos em proteínas, redes sociais, bioinformática, química computacional, reconhecimento de padrões, robótica, entre outras.

O presente trabalho lida com o Problema da Detecção de Cliques com Peso acima de um Limiar. No contexto da Programação Inteira, encontrar todos os cliques acima de um dado peso é equivalente ao problema de encontrar todas as desigualdades violadas de clique. Segundo Chvátal (1973), esses cortes desempenham um papel fundamental na geração de desigualdades fortes para problemas com estrutura de empacotamento de nós (Frota, 2004) e (Coll *et al*., 2002). É importante destacar aqui uma particularidade sobre o problema de otimização envolvido na separação de desigualdades. Apesar do problema de separação de cortes poder ser formulado em termos de se encontrar a desigualdade mais violada, experimentalmente se verificou que mais importante do que a descoberta dessa desigualdade é a descoberta de um grande conjunto de cortes violados (Fischetti e Lodi (2007)) e (Burke *et al*. (2011)).

O restante do trabalho está estruturado da seguinte forma: Na Seção 2 é apresentado o Algoritmo de Bron-Kerbosch adaptado. Quatro implementações foram consideradas, as quais se diferem na representação dos conjuntos utilizados no algoritmo. Nas Seções 3 e 4 são apresentados o algoritmo de Ostergard e a heurística Busca Tabu com Multi-Vizinhanças para a resolução do problema, respectivamente. É válido ressaltar que estes também foram adaptados para resolver o Problema da Detecção de Cliques com Peso acima de um Limiar. A Seção 5 apresenta os experimentos computacionais e relata as principais conclusões obtidas em relação ao tempo computacional gasto para a execução de cada algoritmo implementado, bem como um comparativo dos mesmos. Por fim, a Seção 6 encerra o trabalho apontando as principais conclusões obtidas, bem como o rumo de pesquisa futura.

2. Algoritmo de Bron-Kerbosch

O algoritmo de Bron-Kerbosch clássico (Bron e Kerbosch, 1973) tem como objetivo encontrar todos os cliques maximais de um grafo *G*. É utilizado o processo de *backtracking*, e apenas cliques maximais são gerados, evitando assim uma boa parte da comparação de um determinado conjunto de cliques que já foram testados.

No trabalho de Brito e Santos (2011), os autores adaptaram o algoritmo de Bron-Kerbosch básico para refletir o Problema da Detecção de Cliques com Peso acima de um Limiar. Além disso, propuseram algumas melhorias no algoritmo, descritas sucintamente a seguir. Primeiramente, utilizaram-se da noção de vértice pivô (Tomita *et al*. (2006)), com o propósito de podar alguns

ramos que não contém cliques maximais. O vértice de máximo grau modificado foi escolhido como vértice pivô. Tal conceito se refere à soma dos graus de um vértice v qualquer e os graus dos seus vértices vizinhos. Por fim, fora utilizado uma técnica de estimativa de peso, a qual calcula o limite superior do peso de um clique parcial. Nesse sentido, alguns cliques parciais podiam ser previamente eliminados. O algoritmo implementado por Brito e Santos (2011) é apresentado em Algoritmo 1.

Algoritmo 1 Algoritmo de Bron-Kerbosch adaptado

```
1:  procedimento BKA(C, P, S)
2:      se P e S estão vazios então
3:          se w(C) ≥ limiar então
4:              Adiciona o clique C no conjunto solução;
5:          fim se
6:      fim se
7:      se w(C) + h(C) ≥ limiar então
8:          Escolha um vértice pivô u de P;
9:          para cada Vértice v em P\N(u) faça
10:             BKA(C ∪ {v}, P ∩ N(v), S ∩ N(v));
11:             P := P\{v};
12:             S := S ∪ {v};
13:         fim para cada
14:     fim se
15: fim procedimento
```

O conjunto C representa os vértices que fazem parte do clique. O conjunto P representa os vértices que têm ligação com todos os vértices de C. Por sua vez, o conjunto S contém todos os vértices que já foram analisados e não levam a uma extensão do conjunto P. O peso do Clique C está armazenado em $w(C)$. O somatório dos pesos de todos os vértices do conjunto P está em $h(C)$, onde tal variável é usada como um limite superior para o peso do Clique C.

No presente trabalho, foram realizadas 4 implementações do algoritmo de Bron-Kerbosch adaptado, apresentado em Brito e Santos (2011). Essas implementações são diferentes na forma de armazenamento de G.

- **Uso de matriz**: Nesta implementação foi utilizada uma matriz binária m de dimensão $|\mathrm{V}|$ x $|\mathrm{V}|$ para representação das arestas existentes no grafo. Nesse sentido, caso exista uma aresta conectando os vértices v_1 e v_2, $m[v_1][v_2] = 1$ e $m[v_2][v_1] = 1$. Os valores 0 na matriz binária representam que os vértices não estão conectados. Esta abordagem é vantajosa no processamento do Algoritmo de Bron-Kerbosch adaptado, no momento da verificação da existência ou não de aresta conectando um par de vértices qualquer. Tal verificação é realizada em O(1). Como desvantagem, há um uso considerável de memória.

- **Uso de lista**: Com o uso desta implementação, os vértices vizinhos de um vértice v_1 qualquer são representados por uma lista de adjacência. Em relação a abordagem anterior (Uso de matriz), esta economiza o uso de memória. Além disso, a interseção realizada na linha 10 do Algoritmo 1, é feita de forma mais eficiente, visto que não é necessário percorrer toda a matriz para procurar a interseção dos vértices do conjunto P ou S com os vértices vizinhos do vértice v. Como desvantagem, o tempo gasto para verificação da existência de arco conectando dois vértices qualquer pode ser dispendioso (O(n)).

- **Uso de lista e matriz**: Esta terceira abordagem considera uma implementação híbrida, com o uso de lista e matriz. Nesse sentido, quando da verificação de existência de aresta entre

dois vértices, é utilizada a matriz para esse fim. Quando o objetivo é realizar a interseção do conjunto *P* ou *S* com os vértices vizinhos do vértice *v*, a lista de adjacência é usada.

- **Ordenação do conjunto *P***: Na linha 8 do Algoritmo 1, o vértice pivô *u* é escolhido a partir do vértice pertencente ao Conjunto P com o Máximo Grau Modificado. Uma quarta implementação do Algoritmo Bron-Kerbosch considera a otimização realizada com o uso de lista e matriz (3ª implementação), acrescida da manutenção do conjunto *P* sempre ordenado, de modo a evitar percorrer todo esse conjunto quando da escolha do vértice pivô.

3. Algoritmo de Ostergard

O Algoritmo de Ostergard foi desenvolvido para encontrar o clique máximo em um grafo (Ostergard, 2002). O autor também desenvolveu uma versão do algoritmo para obtenção do clique de peso máximo em um grafo (Ostergard, 2001). O algoritmo impõe uma ordenação aos vértices da seguinte forma: Colore-se o clique uma cor de cada vez, sempre adicionando os vértices pela ordem de importância, dada pelo vértice de peso mínimo no grafo restante, seguido pelo vértice vizinho que tem a maior soma de pesos. Vale ressaltar aqui que em uma coloração de vértices, vértices adjacentes não podem receber cores iguais. Ao determinar uma nova classe de cor, o grafo induzido pelos vértices sem cor é construído e em seguida, contanto que exista um vértice que pode ser adicionado à classe de cor, o vértice com maior grau é adicionado. Os vértices são rotulados como v_n, v_{n-1}, ..., v_1 na ordem em que foram adicionados a uma classe de cor.

Um conjunto de vértices denominado *working set*, adjacentes a todos os vértices fixos na busca pelo clique de peso máximo, é mantido. Em cada nível da árvore de busca, um vértice do conjunto citado anteriormente é adicionado aos vértices fixos e os conjuntos são atualizados. Algumas subárvores são podadas, quando a soma do peso do conjunto atual e do conjunto de trabalho (*working set*) é menor ou igual ao peso de um clique já construído. O algoritmo de Ostergard, adaptado para resolver o problema de interesse do presente trabalho, é apresentado em Algoritmo 2.

Algoritmo 2 Algoritmo de Ostergard adaptado

```
1:  para cada i := n até 1 faça
2:      WCLIQUE(S_i ∩ N(v_i), w[i])
3:  fim para cada
4:  função WCLIQUE(U, peso)
5:      se |U| = 0 então
6:          se peso ≥ limiar então
7:              se cliqueNaoDominado() então
8:                  Adicione o clique no conjunto solução.
9:              fim se
10:         fim se
11:     fim se
12:     enquanto U ≠ ∅ faça
13:         se peso + wt(U) ≤ limiar então
14:             return
15:         fim se
16:         i := min{j | v_j ∈ U}
17:         U := U\{v_i}
18:         WCLIQUE(U ∩ N(v_i), peso + w[i])
19:     fim enquanto
20:     return
21: fim função
```

As adaptações necessárias para refletir o mesmo problema abordado no atual trabalho estão contidas nas linhas 3-7. Em suma, são adicionados ao conjunto solução, todos os cliques com o peso acima de um limiar, previamente informado. Uma verificação adicional é necessária para garantir que não existam cliques dominados por outros cliques no conjunto solução.

Dois trabalhos presentes na literatura estudaram a eficiência do Algoritmo de Ostergard para o Problema do Clique de Peso Máximo: No trabalho de Yamaguchi e Masuda (2008), os autores propõem um novo algoritmo exato que apresenta melhores resultados - quando comparados ao Algoritmo de Ostergard - para grafos com densidade acima de 0.8. O mesmo ocorre no trabalho de Kumlander (2008), onde é utilizada uma melhoria na ordenação dos vértices antes que ocorra a coloração do grafo. Este novo algoritmo é três vezes mais rápido que o Algoritmo de Ostergard.

4. Heurística Busca Tabu com Multi-Vizinhanças

No trabalho de Wu *et al*. (2012), os autores implementaram a heurística Busca Tabu com Multi-Vizinhanças para a resolução do Problema do Clique de Peso Máximo. A metaheurística utiliza-se de 3 vizinhanças, cada qual com movimentos diferentes. Ao contrário da Busca Tabu tradicional, a Busca Tabu nesse trabalho explora, em cada iteração, a união das 3 vizinhanças e seleciona a melhor solução admissível (não-tabu ou melhoria global) da vizinhança, ou seja, a solução que produz o maior ganho de peso. Além disso, os autores implementaram um mecanismo de busca tabu dedicado e a estratégia de reinicialização aleatória.

Em suma, a heurística funciona do seguinte modo: A construção da solução inicial é realizada de forma aleatória, onde um vértice v_1 é adicionado ao clique C, inicialmente vazio. Logo após, outro vértice v_2 é escolhido para ser incluído no clique, desde que este não faça parte do clique ($v_2 \notin C$) e esteja ligado a todos os vértices pertencentes ao mesmo. Esse procedimento é repetido até que não exista mais nenhum vértice v_2 com tal característica.

Os operadores básicos de movimento são denominados *ADD*, *SWAP* e *DROP*. Tais operadores são baseados em dois subconjuntos de vértices - *PA* e *OM*- relacionados a um dado clique C. O subconjunto *PA* é composto dos vértices que não estão inclusos no clique C, porém estão conectados a todos os vértices de C. Por sua vez, o subconjunto *OM* é composto dos vértices que não estão inclusos no clique C, porém estão conectados a $|C|$ - 1 vértices do clique.

O operador *ADD* adiciona um vértice *v* ($v \in PA$) ao corrente clique C. É utilizado somente quando $PA \neq \emptyset$ e representa sempre uma melhora no peso do clique. O operador *SWAP* realiza a troca de um vértice v_1 ($v_1 \in OM$) com o vértice v_2 ($v_2 \in C$) que não esteja ligado com o vértice v_1. Esse movimento pode representar uma piora na qualidade da solução corrente. Por sua vez, o operador *DROP* remove um vértice *v* do clique *C*. Esse operador de movimento sempre diminui o peso do clique.

Para estabelecer uma forma mais global de diversificação, com o objetivo de visitar áreas ainda inexploradas no espaço de busca, os autores utilizaram a estratégia de reinicialização aleatória. Esse processo é acionado sempre que a busca atual estiver presa em um ótimo local profundo. Formalmente, isso acontece quando o número máximo permitido de iterações consecutivas sem melhora, representado por *L*, é excedido: $NI > L$.

A heurística Busca Tabu com Multi-Vizinhanças - adaptado para resolver o Problema da Detecção de Cliques acima de um Limiar - é apresentada em Algoritmo 3.

As seguintes modificações foram realizadas no algoritmo original proposto por Wu *et al*. (2012): O inteiro *L*, que representa a profundidade da busca, foi definido como a quantidade de vértices do grafo ($L = |V|$). Na construção da solução inicial, a cada iteração é escolhido um vértice v_1 diferente, iniciando-se de $v_1 = 1$ até $v_1 = |V|$. No algoritmo original de Wu *et al*. (2012), tal vértice era escolhido sempre de forma aleatória, de modo que essa abordagem não produziu bons resultados no problema em questão abordado no presente trabalho.

A execução é encerrada quando o tempo limite de execução é excedido ou a solução ótima é encontrada, ou seja, todos os cliques acima de um limiar foram detectados. Os cliques são adicionados ao conjunto solução *CSol* quando o peso dos mesmos forem maior que um dado limiar

Algoritmo 3 Heurística Busca Tabu com Multi-Vizinhanças adaptado

Entrada: Um grafo ponderado $G = (V, E, w)$, inteiro L (profundidade da busca), inteiro $QtdCliques$ (solução ótima)

Saída: Conjunto $Csol$ de cliques com peso acima de um limiar (*PesoMin*)

1: **enquanto** ($tempoExecucao < tempoLimite$ e $|Csol| < QtdCliques$) **faça**
2: C = $Inicializar$()
3: Iniciar $lista\_tabu$
4: NI = 0
5: **se** ($w(C) \geq PesoMin$ e $PA = \emptyset$) **então**
6: **se** ($cliqueNaoRepetido(C, Csol)$) **então**
7: Adiciona o clique C no conjunto solução $Csol$.
8: **fim se**
9: $NI = L$
10: **fim se**
11: **enquanto** ($NI < L$ e $|Csol| < QtdCliques$) **faça**
12: Construção das vizinhanças N_1, N_2 e N_3 de C
13: Escolha o melhor vizinho global permitido $C' \in N_1 \cup N_2 \cup N_3$
14: $C = C'$ {Mover para a nova solução}
15: **se** ($w(C) \geq PesoMin$ e $PA = \emptyset$) **então**
16: **se** ($cliqueNaoRepetido(C, Csol)$) **então**
17: Adiciona o clique C no conjunto solução $Csol$.
18: **fim se**
19: $NI = L$
20: **fim se**
21: $NI = NI + 1$
22: Atualize $lista\_tabu$
23: **fim enquanto**
24: **fim enquanto**
25: **return** Conjunto $Csol$

e o conjunto *PA* estiver vazio, a fim de evitar que cliques dominados sejam adicionados ao conjunto solução. Além disso, por se tratar de uma heurística, vários cliques repetidos - cliques que possuem exatamente os mesmos vértices - podem ser encontrados. A heurística verifica isso e considera apenas um dos cliques repetidos.

5. Experimentos Computacionais

Os experimentos computacionais foram realizados em um computador com processador Intel(R) Core(TM) i7-4790, 3.60Hz, 16GB de Memória RAM. Foi estabelecido como critério de parada um tempo limite de execução (TLE) de 60 segundos para a resolução de cada instância. Além disso, para aquelas instâncias onde o tempo de execução foi inferior a 0,001 segundos, foi atribuído o tempo padrão de 0,001 segundos.

Ao todo, foram utilizadas 7292 instâncias para a realização dos experimentos, oriundas de quatro conjuntos de problemas de Programação Inteira. É válido ressaltar aqui que, devido à limitação do número de páginas, tornou-se impossível descrever minunciosamente os resultados encontrados para todas as instâncias. Nesse sentido, as subseções seguintes descrevem como foram geradas as instâncias e os resultados obtidos para cada conjunto de problemas.

A seguinte nomenclatura será usada como referência para cada técnica implementada: As 4 versões utilizadas do Algoritmo Bron-Kerbosch (ver Seção 2) serão nomeadas como **BK(v1)**, **BK(v2)**, **BK(v3)** e **BK(v4)**, as quais representam respectivamente, a implementação do algoritmo utilizando matriz, lista, híbrido (matriz e lista) e usando o conjunto *P* ordenado. O algoritmo de

Ostergard será nomeado como **OSTERGARD** e a heurística Busca Tabu com Multi-Vizinhanças será representado como **BT**.

5.1. Caracterização das instâncias

As instâncias utilizadas neste trabalho foram geradas a partir de quatro conjuntos de problemas de Programação Inteira. O primeiro conjunto (**MIPLIB**) apresenta instâncias de *benchmark* da MIPLIB 2010 (Koch *et al*., 2011), contendo 87 instâncias. Desde a sua introdução em 1992, a MIPLIB tornou-se uma biblioteca padrão de testes, usada para comparar o desempenho dos resolvedores de PI. Ela contém uma coleção de problemas reais, sendo na maior parte aplicações industriais.

O segundo conjunto de instâncias (**INRC**) foi obtido da formulação usada por Santos *et al*. (2014) para resolver problemas da International Nurse Rostering Competition (Haspeslagh *et al*., 2014), contendo 60 instâncias. Trata-se de uma competição realizada para incentivar e comparar pesquisas relacionadas ao Problema de Escalonamento de Enfermeiras.

Por sua vez, o terceiro conjunto consiste de problemas de planejamento de rotas para o **Telebus** (Borndörfer *et al*., 1999), um serviço de transporte de pessoas com deficiência física localizado em Berlim, contendo 28 instâncias. Esses problemas apresentam formulações de PI baseadas no problema de particionamento de conjuntos.

O quarto conjunto (**Uchoa**) consiste de apenas uma instância do problema de *P*-Dispersão, reduzido a uma sequência de problemas de conjunto independente. Tal instância possui muitas restrições esparsas, de modo que a separação de cortes encontra muitos cliques violados.

Para cada problema de Programação Inteira dos conjuntos citados anteriormente, foram gerados vários grafos de conflitos. Esses grafos foram construídos a partir da utilização da rotina de separação de cortes desenvolvida no trabalho de Brito (2015). Nessa rotina, um grafo inicial é construído a partir da análise das restrições impostas pelo programa inteiro. Os demais grafos são construídos com base na análise do problema gerado pela aplicação iterativa de Planos de Corte. Com esse procedimento, foram geradas um total de 7292 instâncias, sendo 399 instâncias geradas a partir do conjunto MIPLIB, 3804 do INRC, 3085 do Telebus e 4 do Uchoa.

A Tabela 1 apresenta as características relacionadas à densidade de cada um dos quatro conjuntos de problemas de Programação Inteira, descritos anteriormente. Um grafo é denso quando possui muitas arestas para uma determinada quantidade de vértices. Quando o grafo possui poucas arestas, ele é dito esparso.

Tabela 1: Densidade para cada conjunto dos problemas de Programação Inteira

Densidade/Conjunto	**MIPLIB**	**INRC**	**Telebus**	**Uchoa**
Densidade Média	0,14	0,01	0,04	0,16
Densidade Mínima	0,01	0,01	0,01	0,11
Densidade Máxima	0,84	0,07	0,37	0,21

Ao se analisar a Tabela 1, pode-se perceber que os conjuntos **INRC**, **Telebus** e **Uchoa** apresentam grafos esparsos. Tal conclusão se dá ao observar os valores da densidade média, mínima e máxima de cada um desses conjuntos. O conjunto **MIPLIB** apresenta algumas instâncias com uma quantidade grande de arestas (grafo denso). A densidade máxima para este conjunto é de 0,84.

5.2. MIPLIB

O conjunto MIPLIB apresenta um total de 399 instâncias, as quais variam o número de vértices entre 7 e 8055, enquanto que o número de arestas varia entre 6 e 91459. A Tabela 2 apresenta um resumo comparativo do tempo total (em segundos) despendido para a execução de todas as instâncias do conjunto, utilizando cada uma das técnicas de resolução implementadas. A tabela informa também qual foi o menor (**TMin**) e maior (**TMax**) tempo despendido na execução das instâncias do conjunto. Por fim, a última coluna da tabela apresenta a quantidade de instâncias que excederam o tempo limite de execução (**TLE**).

Tabela 2: Sumário dos resultados obtidos para as instâncias MIPLIB

Algoritmo	Tempo total	TMax	TMin	TLE
BK(v1)	354,27	TLE	0,001	1
BK(v2)	385,45	TLE	0,001	2
BK(v3)	106,50	18,122	0,001	0
BK(v4)	65,25	11,763	0,001	0
OSTERGARD	10259,84	TLE	0,001	169
BT	6785,60	TLE	0,001	77

Dentre os algoritmos desenvolvidos, o Algoritmo de Bron-Kerbosch **BK(v4)** obteve os melhores resultados, ao conseguir encontrar a solução ótima para todas as instâncias do conjunto MIPLIB, utilizando o menor tempo de CPU. A instância mais crítica foi executada em um tempo de 11,763 segundos. Utilizando-se do Algoritmo de Bron-Kerbosch **BK(v3)** foram necessários aproximadamente 18 segundos. As outras técnicas não encontraram a solução ótima dentro do tempo limite de execução (TLE). Considerando o Algoritmo de Ostergard e a Heurística Busca Tabu com Multi-Vizinhanças, 169 e 77 instâncias não encontraram a solução ótima dentro do tempo limite de execução, respectivamente. A Tabela 3 apresenta um comparativo dos resultados obtidos pelas técnicas na execução de um subgrupo de 11 instâncias. Esse subgrupo contém apenas as instâncias que levaram um tempo superior a 1 segundo para serem executadas, considerando o Algoritmo de Bron-Kerbosch **BK(v4)**.

Tabela 3: Resultados obtidos para as instâncias MIPLIB

Instância	BK(v1)	BK(v2)	BK(v3)	BK(v4)	OSTERGARD	BT
eilB101_cut_21	1,865	4,462	1,473	1,035	TLE	TLE
eilB101_cut_27	3,996	10,216	2,757	1,716	TLE	TLE
eilB101_cut_28	4,484	8,939	2,645	2,190	TLE	TLE
eilB101_cut_29	4,363	11,592	3,510	1,370	TLE	TLE
eilB101_cut_30	8,055	16,607	4,812	4,163	TLE	TLE
eilB101_cut_31	14,310	35,269	9,318	4,024	TLE	TLE
eilB101_cut_32	13,773	23,205	6,918	3,426	TLE	TLE
eilB101_cut_33	19,230	40,400	11,157	11,032	TLE	TLE
eilB101_cut_34	37,177	TLE	18,122	6,134	TLE	TLE
eilB101_cut_35	22,349	59,037	14,993	5,469	TLE	TLE
eilB101_cut_36	16,333	50,042	13,482	11,763	TLE	TLE

Os resultados obtidos para o conjunto **MIPLIB** comprovam a eficiência do Algoritmo de Bron-Kerbosch, principalmente depois das melhorias propostas: A título de comparação, o Algoritmo de Bron-Kerbosch **BK(v1)** despendiu 354,27 segundos para a execução das 399 instâncias do conjunto, ao passo que o Algoritmo de Bron-Kerbosch **BK(v4)** despendiu apenas 65,25 segundos. Quando a análise é feita no desempenho do Algoritmo de Ostergard e da Heurística Busca Tabu com Multi-Vizinhanças, pode-se perceber que as adaptações realizadas nas técnicas para refletir o PECPL, levaram a uma queda considerável de desempenho.

5.3. INRC

O conjunto INRC apresenta um total de 3804 instâncias, as quais variam o número de vértices entre 174 e 3660, enquanto que o número de arestas varia entre 255 e 19421. A Tabela 4 apresenta um resumo comparativo do tempo total (em segundos) despendido para a execução de todas as instâncias do conjunto, utilizando cada uma das técnicas de resolução implementadas. A tabela informa também qual foi o menor (**TMin**) e maior (**TMax**) tempo despendido na execução

das instâncias do conjunto. Por fim, a última coluna da tabela apresenta a quantidade de instâncias que excederam o tempo limite de execução (**TLE**).

Tabela 4: Sumário dos resultados obtidos para as instâncias INRC

Algoritmo	Tempo total	TMax	TMin	TLE
BK(v1)	257,00	1,155	0,001	0
BK(v2)	160,13	0,517	0,001	0
BK(v3)	58,02	0,189	0,001	0
BK(v4)	54,17	0,196	0,001	0
OSTERGARD	61535,29	TLE	0,001	921
BT	72316,95	TLE	0,001	376

Dentre os algoritmos desenvolvidos, o Algoritmo de Bron-Kerbosch **BK(v4)** obteve os melhores resultados, ao conseguir encontrar a solução ótima para todas as instâncias do conjunto INRC, com o menor tempo de uso de CPU. A instância mais crítica foi executada em um tempo irrisório de 0,196 segundos. Considerando o Algoritmo de Ostergard e a Heurística Busca Tabu com Multi-Vizinhanças, 921 e 376 instâncias não encontraram a solução ótima dentro do tempo limite de execução, respectivamente. A Tabela 5 apresenta um comparativo dos resultados obtidos pelas técnicas na execução de um subgrupo de 13 instâncias. Esse subgrupo contém apenas as instâncias que levaram um tempo superior a 0,170 segundos para serem executadas, considerando o Algoritmo de Bron-Kerbosch **BK(v4)**.

Tabela 5: Resultados obtidos para as instâncias INRC

Instância	BK(v1)	BK(v2)	BK(v3)	BK(v4)	OSTERGARD	BT
long_hidden_02_cut_15	0,786	0,380	0,158	0,173	0,386	12,141
long_hidden_02_cut_16	0,832	0,414	0,173	0,187	0,312	18,940
long_hidden_02_cut_18	0,449	0,446	0,189	0,196	1,067	15,057
long_hidden_03_cut_15	0,755	0,454	0,166	0,187	0,152	13,906
long_hidden_03_cut_16	0,742	0,452	0,163	0,182	0,146	15,038
long_hidden_03_cut_17	0,767	0,456	0,167	0,185	0,180	8,904
long_hidden_03_cut_18	0,757	0,456	0,165	0,184	0,171	15,150
long_hidden_05_cut_16	0,828	0,425	0,178	0,182	0,303	11,035
long_hidden_05_cut_17	0,765	0,401	0,160	0,172	0,231	10,946
long_hidden_05_cut_18	0,801	0,446	0,176	0,192	0,305	9,944
long_late_03_cut_13	0,688	0,397	0,148	0,173	0,123	25,130
long_late_03_cut_17	0,752	0,433	0,162	0,182	0,166	16,916
long_late_03_cut_18	0,754	0,430	0,161	0,178	0,166	10,108

Os resultados obtidos para o conjunto **INRC** comprovam a eficiência do Algoritmo de Bron-Kerbosch, principamente depois das melhorias propostas: A título de comparação, o Algoritmo de Bron-Kerbosch **BK(v1)** despendiu 257,00 segundos para a execução das 3804 instâncias do conjunto, ao passo que o Algoritmo de Bron-Kerbosch **BK(v4)** despendiu apenas 54,17 segundos. O Algoritmo de Ostergard e a Heurística Busca Tabu com Multi-Vizinhanças tiveram um ruim desempenho, sendo que, considerando os resultados obtidos com o uso da 1ª técnica, cerca de 25% do total de instâncias não foram resolvidas em sua otimalidade durante o tempo limite de execução (**TLE**).

5.4. Telebus

O conjunto Telebus apresenta um total de 3085 instâncias, as quais variam o número de vértices entre 13 e 1082, enquanto que o número de arestas varia entre 14 e 18323. A Tabela 6

apresenta um resumo comparativo do tempo total (em segundos) despendido para a execução de todas as instâncias do conjunto, utilizando cada uma das técnicas de resolução implementadas. A tabela informa também qual foi o menor (**TMin**) e maior (**TMax**) tempo despendido na execução das instâncias do conjunto. Por fim, a última coluna da tabela apresenta a quantidade de instâncias que excederam o tempo limite de execução (**TLE**).

Tabela 6: Sumário dos resultados obtidos para as instâncias Telebus

Algoritmo	**Tempo total**	**TMax**	**TMin**	**TLE**
BK(v1)	193,46	0,218	0,001	0
BK(v2)	224,59	0,254	0,001	0
BK(v3)	44,39	0,044	0,001	0
BK(v4)	39,17	0,047	0,001	0
OSTERGARD	69886,86	TLE	0,001	888
BT	41990,47	TLE	0,001	223

Dentre os algoritmos desenvolvidos, o Algoritmo de Bron-Kerbosch **BK(v4)** obteve os melhores resultados, ao conseguir encontrar a solução ótima para todas as instâncias do conjunto Telebus, utilizando um menor tempo de CPU. A instância mais crítica foi executada em um tempo irrisório de 0,047 segundos. Considerando o Algoritmo de Ostergard e a Heurística Busca Tabu com Multi-Vizinhanças, 888 e 223 instâncias não encontraram a solução ótima dentro do tempo limite de execução, respectivamente. A Tabela 7 apresenta um comparativo dos resultados obtidos pelas técnicas na execução de um subgrupo de 5 instâncias. Esse subgrupo contém apenas as instâncias que levaram um tempo superior a 0,037 segundos para serem executadas, considerando o Algoritmo de Bron-Kerbosch **BK(v4)**.

Tabela 7: Resultados obtidos para as instâncias Telebus

Instância	**BK(v1)**	**BK(v2)**	**BK(v3)**	**BK(v4)**	**OSTERGARD**	**BT**
t0415_cut_1	0,079	0,071	0,017	0,047	0,014	TLE
t0418_cut_29	0,211	0,246	0,044	0,038	0,585	31,851
t0418_cut_36	0,213	0,251	0,044	0,038	TLE	21,426
t0418_cut_37	0,214	0,248	0,043	0,038	15,937	24,287
t0418_cut_66	0,210	0,248	0,043	0,038	TLE	13,375

Os resultados obtidos para o conjunto **Telebus** comprovam a eficiência do Algoritmo de Bron-Kerbosch, principalmente depois das melhorias propostas: A título de comparação, o Algoritmo de Bron-Kerbosch **BK(v1)** despendiu 193,46 segundos para a execução das 3085 instâncias do conjunto, ao passo que o Algoritmo de Bron-Kerbosch **BK(v4)** despendiu apenas 39,17 segundos. Tempos de computação proibitivos foram despendidos na execução do Algoritmo de Ostergard e da Heurística Busca Tabu com Multi-Vizinhanças, sendo que esta última obteve melhores resultados quando comparados.

5.5. Uchoa

O conjunto Uchoa apresenta um total de 4 instâncias. As características de cada instância são apresentadas na Tabela 8. A 1ª coluna da tabela apresenta o nome da instância. As informações do grafo (instância) são representadas pela quantidade de vértices ($|\mathbf{V}|$) e pela quantidade de arestas ($|\mathbf{A}|$). Outras informações pertinentes ao problema são mostradas na tabela, tais como o **Menor Grau**, **Maior Grau**, **Menor Peso** e o **Maior Peso** dentre os vértices do grafo.

A Tabela 9 apresenta um comparativo dos resultados obtidos, pelas técnicas implementadas no trabalho, na execução das quatro instâncias do conjunto Uchoa. O tempo computacional gasto está expresso em segundos.

Tabela 8: Características das instâncias do grupo Uchoa

Instância	|V|	|A|	Menor Grau	Maior Grau	Menor Peso	Maior Peso
pdistuchoa_cut_1	500	26314	14	320	500	500
pdistuchoa_cut_2	310	5534	5	81	1	746
pdistuchoa_cut_3	371	9929	7	114	4	788
pdistuchoa_cut_4	428	15430	7	172	2	797

Tabela 9: Resultados obtidos para as instâncias Uchoa

Instância	BK(v1)	BK(v2)	BK(v3)	BK(v4)	OSTERGARD	BT
pdistuchoa_cut_1	TLE	TLE	23,977	18,379	TLE	TLE
pdistuchoa_cut_2	0,109	0,073	0,033	0,029	0,598	TLE
pdistuchoa_cut_3	0,436	0,412	0,129	0,104	2,930	TLE
pdistuchoa_cut_4	1,217	1,633	0,360	0,245	9,334	TLE
Tempo total	61,763	62,137	24,499	18,757	72,862	240,000

A instância **pdistuchoa_cut_1**, que é constituída de um grafo contendo 500 vértices e 26314 arestas, levou cerca de 18 segundos para sua execução usando o Algoritmo de Bron-Kerbosch **BK(v4)**. Utilizando-se do Algoritmo de Bron-Kerbosch **BK(v3)** foram necessários aproximadamente 24 segundos. As outras técnicas não encontraram a solução ótima dentro do tempo limite de execução (TLE). Esta instância foi considerada a mais crítica dentre o total de 7292 instâncias utilizadas nos experimentos computacionais, visto que utilizou-se de um maior tempo de processamento.

6. Conclusões

O Problema da Detecção de Cliques com Peso acima de um Limiar foi apresentado e discutido no presente trabalho. Neste problema, o objetivo é encontrar todos os cliques com peso acima de um limiar, previamente informado. Foram utilizadas 3 adaptações de algoritmos clássicos da literatura para a resolução do problema em questão. A fim de se obter uma comparação entre a eficiência das técnicas implementadas, um total de 7292 instâncias, provenientes de quatro conjuntos de problemas de Programação Inteira, foram usadas.

Um tempo limite de 1 minuto foi estipulado como critério de parada para a execução de cada instância. O algoritmo de Bron-Kerbosch adaptado (BKv4) obteve resultados consideravelmente melhores no que se diz respeito ao tempo computacional despendido, quando comparado às outras técnicas implementadas no trabalho. O algoritmo obteve a solução ótima para todas as instâncias, dentro do tempo limite de execução (TLE) pré-estabelecido. Credita-se esse bom desempenho à utilização adequada de estruturas para a representação dos conjuntos pertencentes ao algoritmo.

Uma tentativa de paralelizar o algoritmo Bron-Kerbosch adaptado (BKv4) foi realizada, através da divisão do grafo em subgrafos conexos, com o objetivo de obter vários conjuntos independentes, e então utilizar várias *threads* para a resolução de cada conjunto. Infelizmente, todas as instâncias críticas (tempo de execução superior a 1 segundo) possuem um grafo conexo, o que impossibilitou melhorias no tempo de execução das mesmas. Nesse sentido, como trabalho futuro, é sugerida a paralelização do algoritmo Bron-Kerbosch. Além disso, como pesquisas futuras, também são sugeridas: (i) Paralelização das outras técnicas desenvolvidas no trabalho; (ii) Implementação de outros algoritmos disponíveis na literatura; (iii) Geração de instâncias maiores que as utilizadas no presente trabalho; (iv) Melhorias no Algoritmo de Ostergard adaptado e; (v) Melhorias na heurística Busca Tabu com Multi-Vizinhanças, principalmente na parametrização usada.

Referências

Borndörfer, R., Grötschel, M., Klostermeier, F. e Küttner, C. (1999), Telebus Berlin: Vehicle scheduling in a dial-a-ride system, *Proceedings of the 7th International Workshop on Computer-Aided Transit Scheduling, Springer* Verlag: Berlin, 471(1), 391-422.

Brito, S. S. e Santos, H. G. (2011), Pivoteamento no Algoritmo Bron-Kerbosch para a Detecção de Cliques com Peso Máximo, *XLIII Simpósio Brasileiro de Pesquisa Operacional*.

Brito, S. S. (2015), Grafo de Conflitos: Construção e Aplicações em Problemas de Programação Inteira, *Dissertação (Mestrado em Ciência da Computação)* – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Ouro Preto, Ouro Preto, Minas Gerais.

Bron, C. e Kerbosch, J. (1973), Algorithm 457: finding all cliques of an undirected graph, *Commun. ACM*, 16:575-577.

Burke, E., Marecek, K., Parkes, A. J., Rudová, H. (2011), A branch-and-cut procedure for the udine course timetabling problem, *Annals of Operations Research*, 1-17.

Chvátal, V. (1973), Edmonds polytopes and a hierarchy of combinatorial problems, *Discrete Mathematics*, 4:305-337.

Coll, P., Marenco, J., Méndez Diaz, I. e Zabala, P. (2002), Facets of the graph coloring polytope, *Annals of Operations Research*, 116:79-90.

Debroni, J., Eblen, J. D., Langston, M. A., Myrvold, W., Shor, P. e Weerarupage, D. (2011), A complete resolution of the Keller maximum clique problem, *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, 129–135.

Fischetti, M., Lodi, A. (2007), Optimizing over the first Chvàtal closure, *Mathematical Programming B*, 110(1):3-20.

Frota, Y., Campêlo, M. e Corrêa, R. (2004), Cliques, holes and the vertex coloring polytope, *Information Processing Letters*, 89:159-164.

Goldschmidt, O., Hochbaum, D. S., Hurkens, C., Tu, G. (1996), Approximation Algorithms for the k-Clique Covering Problem, *SIAM Journal on Discrete Mathematics*, 9(3), 492-509.

Haspeslagh, S., De Causmaecker, P., Schaerf, A. e Stølevik, M. (2014), The First International Nurse Rostering Competition 2010, *Annals of Operations Research*, 218 (1), 221-236.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D. E. e Wolter, K. (2011), MIPLIB 2010, *Mathematical Programming Computation*, 3(2), 103-163.

Kumlander, D. (2008), On importance of a special sorting in the maximum weight clique algorithm based on colour classes, *Proc. of the second international conference on Modeling, Computation and Optimization in Information Systems and Management Sciences Communications in Computer and Information Science*, 14:165-174

Ostergard, P. R. J. (2001), A new algorithm for the maximum-weight clique problem, *Nordic J. of Computing*, 8:424-436.

Ostergard, P. R. J. (2002), A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics*, 120:197-207.

Santos, H. G., Toffolo, T. A., Gomes, R. A. e Ribas, S. (2014), Integer Programming Techniques for the Nurse Rostering Problem, *Annals of Operations Research*, 1-27.

Tomita, E., Tanaka, A., Takahashi, H. (2006), The worst-case time complexity for generating all maximal cliques and computational experiments, *Theoretical Computer Science*, 363(1), 28-42.

Vassilevska, V. (2008), Efficient algorithms for clique problems, *Information Processing Letters*, 109(2), 254-257.

Wu, Q., Hao, J. K. e Glover, F. (2012), Multi-neighborhood tabu search for the maximum weight clique problem, *Springer*, 196:611-634.

Yamaguchi, K., Masuda, S. (2008), A new exact algorithm for the maximum weight clique problem, *Proc. of the 23rd International Technical Conference on CircuitsSystems, Computers and Communications*, 317-320.