UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

LUCAS HENRIQUE DE SOUSA CÂNDIDO

UM ALGORITMO BIT-PARALELO PARA O PROBLEMA DA CLIQUE MÁXIMA

QUIXADÁ – CEARÁ

2016

LUCAS HENRIQUE DE SOUSA CÂNDIDO

UM ALGORITMO BIT-PARALELO PARA O PROBLEMA DA CLIQUE MÁXIMA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Orientador: Prof. Dr. Wladimir Araújo Tavares

QUIXADÁ – CEARÁ

2016

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)

C223a Cândido, Lucas Henrique de Sousa.
Um algoritmo bit-paralelo para o problema da clique máxima / Lucas Henrique de Sousa Cândido. – 2016.
45 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2016.
Orientação: Prof. Dr. Wladimir Araújo Tavares.

1. Problema da clique máxima. 2. Coloração de grafos. 3. Branch-and-bound. 4. NP-difícil. I. Título.
CDD 005

LUCAS HENRIQUE DE SOUSA CÂNDIDO

UM ALGORITMO BIT-PARALELO PARA O PROBLEMA DA CLIQUE MÁXIMA

Monografia apresentada no curso de Sistemas de Informação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação. Área de concentração: Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Wladimir Araújo Tavares (Orientador)
Campus Quixadá
Universidade Federal do Ceará – UFC

Prof. Dr. Fábio Carlos Sousa Dias
Campus Quixadá
Universidade Federal do Ceará - UFC

Prof. Me. Paulo Henrique Macedo de Araújo
Campus Quixadá
Universidade Federal do Ceará - UFC

Aos meus pais, família, professores e amigos.

AGRADECIMENTOS

À CAPES, pelo apoio financeiro com a manutenção da bolsa de auxílio.

Ao Prof. Dr. Wladimir Araújo Tavares, pela paciência e excelente orientação.

Aos professores participantes da banca examinadora Fábio Carlos Sousa Dias e Paulo Henrique Macedo de Araújo pelo tempo, pelas valiosas colaborações e sugestões.

Aos meus amigos André Davys e Letícia Saraiva por terem cedido seus notebooks.

A Marisa Silva por passar noites conversando comigo quando pensava em não conseguir e por ter me ajudado a corrigir meu tcc.

Aos meus colegas de turma pelo apoio, palas reflexxões, críticas e sugestões recebidas.

Aos meus pais por acreditarem na minha capacidade.

“Só sei que nada sei”

(Sócrates)

RESUMO

Nesta monografia, apresentamos um algoritmo de branch-and-bound que encontra uma clique máxima em um grafo. O algoritmo combina técnicas já empregadas com sucesso, além de utilizar uma nova proposta de heurística de coloração fracionária, proposta pelos autores, como procedimento de limite superior e estratégia de ramificação, fazendo uso de operações bit-paralelas. Experimentos computacionais mostram que a coloração proposta apresenta uma grande redução na árvore de busca e um bom desempenho em instâncias de alta densidade.

Palavras-chave: Problema da clique máxima. Coloração de grafos. Branch-and-bound. NP-difícil.

ABSTRACT

In this paper we present a branch-and-bound algorithm to find the maximum clique in a graph. The algorithm combines successfully used techniques and also makes use of a new heuristic proposal for fractional coloring create by the authors, as a procedure of upper bound and branching strategy, making use of bit-parallel operations. Computing experiments show that the coloring proposal presents a large reduction in the search tree and a good performance in high density instances.

Keywords: Maximum Clique Problem. Coloring. Branch-And-Bound. NP-hard

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

PCM	Problema da Clique Máxima
HCI	Heurística de Coloração Inteira
HCF	Heurística de Coloração Fracionária

LISTA DE SÍMBOLOS

G	Grafo simples unidirecional, onde $G = (V, E)$
V	Conjunto de vértices de um grafo
E	Conjunto de arestas de um grafo
$\Gamma(v)$	Conjunto de vértices, em que $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$, ou seja, a vizinhaça de v
$\omega(G)$	Tamanho da clique máxima no grafo G
$G[Q]$	Seja Q um conjunto de vértices, então $G(Q)$ representa o sugrafo de G, em que $G(Q) = (Q, E \cap QxQ)$

SUMÁRIO

1 INTRODUÇÃO

Uma clique é um subgrafo em que todos os pares de vértices são adjacentes entre si. O problema da clique máxima consiste em determinar uma clique com a máxima carnalidade. Encontrar uma clique máxima é um problema NP-difícil, sendo assim, é muito complexo se obter eficientemente uma solução (GARY; JOHNSON, 1979; KARP, 1972). Além disso, o problema da clique máxima é inaproximável para qualquer fator $O(n^{1-\varepsilon})$ para qualquer $\varepsilon > 0$, a menos que $P = NP$ (HÅSTAD, 1996). Encontrar a clique máxima em um grafo é um dos problemas NP-difícil mais importantes na matemática discreta e no campo teórico da ciência computação, e tem sido estudado por vários pesquisadores (TOMITA; SEKI, 2003).

O problema da clique máxima (PCM) possui muitas aplicações práticas em diversas áreas. Wu e Hao (2015), Balas e Yu (1986) e Bomze et al. (1999) apresentam algumas delas, que são: bioinformática e quimioinformática (MALOD-DOGNIN; ANDONOV; YANEV, 2010; RAVETTI; MOSCATO, 2008), teoria da decodificação (ETZION; OSTERGARD, 1998), economia (BOGINSKI; BUTENKO; PARDALOS, 2006), problemas de alocação (CARTER; LAPORTE; LEE, 1996; WEIDE; RYAN; EHRGOTT, 2010), redes financeiras (BOGINSKI; BUTENKO; PARDALOS, 2006), localização (BROTCORNE; LAPORTE; SEMET, 2002), agendamento (DORNDORF; JAEHN; PESCH, 2008), análise de transmissão de sinal (CHEN; ZHAI; FANG, 2010), análise de redes sociais (BALASUNDARAM; BUTENKO; HICKS, 2011; PATTILLO; YOUSSEF; BUTENKO, 2012), redes wireless e telecomunicações (BALASUNDARAM; BUTENKO, 2006; JAIN et al., 2005), visão computacional e reconhecimento de padrões (BOMZE et al., 1999), design experimental e recuperação de informações (BALAS; YU, 1986). Por tal aplicabilidade se torna necessário o desenvolvimento de algoritmos exatos, que buscam a solução ótima, para estes problemas com tempo de execução satisfatório.

Dada a sua importância teórica e relevância prática, um esforço considerável tem sido dedicado ao politopo e desigualdades válidas associada ao problema da clique máxima em programação discreta. Por um lado, os métodos exatos mais eficazes são os métodos de branch-and-bound combinatórios. Resultados de experimentos computacionais mostram que os algoritmos de branch-and-bound combinatórios superam outros algoritmos baseados em programação matemática. Tomita e Seki (2003) desenvolveram um algoritmo, chamado de MCQ, que é comparado com outros algoritmos de programação matemática (BALAS; YU, 1986; BOURJOLLY P. GILL; MERCURE, 1996).

No entanto, os métodos exatos requerem um tempo computacional proibitivo em casos gerais e muitas vezes são aplicáveis apenas em problemas de tamanhos limitados. Por outro lado, para lidar com problemas cujas soluções ótimas não podem ser encontradas dentro de um prazo razoável, vários algoritmos heurísticos e metaheurísticos foram desenvolvidos com o objetivo de fornecer soluções sub-ótimas tão boas quanto possível para problemas grandes em tempo aceitável. É claro que os métodos exatos e heurísticos constituem duas soluções complementares que podem ser aplicadas para enfrentar situações diferentes e cumprir objetivos diferentes (WU; HAO, 2015).

Nesta monografia, apresentamos uma heurística de coloração e um algoritmo branch-and-bound para o problema da clique máxima que combina, modifica e/ou aperfeiçoa alguns dos ingredientes utilizados nos algoritmos existentes, como segue:

- Uma ordenação inicial dos vértices através da reconstrução da matriz de adjacência em ordem decrescente de grau;
- Uso da coloração proposta, que é uma adaptação da heurística de coloração fracionária apresentada em Balas e Xue (1996);
- Estratégia de ramificação inspirada em Balas e Yu (1986), chamada de ramificação em largura, utilizando a adaptação da heurística de coloração fracionária;
- A utilização de vetores de bits que possibilita a redução do uso de memória e o aproveitamento do paralelismo de bits (SEGUNDO et al., 2010).

Na seção 2, são mostrados os trabalhos relacionados. Na seção 3, apresentamos a fundamentação teórica. Na seção 4, apresentamos os algoritmos desenvolvido, onde finalmente apresentamos nossa proposta. Na seção 5, apresentamos nossos resultados. E na seção 6, apresentamos nossas conclusões e trabalhos futuros.

2 TRABALHOS RELACIONADOS

Balas e Xue (1996) propõem um algoritmo branch-and-bound que utiliza uma heurística de coloração fracionária para geração de limites superiores para a clique máxima, além disso, utilizam a coloração inteira como estratégia de ramificação. Realizando extensivos testes computacionais Balas e Xue (1996) verificaram que a substituição de procedimentos padrões de limite superior pela heurística de coloração fracionária apresenta um fator de diminuição de quase dois na árvore de busca.

No algoritmo MCQ (TOMITA; SEKI, 2003), uma heurística de coloração inteira gulosa é utilizada como procedimento de limite superior e como estratégia de ramificação.

O algoritmo proposto por Tomita et al. (2010), chamado de MCS, diferente do MCQ ainda aplica um algoritmo de re-coloração ao final de cada iteração do algoritmo de coloração inteira nos vértices que possuem limite superior maior que o tamanho da clique máxima atual. O algoritmo de re-coloração tenta aumentar o números de vértices coloridos utilizando um certo número de cores estabelecido em cada subproblema, reduzindo assim o fator de ramificação do algoritmo. Nota-se que em Tomita e Seki (2003), a utilização da coloração inteira como estratégia de ramificação diminui consideravelmente o fator de ramificação, este fato pode ser verificado na redução drástica do número de subproblema. Em Tomita et al. (2010), desenvolve-se uma estratégia de ramificação, baseada em uma ordem inicial fixa dos vértices, para a heurística de coloração gulosa, e no algoritmo de re-coloração, conseguindo reduzir ainda mais o fator de ramificação.

Os trabalhos anteriores apresentam métodos que reduzem o número de subproblemas enumerados durante a busca, contudo, outro fator importante a ser considerado no problema do clique máximo, são as várias operações de conjunto para encontrar a clique máximo em um grafo, estas operações se não bem implementadas podem consumir bastante tempo de processamento. Segundo et al. (2010) apresentam um algoritmo chamado de BBMC para busca da clique máximo utilizando vetores de bits. Através da implementação de vetores de bits os autores conseguiram grande redução do custo computacional. Por este motivo, utilizamos vetores de bits em nossa proposta.

Em Balas e Xue (1996), eles propõe a utilização da heurística de coloração fracionária como procedimento de limite superior. Porém, a coloração fracionária é totalmente inútil para a definição da estratégia de ramificação. Esse problema é superado definindo a estratégia de ramificação usando a coloração inteira inicial. Neste trabalho, fazemos as adaptações necessárias

para que a coloração fracionária possa ser utilizada como estratégia de ramificação.

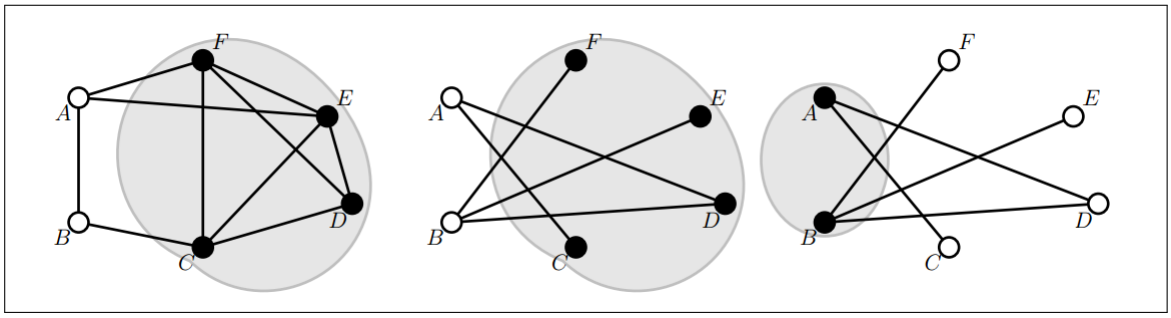
Esta monografia propõe um algoritmo que combina a coloração inteira com uma adaptação da coloração fracionária, porém a heurística de coloração fracionária será utilizada aqui para definição do conjunto de ramificação, assim como a coloração inteira é aplicada nos algoritmos MCQ e MCS. Além de disso, a solução será implementada utilizando o bit parelelismo. Comparamos os algoritmos bit-paralelos Segundo, Rodríguez-Losada e Jiménez (2011) e Segundo et al. (2013) que utilizam a coloração inteira e a recoloração, respectivamente, com o algoritmo proposto nesta monografia. Recorremos a estes algoritmos, no lugar do MCQ e MCR, por serem algoritmos bit-paralelos, assim a comparação entre eles se torna mais justa.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Problema

O problema da clique máxima consiste em determinar uma clique (isto é, um subgrafo completo) com a máxima carnalidade. Este problema é notável por sua capacidade de modelar outros problemas combinatórios e muitas aplicações do mundo real. É um dos problemas NP-difícil mais estudados, existem muitos algoritmos presentes na literatura e vários novos métodos sendo continuamente propostos (TOMITA; SEKI, 2003).

Figura 1 – Ilustração da relação entre a clique máxima, conjunto independente máximo e cobertura mínima de vértices. Dado o grafo G com $V = \{A,B,C,D,E,F\}$ (a esquerda) e o seu complemento $\overline{G}$ (no meio e a direita), o conjunto de vértices $\{C,D,E,F\}$ é a clique máxima em G e é um conjunto independente máximo em $\overline{G}$, enquanto $\{A,B\} = V \setminus \{C,D,E,F\}$ é uma cobertura mínima de vértices em $\overline{G}$.



Fonte: Wu e Hao (2015)

O PCM é estritamente equivalente a dois outros problemas bem conhecidos de otimização combinatória: o problema do conjunto independente máximo (PCIM) e o problema de cobertura mínima de vértices (PCMV). Dado $G = (V,E)$, um conjunto independente I de G é um subconjunto de V tal que cada dois vértices de I não são conectados por uma aresta, isto é, $\forall u,v \in I, (u,v) \notin E$. O PCI consiste em determinar o conjunto independente de maior cardinalidade. Uma cobertura de vértice V' de G é um subconjunto de V, tal que toda aresta $(u,v) \in E$ tem pelos menos um ponto final em V'. O PCMV consiste em encontrar a cobertura de vértice com a menor cardinalidade (WU; HAO, 2015).

O complemento do grafo G é representado por $\overline{G} = (V,\overline{E})$, onde o conjunto $\overline{E} = \{(u,v) \mid (u,v) \notin E\}$. Seja $W \subset V$, o subgrafo induzido por W em G, então $G[W] = (W,(W \times W) \cap E)$. Para qualquer $W \subset V$, Balas e Yu (1986) afirmam que as três definições são equivalentes

e facilmente verificáveis:

1. Se W é máximo clique em G.
2. Então, W é um conjunto independente máximo em $\overline{G}$.
3. E $V \setminus W$ é uma cobertura de vértices mínima em $\overline{G}$.

Desse modo, os três problemas a seguir são equivalentes

1. Buscar a clique máximo em G,
2. Buscar o conjunto independente máximo em $\overline{G}$ e
3. Buscar a cobertura de vértices mínima em $\overline{G}$

A Figura 1 apresenta a relação entre os três problemas.

Esta monografia é dedicada ao estudo e a contribuição de novos resultados para o problema da clique máxima.

3.2 Bit Paralelismo

O bit paralelismos consiste no uso de um vetor de bits (*bit field*, *bitmap*, *bitboard*, *bit string*, etc), que é uma estrutura de dados dimensional que armazena bits individuais de forma compacta, e é eficaz na exploração do bit-paralelismo em nível de hardware para realizar rapidamente as operações, bem como reduzir o espaço de armazenamento (SEGUNDO; RODRÍGUEZ-LOSADA; JIMÉNEZ, 2011).

Dado o conjunto A pertencente ao universo $\sum = \{0, 1, \cdots, u-1\}$, cujo o tamanho u é conhecido e não é abusivamente grande. Podemos representar o conjunto A utilizando vetor de bits, onde o bit na posição i é um 1 se $i \in A$, caso contrário, é 0. Seja a palavra de tamanho h e o vetor de bits B, de tamanho $\lceil u/h \rceil$, representando o conjunto A, temos que a palavra $B[j]$ contém os elementos $jh, jh+1, \cdots, (j+1)h-1$ do universo $\sum$.

Utilizamos a linguagem C para implementar o vetor de bits, dessa forma uma palavra é representada por um inteiro não negativo, e o tamanho da palavra é dado pelo número de bits do maior inteiro suportado pelos compiladores atuais (tipicamente 64).

O vetor de bits pode ser declarado na linguagem C como apresenta o Algoritmo 1:

Algorithm 1 Representação de um vetor de bits em C

```
1: #define INT_SIZE 64
2: typedef uint64_t word;
3: word B[(tamanho do universo de B) / INT_SIZE + 1]
```

Deste modo, a consulta de um elemento x é feita em duas dimensões. Portanto, para endereçar a palavra correspondente em B, calcula-se a função $high(x) = x/h$. Além disso, o índice do bit em $B[high(x)]$ é dado por $low(x) = x \bmod h$. Desta maneira, as operações

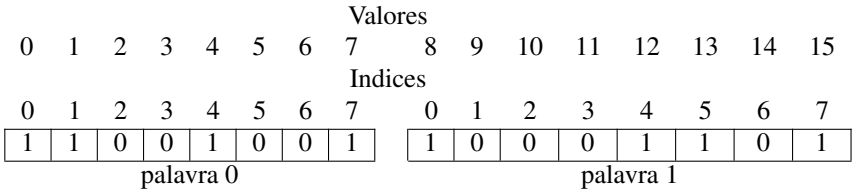
$$\begin{aligned} \text{setbit}(B,x) &= B[high(x)] \mid (1LL << low(x)) \\ \text{resetbit}(B,x) &= B[high(x)] \;\&\sim (1LL << low(x)) \\ \text{test}(B,x) &= B[high(x)] \;\& \;(1LL << low(x)) \end{aligned}$$

representam a inserção, remoção e o teste de existência, respectivamente, do elemento x em B. Além dessas operações, implementamos as seguintes funções:

- index(i, j): retorna o valor do elemento que esta na posição i, j.
- LSB(B, i, n): retorna a posição do menor elemento em $\{h*i,\ldots,n-1\} \cap B$.
- COUNT(B, n): retorna o número de elementos em B.

Seja $A = \{0,1,4,7,8,12,13,15\}$ pertencente ao universo $\Sigma = \{0,1,\ldots,15\}$ e um palavra de tamanho 8, então a representação B, em vetor de bits, de A possui $\lceil 16/8 \rceil = 2$ palavras. A representação gráfica de B pode ser vista na Figura 2.

Figura 2 – Representação gráfica de B



Fonte: Elaborado pelo autor

Logo, a verificação do elemento 12 é feita seguinte maneira, primeiro é verificada em qual palavra ele se encontra, então $high(12) = 12/8 = 1$. Agora, sabemos que 12 está na palavra 1, contudo ainda não sabemos em qual bit ele está, por isso utilizamos a função low, portanto $low(12) = 12 \bmod 8 = 4$. Pronto, sabemos que 12 se encontra na palavra 1 e no bit 4, se o valor neste posição for 1, então $12 \in A$, caso contrário $12 \notin A$.

Uma das vantagens de se utilizar vetor de bits se dá ao fato dos conjuntos serem representados como agrupamentos de bits, assim operações lógicas podem ser utilizadas para simular as operações de conjuntos. Seja os conjuntos A_1 e A_2 e suas representações como vetores de bits B_1 e B_2, podemos simular as operações de conjuntos como descritas a seguir:

$A_1 \cap A_2 \equiv B_1 \& B_2$

$A_1 \cup A_2 \equiv B_1 | B_2$

$\overline{A_1} \equiv \sim B_1$

$A_1 \setminus A_2 \equiv B_1 \& \sim B_2$

$A_1 \oplus A_2 \equiv B_1 \wedge B_2$

Portanto, operações sobre conjuntos podem ser processadas em blocos de informações (palavras, no caso desta monografia, de 64 bits) ganhando grande aumento de processamento. Segundo et al. (2007) mostram que um modelo de bits adequado pode causar uma grande redução em relação ao tamanho da palavra na CPU. O bit-paralelismo é utilizado nesta monografia, devido ao fato do PCM envolver muitas operações de conjuntos, seu uso faz com que haja um ganho de desempenho ao realizar tais operações.

3.3 Algoritmos Branch-and-Bound

Em essência, os métodos de branch-and-bound são esquemas enumerativos para resolver problemas de otimização. A utilidade do método deriva do fato de que, em geral, apenas uma pequena fração das possíveis soluções devem realmente ser enumerados, as soluções restantes devem ser eliminadas a partir da consideração de limites que estabelecem que tais soluções não podem ser ótimas (MITTEN, 1970).

O nome branch-and-bound surge a partir de duas operações básicas:

- *Branching*, consiste em dividir coleções de conjuntos soluções em subconjuntos.
- *Bounding*, consiste em estabelecer limites sobre os subconjuntos de soluções.

O procedimento branch-and-bound envolve a aplicação recursiva das operações de *branching* e *bounding*, apagando subconjuntos conhecidos que não podem conter uma solução ótima (MITTEN, 1970).

Wu e Hao (2015) apresentam um algoritmo branch-and-bound simples para o PCM, denotado de CP. O Algoritmo 2 contém o pseudocódigo do algoritmo CP.

Algorithm 2 Algoritmo branch-and-bound simples para o PCM

```
1: function CP(C, P)
2:     if |C| > |C*| then
3:         C* = C
4:     if |C| + |P| <= |C*| then return
5:     for all p ∈ P em alguma predeterminada ordem do
6:         P = P \ {p}
7:         C' = C ∪ {p}
8:         P' = P ∩ Γ(p)
9:         CP(C', P')
```

Fonte: Wu e Hao (2015)

O branch-and-bound do algoritmo CP funciona da seguinte maneira: sua operação *branching* consiste em gerar todas as cliques que o vértice p pertence, verificando assim todas as cliques em busca da maior clique. Quando a clique atual mais o número de vértices que podem entrar na clique é menor ou igual a maior clique encontrada até o momento, o subproblema é podado, está é sua operação de *bounding*. Contudo, esta operação de *bounding* não é muito boa, utilizar o número de candidatos a uma clique como limite superior poda muito pouco os subproblemas.

Por este motivo, em uma tentativa de aumentar o número de subproblemas podados nós adotamos a seguinte representação: um subproblema para o PCM é descrito pela tripla (C,P,LB), onde LB é o tamanho da maior clique encontrada até o momento, $C = \{v_1,\ldots,v_d\}$ (onde d é a profundidade do subproblema) é a clique atual que está sendo construída pelo subproblema e P é o conjunto de vértices que podem entrar na clique atual, ou seja, $P \subseteq \cap_{i=1}^{d}\Gamma(v_i)$, denominado de conjunto candidato.

Se (C,P,LB) não é podado, um vértice $v \in P$ é selecionado e adicionado à clique atual. Quando o vértice v é adicionado, dizemos que o vértice v foi ramificado. Todas as cliques maximais que o vértice v pertence são enumerados implicitamente pelo subproblema $(C\cup\{v\}, P\cap\Gamma(v), LB)$, depois disso o vértice v é removido de P.

Comumente, em cada subproblema, nós particionamos o conjunto $P = U \cup R$ tal que $\omega(G[U]) \leq LB - \omega(C)$, de modo que apenas os vértices que estão em R são utilizados no processo de ramificação. Os vértices são ramificados em uma ordem definida pela estratégia de ramificação.

Seja $(v_1,\ldots,v_n)$ uma ordem dos elementos de R definida pela estratégia de ramificação. Então os seguintes subproblemas serão gerados:

$$
\begin{array}{lcl}
x_{v_n} = 1 \vee & \Leftrightarrow & (C\cup\{v_n\}, P\cap\Gamma(v_n), LB) \\
(x_{v_n} = 0 \wedge x_{v_{n-1}} = 1) \vee & \Leftrightarrow & (C\cup\{v_{n-1}\}, (P\setminus\{v_n\})\cap\Gamma(v_{n-1}), LB) \\
\vdots & & \vdots \\
(x_{v_n} = 0 \wedge x_{v_{n-1}} = 0 \wedge \ldots \wedge x_{v_1} = 1) \vee & \Leftrightarrow & (C\cup\{v_1\}, (P\setminus\{v_n,\ldots,v_2\})\cap\Gamma(v_1), LB)
\end{array}
$$

Esse esquema de ramificação foi estabelecido primeiramente em Balas e Yu (1986).

A estrutura básica do algoritmo branch-and-bound para o PCM incorporando o esquema de ramificação de Balas e Yu (1986) utilizado nesta monografia está apresentado no algoritmo 4.

Algorithm 3 Estrutura básica do algoritmo branch-and-bound

```
1: function CLIQUE(C, P, LB)
2:     if |C| > LB then
3:         LB = |C|
4:     Particiona P = U ∪ R tal que ω(G[U]) ≤ LB − ω(C)
5:     Seja (v_1, ..., v_n) uma ordem dos vértices de R
6:     for i ← n até 1 do
7:         CLIQUE(C ∪ {v_i}, P ∩ Γ(v_i), LB)
8:         P = P \ {v_i}
```

Fonte: Elaborado pelos autores

Na seção 3.4 apresentamos algumas estratégias de ramificação utilizadas na literatura para definir a ordem dos elementos de R e na seção 4 finalmente será apresentado a nossa estratégia de ramificação.

3.4 Limite Superior

Um limite superior conhecido para o tamanho da clique máxima pode ser obtido através de uma coloração. A coloração consiste em particionar os vértices de um grafo em conjuntos independentes. Seja $C_1, \ldots, C_k$ uma coloração de $G[V]$, sabemos que $\omega(G[V]) <= k$. Nota-se que em cada classe de cor, pode-se escolher no máximo um vértice para compor a clique, assim $\omega(G[V])$ não pode ser maior do que k.

Por este motivo, vários algoritmos branch-and-bound usam coloração por aproximação para obter um limite superior para o tamanho da clique máxima. Uma elaborada coloração pode reduzir muito o espaço de busca. Coloração, no entanto, consome tempo, e torna-se importante escolher uma troca adequada entre o tempo necessário para uma coloração aproximada e a redução do espaço de busca obtido pela mesma (TOMITA; SEKI, 2003).

Basicamente, duas maneiras diferentes são exploradas na literatura para usar a coloração nas operações de *branching* e de *bounding*. A primeira maneira colore o grafo inicial apenas uma vez antes que a rotina comece e usa essa coloração em toda a ramificação. Esta estratégia tem a principal vantagem de executar o algoritmo de coloração apenas uma vez. Entretanto, como o algoritmo da clique manipula muitos e diferentes subgrafos do grafo inicial G, a coloração para G não é necessariamente apropriada para a estimativa desses subgrafos reduzidos. A segunda estratégia aplica diferentes algoritmos de coloração para diferentes nós da árvore de busca. Esta estratégia torna possível obter limites mais próximos da clique máxima dos subgrafos. No entanto, colorir vários grafos pode ser também demorado. Com a finalidade de

economizar tempo de computação, heurística gulosas de coloração normalmente são empregadas (WU; HAO, 2015).

Esta monografia adota a segunda estratégia, pois esta aumenta o número de vértices podados e possui um melhor limite superior. Nesta seção é apresentada algumas heurísticas gulosas de coloração, utilizadas na literatura, que podem ser aplicadas como limites superiores e como estratégias de ramificação para o problema da clique máxima.

3.4.1 Coloração Inteira

Seja (C, P, LB) um subproblema do PCM. A heurística de coloração inteira consiste em atribuir para cada $v \in P$ um inteiro positivo $UB[v]$, chamado de cor, com as seguintes propriedades:

1. Se $(v,u) \in E$, então $UB[v] \neq UB[u]$, e
2. $UB[v] = 1$, ou $UB[v] = i > 1$, logo existem vértices $v_1 \in \Gamma(v), v_2 \in \Gamma(v), \ldots, v_{i-1} \in \Gamma(v)$ em P com $UB[v_1] = 1, UB[v_2] = 2, \ldots, UB[v_{i-1}] = i-1$.

Consequentemente, nós sabemos que

$$\omega(P) \leq Max\{UB[v] | v \in P\}$$

e por isso, se $|C| + Max\{UB[v] | v \in P\} \leq |C_{max}|$ então, nós podemos descartar P (TOMITA; SEKI, 2003). Por este motivo, utilizamos as cores geradas pelas heurísticas de coloração como estratégia de ramificação e como *bounding*.

Algorithm 4 Procedimento da coloração inteira

```
1:  function COLORAÇÃO_INTEIRA(V, R, UB, LB)
2:      U = S = V
3:      i = 1
4:      n = 0
5:      while U ≠ ∅ do
6:          C_i = ∅
7:          while S ≠ ∅ do
8:              selecione o primeiro vértice u ∈ S
9:              C_i = C_i ∪ u
10:             S = S \ {u}
11:             S = S ∩ Γ̄(u)
12:             if i ≥ LB then
13:                 R[n] = u
14:                 UB[u] = i
15:                 n = n + 1
16:         U = U \ C_i
17:         S = U
18:         i = i + 1
19:     return i
```

Fonte: Segundo, Rodríguez-Losada e Jiménez (2011)

O valor $UB[v]$ para $v \in V$ pode ser facilmente atribuído pelo Algoritmo 4 da seguinte maneira: selecionamos um vértice u de S e em seguida adicionamos a primeira classe de cor C_1, assim $UB[u] = 1$, depois retiramos de S todos os vértices que são adjacentes a u e repetimos o processo até que $S = \emptyset$, logo C_1 foi preenchido e $\forall v \in C_1\ UB[v] = 1$. Depois retiramos de U os vértices contidos em C_1, fazemos S receber os vértices ainda não coloridos e avançamos para próxima cor, repetimos esse processo até que todos os vértices estejam coloridos.

Note que apenas os vértices que possuem cor maior que LB são armazenados em R, pois pela afirmação apresentada anteriormente podemos concluir que se $|C| + UB[v] \leq LB$, então o subproblema $(C \cup \{v\}, P \cap \Gamma(v), LB)$ não possui um clique maior do que LB. Isso permite implicitamente mais uma operação *bounding* no algoritmo branch-and-bound.

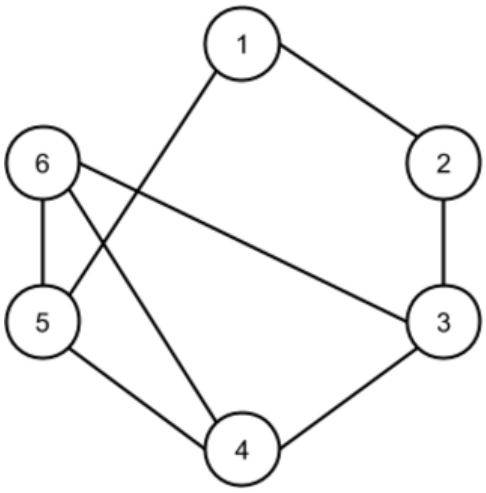
Considere o seguinte exemplo:

Seja (3, 4, 5, 6, 1, 2) a ordem inicial dos vértices (decrescente por grau). A coloração obtida pela heurística de coloração inteira será:

1. $C_1 = \{3, 5\}$
2. $C_2 = \{4, 1\}$
3. $C_3 = \{6, 2\}$

Na tabela seguinte, o valor $UB[v]$ para cada vértice v é mostrado:

Figura 3 – Grafo usado no Exemplo 1



Fonte: Balas e Xue (1996)

Tabela 1 – Tabela com os valores $UB[v]$ para cada vértice v

v	3	4	5	6	1	2
$UB[v]$	1	2	1	3	2	3

Fonte: Produzido pelos autores

Observe que a ordem inicial dos vértices afeta a forma que os vértices são coloridos. Na próxima tabela, o valor de $UB[v]$ é apresentado seguindo uma ordem δ:

Tabela 2 – Tabela com os valores $UB[v]$ para a ordem δ

δ	6	5	4	3	1	2
$UB[v]$	1	2	3	2	4	3

Fonte: Produzido pelos autores

Grande parte dos algoritmos presentes na literatura adotam a ordem descrente dos vértices por grau como ordem inicial. Pois os vértices com um grau elevado possuem muitos vértices adjacentes, assim eles geram muitos subproblemas quando ramificados, então quando a profundidade dos subproblemas é pequena os vértices de maior grau tendem a ficar no conjunto

U, ou seja, não serão ramificados nesses subproblemas. Deste modo, os vértices que possuem grau elevado tendem a serem ramificados quando a profundidade dos subproblemas é maior, onde grande partes dos vértices já terão sido descartados, diminuindo assim o número de subproblemas gerados pelos mesmos.

3.4.2 Coloração Fracionária

Uma heurística de coloração fracionária (HCF) funciona da seguinte forma: suponha que temos uma coloração (inteira) $C_1 = (S_1, \ldots, S_q)$, onde cada S_i é uma classe de cor (conjunto independente). Como C_1 é uma partição de V, tipicamente algumas das classes de cor S_i não são maximais. Suponha que agora podemos aumentar essas classes de cor colorindo o maior número de vértices que pudermos com uma segunda cor entre as q classes de cor. Seja V_2 o conjunto desses vértices que agora pertencem a duas classes de cor. Então a coloração C_1 torna-se, assim, um conjunto de sobreposição em vez de subconjuntos disjuntos, mas o seu tamanho permanece inalterado.

Algorithm 5 Procedimento da coloração fracionária

function COLORAÇÃO_FRACIONARIA(V)
  $t_1^0 = \infty$, $C = \emptyset$, $y = 0$, $k = 1$
  while true **do**
    $U_k = V$
    for all $v \in U_k$, selecione uma classe de cor $S_i \in C$, tal que $S_i \cup \{v\}$ é um conjunto independente **do**
      $S_i = S_i \cup \{v\}$
      $U_k = U_k \setminus \{v\}$
    aplique HCI em $G[U_k]$, seja C_k a coloração encontrada
    if $(|C| + |C_k|)/k < t_1^{k-1}$ **then**
      $t_1^k = (|C| + |C_k|)/k$
      $C = C \cup C_k$
      $k = k + 1$
    else
      $C^* = C$
      $t_1^* = t_1^{k-1}$
      $y_S^* = \begin{cases} 1/(k-1) & para \quad S \in C^*, \\ 0 & para \quad S \notin C^* \end{cases}$
      break
  return $\lfloor t_1^* \rfloor$

Fonte: Balas e Xue (1996) (com adaptação)

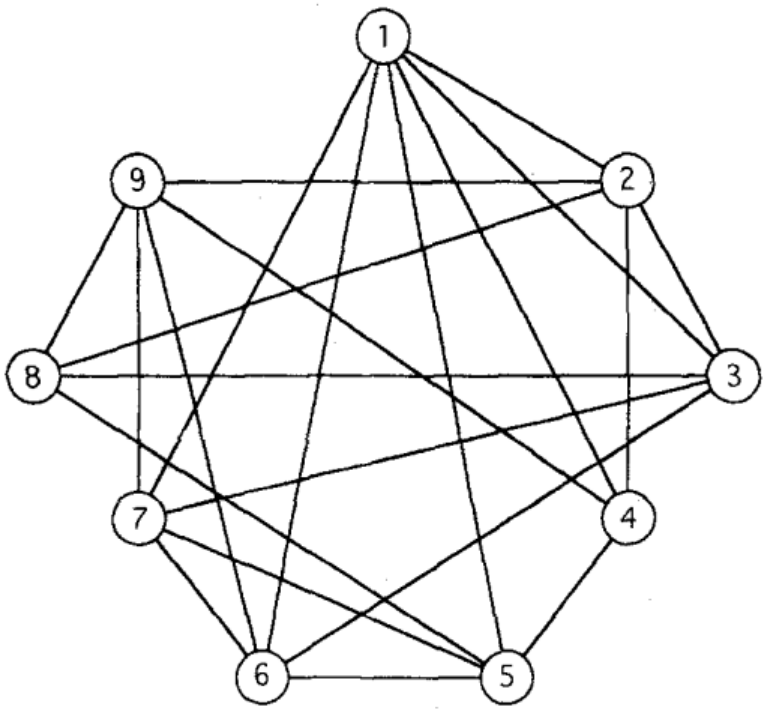
Se agora procuramos uma nova coloração C_2 dos vértices $V \setminus V_2$, em seguida, iremos ter coloridas cada vértice duas vezes, e assim, obtém-se uma coloração fracionária total de

tamanho $(|C_1|+|C_2|)/2$. No entanto, uma vez que C_2 é uma coloração com menos vértices do que $|V|$, então tipicamente $|C_2| < |C_1|$ e, por conseguinte, $(|C_1|+|C_2|)/2 < |C_1|$, isto é, a coloração fracionária é melhor do que C_1. Este processo pode ser repetido utilizando uma terceiro, quarto, ..., k-ésima coloração, desde que melhorias possam ser obtidas (BALAS; XUE, 1996).

As cores são geradas a cada iteração pela HCI. Balas e Xue (1996) denota $C = C_1 \cup C_2 \cup \ldots C_k$ como a coleção de colorações geradas durante o Algoritmo 5. A cada iteração k, Balas e Xue (1996) denota U_k como o conjunto de vértices que ainda não foram coloridos durante a interação, e t_1^k como o valor atual da coloração fracionaria. A provam que $\lfloor t_1^* \rfloor$ é um limite superior para o tamanho da clique máxima em G pode ser encontrada em Balas e Xue (1996). A coloração fracionária é utilizada por Balas e Xue (1996) apenas como limite superior, como estratégia de ramificação é usada a coloração inteira realizada na primeira iteração da HCF.

Considere o seguinte grafo como exemplo:

Figura 4 – Grafo usado no Exemplo 2



Fonte: Balas e Xue (1996)

Quando $k = 1$. $U_1 = V$, como ainda não existe nenhuma coloração o HCF pula o passo 5 e executa o HCI gerando a coloração $C_1 = \{S_1, \ldots, S_5\}$, com $S_1 = \{1,9\}$, $S_2 = \{2,5\}$, $S_3 = \{3,4\}$, $S_4 = \{6,8\}$, $S_5 = \{7\}$. Isto dá o seguinte limite de $t_1^1 = |C| = |C_1| = 5$ e $C = C \cup C_1$.

Quando $k = 2$. O vértice 2 pode ser adicionado ao S_5 e o vértice 4 ao S_4, isto é, $S_4 = \{4,6,8\}$, $S_5 = \{2,7\}$, $U_2 = \{1,3,5,6,7,8,9\}$. O ICH produz $C_2 = \{S_6, S_7, S_8, S_9\}$, com $S_6 = \{1,8\}$, $S_7 = \{3,5,9\}$, $S_8 = \{6\}$, $S_9 = \{7\}$. $(|C| + |C_2|)/2 = 4.5 < 5$, logo $t_1^2 = 4.5$, $C = C \cup C_2$.

Quando $k = 3$. O vértice 2 pode ser adicionado ao S_8, vértice 4 e 8 ao S_9, isto é, $S_8 = \{2,9\}$, $S_9 = \{4,7,8\}$. Então $U_3 = \{1,3,5,6,7,9\}$. Assim o ICH produz $C_3 = \{S_{10}, S_{11}, S_{12}, S_{13}\}$, com $S_{10} = \{1,9\}$, $S_{11} = \{3,5\}$, $S_{12} = \{6\}$, $S_{13} = \{7\}$. Deste modo, $(|C| + |C_3|)/3 = 4.33 < 4.5$, então $t_1 s = 4.33$ e $C = C \cup C_3$.

Este processo continua até o número de classes geradas exceder nosso limite, então nós paramos com:

$C^* = \{S_1, \ldots, S_{13}\}$, $y_S^* = \frac{1}{3}$ para $i = 1, \ldots, 13$ e $\lfloor t_1^* = 4 \rfloor$.

A heurística de coloração fracionária é um poderoso dispositivo para o fortalecimento do limite superior obtido pela coloração inteiro (BALAS; XUE, 1996). Por este motivo, esta monografia define uma heurística de coloração baseada na HCF.

3.4.3 Re-Coloração

A re-coloração em Tomita et al. (2010) é baseado na seguinte propriedade. Para uma dada coloração admissível de um grafo, $C = \{C_1, \ldots, C_n\}$, é possível reatribuir um vértice $v \in C_k$ a um conjunto independente diferente $C_j, j < k$, se as seguintes propriedades são verdadeiras:

1. Existe um vértice $w \in C_j$ tal que $\Gamma(v) \cap C_j = \{w\}$, isto é, w é o único membro da vizinhança de v em C_j;
2. Existe uma classe de cor C_l tal que $|\Gamma(w) \cap C_l| = \emptyset$, isto é, não contém qualquer vizinho de w.

Se for esse o caso, é possível produzir uma nova coloração admissível através da troca de v e w de modo que $UB[v] = j$ e $UB[w] = l$.

Seja (C, P, LB) um subproblema do PCM. O algoritmo de re-coloração funciona da seguinte maneira:

1. Encontre uma coloração parcial $C_1, C_2, \ldots, C_{LB}$;
2. Para cada vértice v não colorido veja se é possível aplicar a re-coloração.;
3. Se possível, troque os vértices e retorne para o passo 2;
4. Senão, remova todos os vértices adjacentes a v e adicione v a C_i;
5. Volte para o passo 2.

O algoritmo de re-coloração primeiro calcula um conjunto inicial de classes de cores com $i \leq LB$, cada vértice candidato adicional, a menos que seja trocado, será um vértice com cor $UB[v] > LB$ na nova coloração parcial, de modo que todos eles têm potencial para podar o espaço de busca se recolorido com êxito(SEGUNDO et al., 2013). O Algoritmo 6 e 7 descrevem o algoritmo de re-coloração em pseudocódigo.

Algorithm 6 Passo de re-coloração

```
1:  function RE_COLORAÇÃO(v, LB, C_1, ..., C_LB, C_new)
2:      for all k_1 = 1 até LB − 1 do
3:          if |C_k1 ∩ Γ(v)| = 1 then
4:              w = ao vértice em C_k1 ∩ Γ(v)
5:              for all k_2 = k_1 + 1 até LB do
6:                  if C_k1 ∩ Γ(w) = ∅ then
7:                      C_new = C_new \ {v}
8:                      C_k1 = C_k1 ∪ {v}
9:                      C_k1 = C_k1 \ {w}
10:                     C_k2 = C_k2 ∪ {w}
11:                     return
12:         else
13:             if C_k1 ∩ Γ(v) = ∅ then
14:                 C_new = C_new \ {v}
15:                 C_k1 = C_k1 ∪ {v}
16:                 return
```

Fonte: Segundo et al. (2013) (com adaptação)

Algorithm 7 Procedimento de re-coloração

```
1:  function COLORAÇÃO(V, LB, R, UB)
2:      U = V, S = V
3:      i = 1, n = 0
4:      while U ≠ ∅ do
5:          C_i = ∅
6:          while S ≠ ∅ do
7:              selecione o primeiro vértice u ∈ S
8:              if i ≥ LB e |S| = 1 then
9:                  RE_COLORAÇÃO(u, LB, C_1, ..., C_LB, C_i)
10:             if u não foi recolorido then
11:                 C_i = C_i ∪ u
12:                 S = S \ {u}
13:                 S = S ∩ Γ̄(u)
14:                 if i ≥ LB then
15:                     R[n] = u
16:                     UB[u] = i
17:                     n = n + 1
18:         U = U \ C_i
19:         S = U
20:         if C_k ≠ ∅ then
21:             i = i + 1
22:     return i
```

Fonte: Segundo et al. (2013) (com adaptação)

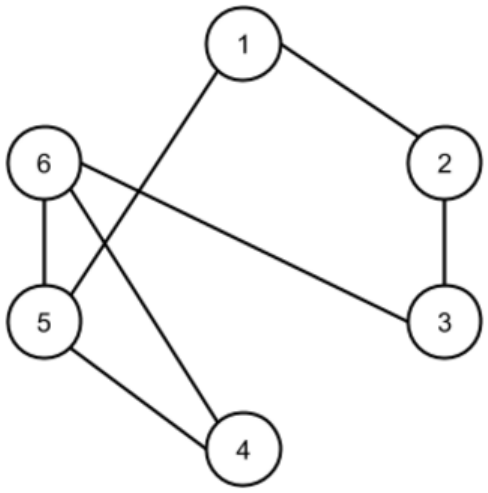
A versão descrita no início desta seção tenta recolorir todos os vértices que não estão nas classes de cores $C_1, \ldots, C_{LB}$. Contudo, realizar está operação nesses vértices causa sobrecarga. Em busca de preservar sua capacidade de poda, Segundo et al. (2013) tentaram algumas variantes em busca de um melhor compromisso. A estratégia que obteve em média o melhor desempenho, aplica a re-coloração nos vértices que podem ainda melhorar a poda globalmente, isto é, produzir uma classe de cor C_i vazia e, portanto, diminuir o $UB[v]$ de todos os vértices remanescentes não coloridos. Isto só pode ser conseguido quando S só possui um elemento (avalie o passo 8)

Considere o seguinte exemplo da Figura 5. Seja (3, 4, 5, 6, 1, 2) a ordem inicial dos vértices e $LB = 2$. A coloração obtida dos vértices com $UB[v] \leq LB$ será:

1. $C_1 = \{3,5\}$
2. $C_2 = \{4,1\}$

Em seguida o vértice 6 será colorido e como $|S| \neq 1$, então $C_3 = \{6\}$, depois é a vez do vértice 2, como $|S| = 1$, então 2 será recolorido, de modo que $C_1 \cap \Gamma(2) = \{3\}$ e $C_2 \cap \Gamma(3) = \emptyset$, assim podemos colocar o vértice 2 em C_1 e o vértice 3 em C_2, deste modo re-coloração é aplicada

Figura 5 – Grafo usado no Exemplo 3



Fonte: Elaborado pelos autores

com sucesso. A coloração final obtida é:

1. $C_1 = \{2,5\}$
2. $C_2 = \{4,1,3\}$
3. $C_3 = \{6\}$

Na tabela seguinte, o valor $UB[v]$ para cada vértice v é mostrado:

Tabela 3 – Tabela com os valores $UB[v]$ para cada vértice v

v	3	4	5	6	1	2
$UB[v]$	2	2	1	3	2	1

Fonte: Produzido pelos autores

Observando os resultados de Segundo et al. (2013) pode-se notar que re-coloração é mais rápida em grafos de densidade alta. Na nossa proposta também tentamos colocar os vértices contidos em R em cada passo da coloração fracionário, isso poderá ser visto quando o algoritmo proposto for apresentado na seção 4.

4 ALGORITMOS DESENVOLVIDOS

4.1 Algoritmos BBMC, BBMCR e BBMCF

Implementamos as heurísticas gulosas de coloração inteira, fracionária e de re-coloração nos algoritmos BBMC, BBMCF e BBMCR respectivamente. Todos os algoritmos serão desenvolvidos utilizando a linguagem de programação C++.

Utilizamos a estrutura básica apresentada na seção 3.3 nos algoritmos BBMC, BBMCF e BBMCR, em que no passo 4 realizamos a partição de P usando sua heurísticas de coloração correspondente. Além disso, utilizamos a maior cor obtida pelas heurísticas e as cores dos vértices como *bounding*, da seguinte maneira:

$$\text{se } |P| + \text{máximo}\{UB[v] | v \in P\} \leq LB \text{ pare.}$$

Com estas modificações o algoritmo 4 fica da seguinte forma:

Algorithm 8 Algoritmo branch-and-bound para o PCM

```
1: function CLIQUE(C, P, LB)
2:     if |C| > LB then
3:         LB = |C|
4:     Usando uma heurística de coloração fracionária particiona P = U ∪ R tal que ω(G[U]) ≤ LB − ω(C).
5:     if |P| + máximo{UB[v] | v ∈ P} ≤ LB then return
6:     Seja (v_1, ..., v_n) uma ordem dos vértices de R
7:     for i ← n até 1 do
8:         if |C| + UB[v] ≤ LB then
9:             return
10:        CLIQUE(C ∪ {v_i}, P ∩ Γ(v_i), LB)
11:        P = P \ {v_i}
```

Fonte: Elaborado pelos autores

4.2 Utilizar Estratégia Bit-Paralela

Segundo et al. (2007) provam que um modelo bit paralelo adequado pode causar uma redução exponencial em função do tamanho da palavra da CPU. Por este motivo, modelamos o problema da clique máximo em uma estratégia bit-paralela.

Usando estratégia bit-paralela, assumimos que o conjunto de vértices é $V = \{0, \ldots, n-1\}$, para um n de tamanho conhecido. Então podemos declarar um grafo G, usando as notações apresentadas na seção 3.2, da seguinte forma:

Algorithm 9 Declaração de um Grafo usando vetor de bits

```
1: #define MAX_VERTEX 4000
2: word bit[MAX_VERTEX][MAX_VERTEX/INT_SIZE + 1]
3: word V[MAX_VERTEX/INT_SIZE + 1]
```

Fonte: Elaborado pelo autor

O vetor de bits *bit* armazena a lista de adjacência e V armazena os vértices de G. Seja uma palavra de tamanho 8 e o grafo da Figura 3 a representação gráfica de *bit* e V pode ser encontrada na Figura 6. Como os índices dos vetores em programação começam de 0, então nós representamos o vértice 1 com a posição 0 e assim sucessivamente.

Figura 6 – Representação gráfica de *bit* e V

V

0	1	2	3	4	5	6	7
1	1	1	1	1	1	0	0

bit

	0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0
2	0	1	0	1	0	1	0	0
3	0	0	1	0	1	1	0	0
4	1	0	0	1	0	1	0	0
5	0	0	1	1	1	0	0	0

Fonte: Elaborado pelo autor

Seja $Vnbr$ o número de vértices em um grafo e $W = Vnbr/INT\_SIZE + 1$, então as seguintes operações representam:

- COUNT(bit[v], W): O grau de v;
- COUNT(V, W): O números de vértices em V;
- index(i, j): O vértice que está na palavra i e no bit j;
- $\sim$U: O complemento de U.

As operações setbit(U, v), resetbit(U, v) e test(U, v) representam a inserção, remoção e o teste existência, respectivamente, do vértice v em U. Para realizarmos a união, intercessão e subtração dos conjuntos N e M utilizamos os Algoritmos 10, 11 e 12.

Algorithm 10 UNIÃO

1: word $U[W]$
2: **for** $i = 0$ até W **do**
3: $\quad U[i] = N[i]|M[i]$

Fonte: Elaborado pelos autores

Algorithm 11 INTERSECÇÃO

1: word $I[W]$
2: **for** $i = 0$ até W **do**
3: $\quad I[i] = N[i]\&M[i]$

Fonte: Elaborado pelos autores

Algorithm 12 SUBTRAÇÃO

1: word $S[W]$
2: **for** $i = 0$ até W **do**
3: $\quad S[i] = N[i]\& \sim M[i]$

Fonte: Elaborado pelos autores

Agora com nossa estratégia bit-paralela definida, implementamos os algoritmos BBMC, BBMCF e BBMCR utilizando bit-parelelismo. Utilizamos todas as operações apresentadas nessa seção nas operações de união, intercessão, subtração de conjuntos, remoção, inserção, teste de existência presentes nos algoritmos de heurística de coloração inteira, fracionária e de recoloração, e no algoritmo CLIQUE.

4.3 Heurística de Coloração Fracionária

Desenvolvemos uma heurística de coloração, denominada de coloração fracionária modificada. A heurística de coloração fracionária modificada funciona da seguinte forma:

1. Determina cada classe de cor C_c com $c \leq LB$ utilizando heurística de coloração inteira.
2. Aplique a heurística de coloração fracionária nas classes de cores geradas no passo 1.
   2.1. No primeiro passo da coloração fracionária, onde os vértices são coloridos novamente, tente colorir os vértives com $UB[v] > LB$, ou seja, os vértices que não estão na coloração gerada pelo passo 1.
   2.2. Se conseguir, então existe $estavel_i \cap \Gamma(v) = \emptyset$, adicione v a $estavel_i$ e remova v do conjunto vértices com $UB[v] > LB$.

2.3. Senão, execute a HCF normalmente.

3. Se a coloração fracionária conseguiu reduzir o limite, seja t o limite encontrado, gere as classes de cores $C_{t+1}, C_{t+2}, \ldots, C_n$ com a coloração inteira, onde n é o maior cor encontrado pela coloração inteira.
4. Senão, gera as classes de cores $C_{t+1}, C_{t+2}, \ldots, C_n$ com a coloração inteira.
5. Salve a ordem dos vértices com $UB[v] > LB$.

O Algoritmo 13 apresenta o algoritmo de coloração fracionária modificada.

Inicialmente a heurística de coloração fracionária modificada constrói as classes de cores $\{estavel_1, \ldots, estavel_{LB}\}$, em seguida é verificado se todos os vértices de V foram coloridos, se foram então encerra a coloração e retorna 0 (veja que se todos os vertices estão contidos em algum $estavel_i$, tal que $i \leq LB$, nenhum vértice será ramificado).

Agora, aplicamos a coloração fracionário, como a HCF foi explicada na seção 3.4.2 vamos nos focar apenas na nossa alteração. Seja v um vértice com $UB[v] > LB$ e o $estavel_i$ tal que $i \leq LB$, onde $estavel_i \cap \Gamma(v) = \{w\}$ e exista um $estavel_j \cap \Gamma(w) = \emptyset$, então quando a HCF for atribuir uma outra cor para w, este provavelmente seja colocado no $estavel_j$, neste momento note que esta etapa se comporta exatamente como a proposta da recolação apresentada na seção 3.4.3. Assim, antes de atribuímos mais um cor para os vértices em U tentamos primeiro recolorir os vértices com $UB[v] > LB$. Portanto, além de diminuir o tamanho do limite superior a heurística de coloração fracionária modificada também aumenta o número de vértices com $UB[v] \leq LB$.

Digamos que ao termino da HCF nosso limite tenha sido reduzido para $LB - 2$, então podemos criar mais dois estáveis $estavel_{LB-1}$ e $estavel_{LB}$. Deste modo, estamos aumentando novamente o número de vértice com $UB[v] \leq LB$. Esta operação é realizada quando aplicamos a HCI nos vértices que ainda não foram coloridos, onde armazenamos apenas os vértices com $UB[v] > LB$.

Algorithm 13 Procedimento da heurística de coloração fracionária modificada

function COLORAÇÃO_FRACIONÁRIA_MODIFICADA(V, LB, O, UB)
  $S = V$
  $estavel = \emptyset$
  aplique a heurística de coloração inteira em S para gerar as classes de cores $estavel_1, \ldots, estavel_{LB}$
  if todos os vértices em S foram coloridos **then**
    return 0
  $k = 2$, $t = \infty$, $size = LB$
  if exister mais de um máximal **then**
    while TRUE **do**
      $U = V \setminus S$
      for all $estavel_i$ não máximal **do**
        $T = \{v | estavel_i \cap \Gamma(v) = \emptyset, UB[v] > LB\}$
        while $T \neq \emptyset$ **do**
          $v =$ selecione um vértice em T
          $estavel_i = estavel_i \cup \{v\}$
          $S = S \setminus \{v\}$
          $T = T \setminus \{v\}$
          $T = T \cap \overline{\Gamma}(v)$
        while houver candidatos no $estavel_i$ **do**
          $v =$ seleciona um vértice que pode entrar no $estavel_i$
          $estavel_i = estavel_i \cup \{v\}$
          $U = U \setminus \{v\}$
      aplique a HCI em $G[U]$, seja C_k a coloração entrada
      $novo\_t = (size + |C_k|)/k$
      if $t - novo\_t > 0.1$ **then**
        $t = novo\_t$
        $size = size + |C_k|$
        $k = k + 1$
      else
        pare
  $t =$ minimo$\{t, LB\}$
  Aplique a coloração inteira em S começando da cor $t + 1$ e armazene em O a ordem dos vértices com $UB[v] > LB$.

Fonte: Elaborado pelo Autor

No pior caso a heurística de coloração fracionária modificada se comporta como a coloração inteira, isto ocorre quando a coloração fracionária não faz nenhum melhora e não consegue recolorir nenhum vértice. Por questões de eficiência armazenamos quais são os vértices não maximais e os seus candidatos quando estamos gerando as classes de cores, assim evitamos percorrer todos os estáveis e os vértices quando estamos em busca de não maximais e de candidatos. Note também que só realizamos a coloração fracionária se tivermos mais de um não maximal.

5 RESULTADOS COMPUTACIONAIS

Nesta seção são apresentados os resultados da execução dos algoritmos BBMC, BBMCF, BBMCR, BBMCFM e BBMCFM2. Os resultados são apresentados na Tabela 5 e Tabela 4 para os grafos DIMACS e aleatórios.

Os algoritmos foram executados em um computador com o processador Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 6GB de memória RAM e Linux 64 bit com tempo limite de execução de uma hora. O desempenho dos algoritmos são avaliados pelos seus tempos de execução e os números de subproblemas. Utilizadas instâncias de grafos aleatórios e grafos do benchmark DIMACS[1].

Implementamos dois algoritmo utilizando a heurística de coloração fracionária modificada, o BBMCFM e BBMCFM2. O BBMCFM utiliza a coloração fracionária modificada exatamente como apresentada na seção 4.3. Já o BBMCFM2 restringe a coloração fracionária em duas iterações, em uma tentativa de diminuir a sobrecarga de operações.

Para cada tabela comparamos o número de subproblemas gerados e o tempo de computação para se resolver as instâncias de tamanhos $(|V|)$, densidades e tamanho de cliques diferentes (ω). O ω se refere a maior clique encontrada entre os algoritmo e o *LB* é o limite inferior encontrado.

[1] Centro de Matemática Discreta e Ciência da Computação Teórica

Tabela 4 – Resultados em grafos aleatórios

Grafo					Número de Subproblemas					CPU (segundos)				
Nome	|V|	Densidade	ω	LB	BBMC	BBMCF	BBMCR	BBMCFM	BBMCFM2	BBMC	BBMCF	BBMCR	BBMCFM	BBMCFM2
200-0.90-1	200	0.90	41	35	20875498	8786824	20329914	**5686633**	6605049	**20.80**	62.3460	30.4482	31.1645	25.2560
200-0.95-1	200	0.95	60	57	22092722	5607578	21452914	**3529481**	4486337	28.9958	63.0708	44.4350	32.0433	**24.87**
200-0.95-2	200	0.95	62	59	5126413	1601306	5055392	**1021741**	1274419	**6.88**	18.3092	10.9305	9.1612	7.2349
200-0.95-3	200	0.95	64	62	5426102	1433181	5255205	**901404**	1139326	7.5167	17.2897	11.6878	9.1017	**6.79**
200-0.95-4	200	0.95	61	56	14638968	4177741	14284449	**2983659**	3732892	**19.48**	47.0687	29.9967	27.2926	20.8862
200-0.95-5	200	0.95	59	54	21195682	6219942	20596099	**3863680**	4819673	27.5088	65.3643	41.7204	33.8176	**26.34**
200-0.95-6	200	0.95	63	57	10862601	3649774	10689083	**2304815**	2786517	**14.52**	40.1136	22.4940	20.3121	15.1690
200-0.95-7	200	0.95	61	54	17668850	4712694	17216081	**2749906**	3565455	23.9556	52.5337	35.9307	26.2097	**20.46**
200-0.95-8	200	0.95	63	56	10894876	3395100	10623123	**2159313**	2630743	**14.15**	35.1387	21.8917	18.5062	14.3803
200-0.95-9	200	0.95	61	54	8843300	2590150	8679730	**1584979**	1974948	11.8414	29.2449	18.2508	14.1030	**11.09**
200-0.95-10	200	0.95	63	62	3740829	1164541	3669182	**729848**	893086	5.1854	13.9030	8.0241	6.7408	**5.14**
200-0.98-1	200	0.98	92	88	111584	15865	107123	**8426**	10681	0.1999	0.3192	0.3424	0.1101	**0.10**
200-0.98-2	200	0.98	95	91	2055	789	2055	**787**	907	**0.01**	0.0099	0.0117	0.0092	0.0076
200-0.98-3	200	0.98	96	91	35116	4000	34364	**1626**	2848	0.0720	0.0724	0.1161	**0.02**	0.0260
200-0.98-4	200	0.98	96	88	32538	4658	32462	**1911**	2799	0.0592	0.0706	0.1035	**0.02**	0.0281
200-0.98-5	200	0.98	93	87	22624	4693	22602	**1913**	2349	0.0432	0.0721	0.0740	0.0230	**0.02**
200-0.98-6	200	0.98	93	88	79876	8136	79828	**4225**	6636	0.1576	0.1394	0.2698	0.0669	**0.06**
200-0.98-7	200	0.98	94	86	35662	5129	35122	**2690**	3868	0.0680	0.0885	0.1160	0.0391	**0.04**
200-0.98-8	200	0.98	98	92	25036	3156	21638	**1667**	1921	0.0475	0.0487	0.0773	0.0275	**0.02**
200-0.98-9	200	0.98	92	88	18123	**2781**	18123	2867	5159	**0.04**	0.0478	0.0621	0.0427	0.0475
200-0.98-10	200	0.98	96	90	4594	1571	4594	**926**	1086	**0.01**	0.0172	0.0160	0.0160	0.0092
200-0.99-1	200	0.99	118	117	179	135	179	**130**	131	**0.00**	0.0014	0.0011	0.0014	0.0022
200-0.99-2	200	0.99	122	120	276	255	276	123	**123**	**0.00**	0.0022	0.0019	0.0016	0.0011
200-0.99-3	200	0.99	123	121	212	210	212	130	**130**	**0.00**	0.0011	0.0011	0.0012	0.0011
200-0.99-4	200	0.99	121	119	231	230	231	225	**225**	**0.00**	0.0012	0.0016	0.0018	0.0022
200-0.99-5	200	0.99	119	114	547	538	547	**234**	235	**0.00**	0.0030	0.0019	0.0017	0.0021
200-0.99-6	200	0.99	121	119	250	248	250	246	**246**	0.0017	0.0018	**0.00**	0.0021	0.0027
200-0.99-7	200	0.99	122	120	129	125	129	123	**123**	**0.00**	0.0010	0.0011	0.0012	0.0015
200-0.99-8	200	0.99	123	123	42	**8**	42	16	18	**0.00**	0.0006	0.0013	0.0005	0.0006
200-0.99-9	200	0.99	117	114	455	375	455	118	**118**	0.0015	0.0035	0.0030	0.0013	**0.00**
200-0.99-10	200	0.99	121	118	254	247	254	123	**123**	0.0014	0.0018	0.0017	0.0015	**0.00**
300-0.90-1	300	0.90	47	40	2870243376	**376342967**	1986851867	464773309	685653969	timeout	timeout	timeout	timeout	timeout
300-0.98-1	300	0.98	117	107	1232080850	100584484	751749999	**35562883**	70884219	timeout	timeout	timeout	1095.3330	**1088.20**
300-0.98-2	300	0.98	120	115	295155714	17775777	285936376	**7998244**	14555035	930.8152	660.7726	1486.7941	**245.62**	260.2586
300-0.98-3	300	0.98	123	119	349573838	17204530	336176470	**9278944**	18070124	1097.4183	686.9012	1768.2509	300.7601	**298.44**
300-0.98-4	300	0.98	120	111	167390763	7596225	167321005	**3753955**	8201217	525.0944	299.4765	872.6504	**123.71**	131.5659
300-0.98-5	300	0.98	121	111	82652778	4675542	82429491	**2556522**	4698228	260.3051	184.9258	430.4211	80.0434	**75.63**
300-0.98-6	300	0.98	121	110	238962616	15289182	235451945	**7796403**	15210643	731.2073	568.1903	1196.9488	**239.60**	266.3419
300-0.98-7	300	0.98	119	114	516806770	47684877	516624440	**22593327**	40898241	1595.5964	1601.6188	2674.4259	667.5783	**636.18**
300-0.98-8	300	0.98	122	112	170967309	7242315	167114228	**4119119**	8210182	536.7680	278.1587	884.9007	**127.36**	133.8384
300-0.98-9	300	0.98	117	106	1213819607	96990987	724026303	**36724927**	71362902	timeout	3318.2697	timeout	**1086.54**	1094.1579
300-0.98-10	300	0.98	120	116	651657031	42386810	635434702	**20495943**	43713654	2028.4177	1633.5583	3296.5233	**670.26**	694.3447
300-0.99-1	300	0.99	157	150	2350262	461369	2350262	**45300**	79665	9.1344	14.9620	15.9744	**1.43**	1.6475
300-0.99-2	300	0.99	160	156	25018	1911	25018	**970**	1215	0.1056	0.0556	0.1850	**0.02**	0.0246
300-0.99-3	300	0.99	161	157	5739	2199	5739	**981**	1062	0.0244	0.0598	0.0420	0.0214	**0.02**
300-0.99-4	300	0.99	159	153	19322	3983	19322	**1525**	1950	0.0803	0.1052	0.1366	**0.03**	0.0333
300-0.99-5	300	0.99	162	157	2099	938	2099	**680**	723	**0.01**	0.0153	0.0168	0.0098	0.0123
300-0.99-6	300	0.99	154	144	1166565	51175	1166565	**15340**	45180	4.4298	1.9449	8.0279	**0.53**	0.9157
300-0.99-7	300	0.99	161	159	17786	1410	17786	**587**	984	0.0785	0.0464	0.1358	**0.02**	0.0240
300-0.99-8	300	0.99	162	155	3798	1391	3798	**1018**	1140	**0.02**	0.0353	0.0292	0.0174	0.0177
300-0.99-9	300	0.99	159	155	8974	1149	8960	**750**	778	0.0377	0.0309	0.0670	0.0151	**0.01**
300-0.99-10	300	0.99	163	157	52885	3322	52885	**779**	1717	0.2219	0.1285	0.3778	**0.02**	0.0375

Tabela 5 – Resultados em grafos DIMACS

Grafo					Número de Subproblemas					CPU (segundos)				
Nome	|V|	Densidade	ω	LB	BBMC	BBMCF	BBMCR	BBMCFM	BBMCFM2	BBMC	BBMCF	BBMCR	BBMCFM	BBMCFM2
brock200_1	200	0.75	21	18	305268	179391	295072	**112074**	120484	**0.19**	0.6868	0.2584	0.3079	0.2687
brock200_2	200	0.50	12	8	3941	2908	3784	**1764**	1806	0.0053	0.0069	**0.00**	0.0066	0.0039
brock200_3	200	0.61	15	11	15373	10806	14882	**6466**	6836	0.0146	0.0317	**0.01**	0.0170	0.0122
brock200_4	200	0.66	17	14	58489	38000	56388	**22987**	24099	**0.03**	0.1115	0.0405	0.0535	0.0474
brock400_1	400	0.75	27	21	115686743	62817595	112737417	**34781004**	38467406	**105.98**	421.6063	155.1277	162.3512	135.4872
brock400_2	400	0.75	29	19	64142222	34780940	62558628	**18920569**	20987652	**61.57**	244.8015	90.2715	90.9121	76.5506
brock400_3	400	0.75	31	19	39573573	21471052	38530141	**12020342**	13267956	**36.28**	143.4371	52.9985	54.6108	45.9361
brock400_4	400	0.75	33	20	13321699	7107138	13022059	**3808316**	4219390	**14.51**	58.2903	20.7073	20.8560	16.9439
brock800_1	800	0.65	23	17	1921600029	**502032531**	1876763298	602759157	612022627	**2080.45**	timeout	3289.5454	timeout	2495.0747
brock800_2	800	0.65	24	15	1594558739	480533053	1558811600	**328103125**	502055211	**1805.79**	timeout	2792.8651	2167.3814	2113.0110
brock800_3	800	0.65	25	16	1256344014	509761252	1227800604	**266010704**	400626281	**1396.15**	timeout	2145.9666	1663.3930	1635.3582
brock800_4	800	0.65	26	14	479302346	300662572	470667347	**82395050**	140712035	**651.64**	3153.3091	980.3187	664.4234	682.4056
C125.9	125	0.90	34	33	35309	17837	34348	**12253**	13388	**0.03**	0.0878	0.0373	0.0459	0.0370
C250.9	250	0.90	44	35	908740674	367249155	882564905	**193039001**	226922896	**914.94**	2728.1599	1335.2597	1117.7697	927.4203
C500.9	500	0.90	53	46	2701641943	**353568215**	1707328623	358632274	639070432	timeout	timeout	timeout	timeout	timeout
C1000.9	1000	0.90	54	54	2063394333	**228464236**	1119588470	274299025	431031601	timeout	timeout	timeout	timeout	timeout
C2000.5	2000	0.50	16	11	2971363395	**447332886**	1853015164	545188644	785701917	timeout	timeout	timeout	timeout	timeout
C2000.9	2000	0.90	61	61	1389785397	**134121988**	672648977	159915756	252594342	timeout	timeout	timeout	timeout	timeout
c-fat200-1	200	0.08	12	12	4	4	4	3	**3**	0.0003	0.0004	0.0003	0.0003	**0.00**
c-fat200-2	200	0.16	24	24	1	1	1	1	**1**	0.0003	0.0004	0.0004	0.0002	**0.00**
c-fat200-5	200	0.43	58	58	27	27	27	27	**27**	0.0006	0.0005	0.0003	0.0003	**0.00**
c-fat500-1	500	0.04	14	14	1	1	1	1	**1**	**0.00**	0.0015	0.0019	0.0010	0.0011
c-fat500-2	500	0.07	26	26	1	1	1	1	**1**	0.0021	0.0028	0.0014	**0.00**	0.0011
c-fat500-5	500	0.19	64	64	1	1	1	1	**1**	0.0015	0.0024	0.0028	0.0016	**0.00**
c-fat500-10	500	0.37	126	126	1	1	1	1	**1**	0.0014	0.0017	**0.00**	0.0014	0.0014
gen200_p0.9_44	200	0.90	44	29	398425	160364	391075	**106038**	138483	**0.40**	1.0656	0.5862	0.5575	0.5387
gen200_p0.9_55	200	0.90	55	38	367429	135460	358644	**101089**	119642	**0.38**	1.0039	0.5674	0.5685	0.4725
gen400_p0.9_55	400	0.90	55	33	1626377672	282911629	1363695333	**238521229**	379161417	2993.8104	timeout	timeout	**2806.40**	2948.2212
gen400_p0.9_65	400	0.90	49	33	2442191752	**314567253**	1631883253	347734387	564684120	timeout	timeout	timeout	timeout	timeout
gen400_p0.9_75	400	0.90	75	40	2335134802	315120806	1632575945	**299367980**	434564097	timeout	timeout	timeout	3404.6640	**3000.54**
hamming6-2	64	0.90	32	32	1	1	1	1	**1**	**0.00**	0.0001	0.0001	0.0001	0.0001
hamming6-4	64	0.35	4	4	109	109	97	96	**96**	0.0001	0.0002	**0.00**	0.0002	0.0001
hamming8-2	256	0.97	128	128	1	1	1	1	**1**	0.0005	0.0007	**0.00**	0.0004	0.0011
hamming8-4	256	0.64	16	16	18105	7245	17477	**4200**	4766	**0.02**	0.0464	0.0173	0.0221	0.0198
hamming10-2	1024	0.99	512	512	1	1	1	1	**1**	**0.01**	0.0055	0.0056	0.0054	0.0056
hamming10-4	1024	0.83	40	32	1976408434	**154976822**	1191266076	187889702	280054775	timeout	timeout	timeout	timeout	timeout
johnson8-2-4	28	0.56	4	4	24	24	24	24	**24**	**0.00**	0.0000	0.0001	0.0000	0.0000
johnson8-4-4	70	0.77	14	14	168	62	168	2	**2**	0.0004	0.0005	0.0002	**0.00**	0.0001
johnson16-2-4	120	0.76	8	8	369095	311957	365418	**267847**	267877	**0.05**	0.1181	0.0649	0.0703	0.0688
johnson32-2-4	496	0.88	16	16	19769232483	**5501596100**	12772098339	7686522844	8135394267	timeout	timeout	timeout	timeout	timeout
keller4	171	0.65	11	8	12285	7586	11959	**3353**	3977	0.0088	0.0210	0.0076	0.0086	**0.01**
keller5	776	0.75	27	15	2632068439	**267415148**	1775887113	288100813	576407832	timeout	timeout	timeout	timeout	timeout
MANN_a9	45	0.93	16	16	60	60	54	54	**54**	**0.00**	0.0003	0.0001	0.0002	0.0002
MANN_a27	378	0.99	126	125	37904	32190	37904	28712	**28712**	**0.14**	0.4922	0.2298	0.3821	0.3658
MANN_a45	1035	1.00	345	342	2851246	2469957	2851241	**1980488**	1980534	**48.18**	218.6293	81.8011	131.0775	126.3348
p_hat300-1	300	0.24	8	7	1545	1331	1484	**911**	914	**0.00**	0.0068	0.0018	0.0023	0.0022
p_hat300-2	300	0.49	25	24	3757	2535	3711	**2133**	2246	**0.00**	0.0232	0.0064	0.0172	0.0152
p_hat300-3	300	0.74	36	30	454361	263818	447684	**182032**	196360	**0.58**	2.2420	0.7940	1.0514	0.8615
p_hat500-1	500	0.25	9	7	9899	7657	9565	**5667**	5714	**0.01**	0.0230	0.0111	0.0124	0.0115
p_hat500-2	500	0.50	36	34	60581	38627	60052	**27164**	28862	**0.10**	0.4933	0.1465	0.1992	0.1604
p_hat500-3	500	0.75	50	44	22108747	11252038	21814805	**7494230**	8438807	**40.48**	169.4609	59.6663	72.4279	56.6301
p_hat700-1	700	0.25	11	9	26151	18702	25376	**14465**	14615	**0.02**	0.0761	0.0280	0.0428	0.0367
p_hat700-2	700	0.50	44	41	583204	340244	578660	**240576**	262578	**1.34**	7.1829	2.1035	2.9111	2.2926
p_hat1000-1	1000	0.24	10	9	167600	130518	163064	**74385**	82364	**0.12**	0.4690	0.1684	0.2120	0.1878
p_hat1000-2	1000	0.49	46	43	21535710	12404261	21416937	**6244225**	7939306	**55.76**	300.3035	86.4167	97.3893	79.1614
p_hat1000-3	1000	0.74	61	60	1140524385	**120598478**	704545842	177256079	295633338	timeout	timeout	timeout	timeout	timeout
p_hat1500-1	1500	0.25	12	9	1359727	1143357	1328481	**585796**	668914	**1.20**	5.6911	1.7655	1.9531	1.7636
p_hat1500-2	1500	0.51	63	58	803034232	**76428352**	508387880	115902787	201579378	timeout	timeout	timeout	timeout	timeout
p_hat1500-3	1500	0.75	85	85	600848248	**53965912**	376036661	81556475	157412255	timeout	timeout	timeout	timeout	timeout
san200_0.7_1	200	0.70	30	15	4222	1603	4056	**1023**	1122	**0.00**	0.0084	0.0054	0.0043	0.0042
san200_0.7_2	200	0.70	18	12	458	368	450	**278**	282	**0.00**	0.0018	0.0018	0.0012	0.0012
san200_0.9_1	200	0.90	70	45	71	71	71	71	**71**	**0.00**	0.0006	0.0008	0.0007	0.0008
san200_0.9_2	200	0.90	60	35	25133	7740	24193	**5577**	6621	**0.02**	0.0501	0.0314	0.0318	0.0293
san200_0.9_3	200	0.90	44	25	8064	3284	7809	**2387**	2720	**0.01**	0.0186	0.0111	0.0111	0.0097
san400_0.5_1	400	0.50	13	7	2418	984	2328	**748**	778	**0.00**	0.0076	0.0050	0.0049	0.0046
san400_0.7_1	400	0.70	40	20	868609	210271	828267	**144571**	158219	1.0316	1.6799	1.3102	0.6766	**0.60**
san400_0.7_2	400	0.70	30	15	1024801	**425594**	1009874	973788	1085212	**1.45**	3.6020	1.8516	4.3968	3.9551
san400_0.7_3	400	0.70	22	12	1079316	555797	1066383	**318312**	340437	1.3438	3.7374	1.6211	1.4459	**1.25**
san400_0.9_1	400	0.90	56	50	2913553643	**202633431**	1808321515	226844618	468620152	timeout	timeout	timeout	timeout	timeout
san1000	1000	0.50	15	8	160367	80076	157511	**33399**	36351	0.5220	1.0304	0.6039	0.2823	**0.26**
sanr200_0.7	200	0.70	18	14	110794	68371	107117	**34980**	43160	**0.07**	0.2258	0.0855	0.1049	0.0895
sanr200_0.9	200	0.90	42	38	7438994	3008822	7155586	**1843565**	2522043	**7.98**	23.0140	10.9503	11.3620	9.9706
sanr400_0.5	400	0.50	13	10	254123	180432	245373	**97948**	106720	**0.14**	0.5579	0.1957	0.2389	0.2024
sanr400_0.7	400	0.70	21	17	57236899	34334803	55554457	**15101935**	20014114	**41.15**	160.8070	58.5075	57.6085	53.5060

Cabe aqui ressaltar que os algoritmos apresentados no estado da arte não são formalmente detalhados. Em alguns artigos encontramos o seguinte trecho: Os vértices em V são inicialmente ordenados em ordem decrescente de grau e empates são resolvidos aleatoriamente. Este trecho mostra a dificuldade de comparação e reprodução com os trabalhos apresentados na literatura. Nesta monografia, todos os algoritmos são implementados usando a mesma metodologia. Por isso os resultados podem ser diferentes dos encontrados na literatura.

Observando os grafos estruturados do benchmark DIMACS, podemos notar que nas

instâncias da família *brock*, instâncias de densidade média, o BBMC resolveu mais rápido quase todas as instâncias, exceto *brock200_2* e *brock200_3* em que o BBMCR foi melhor.

Nas instâncias da família *C* de densidade alta, os resultados foram mais diversos, o BBMC foi mais eficiente nas duas primeiras instâncias e todos os outros algoritmos estouraram o tempo limite. Porém, na instância *C500.9* o algoritmo BBMCFM encontra uma clique de tamanho 53 antes do tempo limite acabar, onde a segunda melhor solução encontrada foi pelo BBMC de 51.

Na família *c-fat*, densidade baixa, os algoritmos BBMCFM e BBCFM2 obtiveram melhor desempenho em quase todas as instância, exceto nos casos *c-fat500-1* e *c-fat500-10* que foram resolvidas com melhor desempenho pelo BBMC e BBMCR respectivamente.

Na família *gen* podemos notar resultados interessantes, as duas primeiras instâncias o BBMC se destaca, mas quando o número de vértices e o tamanho da clique aumenta os algoritmos BBMCFM e BBMCFM2 o superam. O caso mais impressionante é na instância *gen400_p0.9-75*, onde apenas os algoritmos propostos conseguiram dentro do tempo limite e a terceira melhor solução ainda estava em subproblema de clique máxima de tamanho 66.

As instâncias da família *hamming* possuem densidades variadas, tendo o BBMC e o BBMCR como os algoritmos com melhor desempenho nessas instâncias.

Para a família *johnson* os algoritmos propostos e o BBMC apresentam um melhor desempenho. Na instância *keller4* o BBMCFM2 se destaca. E nas restantes instâncias, com exceção das instâncias *san400_0.7_1*, *san400_0.7_3* e *san1000* que o BBMCFM2 foi melhor, o BBMC possui o melhor desempenho.

Os grafos do benchmark do DIMACS são bem estruturados e complexos ficando difícil chegar a uma conclusão observando o desempenho dos algoritmos BBMCFM e BBMCFM2, contudo, podemos notar que seus números de subproblemas gerados em quase todas as instâncias foram menores, exceto em alguns casos que BBMCF se saiu melhor, chegando a uma diminuição na árvore de busca em um fator de 3 na instância *brock800_4* em relação ao BBMCF. Por isso, podemos concluir que a heurística de coloração fracionária modificada deve se comportar melhor em grafos de alta densidade por causa da sua capacidade de reduzir o número de subproblemas (podemos verificar isso nos resultados das instâncias *gen400_p0.9-75* e *C500.9* já discutidos acima). Por este motivo realizamos testes entre os algoritmos com grafos aleatórios de alta densidade.

Na Tabela 4 podemos verificar que a heurística de coloração fracionária modificada

foi melhor em 64% das instâncias testadas, comprovando que a coloração fracionária modificada realmente possui um bom desempenho em grafos com densidade alta.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nesta monografia aprendemos sobre os algoritmos para o problema da clique máxima, seus limites e estratégia de ramificação, além do uso de vetores de bits. Tivemos como objetivo apresentar um algoritmo branch-and-bound e propor uma nova heurística de coloração gulosa, a fracionária modificada, em que aumentasse o número de vértices não ramificados nos subproblemas.

Os experimentos realizados mostram que a heurística de coloração fracionária modificada apresenta um bom desempenho para grafos de alta densidade, comparado aos outros algoritmos estudados. Notamos também que a coloração fracionária modificada atinge seu objetivo de reduzir o número de subproblemas gerados, onde o mesmo superou os outros algoritmos em quase todas as intâncias. Podemos notar também, que nosso algoritmo foi capaz de resolver instâncias, tanto nos casos aleatórios como nos grafos do benchmark do DIMACS, que os outros não conseguiram dentro tempo limite.

Além da óbvia importância de aplicações práticas de algoritmos NP-difíceis rápidos, nosso trabalho tambem é interessante do ponto de vista teórico, pois usa a heurística coloração fracionária como estratégia de ramificação, uma vez que estamos cientes que outros trabalhos não apresentam tal estratégia.

Como trabalhos futuros pretendemos realizar mais testes entre os algoritmos com grafos de diferentes tamanhos e densidades, além de aumentar o tempo limite de execução, para melhores conclusões sobre o comportamento da heurística de colorção fracionária modificada.

REFERÊNCIAS

BALAS, E.; XUE, J. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. **Algorithmica**, Springer, v. 15, n. 5, p. 397–412, 1996.

BALAS, E.; YU, C. S. Finding a maximum clique in an arbitrary graph. **SIAM Journal on Computing**, SIAM, v. 15, n. 4, p. 1054–1068, 1986.

BALASUNDARAM, B.; BUTENKO, S. Graph domination, coloring and cliques in telecommunications. In: **Handbook of Optimization in Telecommunications**. [S.l.]: Springer, 2006. p. 865–890.

BALASUNDARAM, B.; BUTENKO, S.; HICKS, I. V. Clique relaxations in social network analysis: The maximum k-plex problem. **Operations Research**, INFORMS, v. 59, n. 1, p. 133–142, 2011.

BOGINSKI, V.; BUTENKO, S.; PARDALOS, P. M. Mining market data: a network approach. **Computers & Operations Research**, Elsevier, v. 33, n. 11, p. 3171–3184, 2006.

BOMZE, I. M.; BUDINICH, M.; PARDALOS, P. M.; PELILLO, M. The maximum clique problem. In: **Handbook of combinatorial optimization**. [S.l.]: Springer, 1999. p. 1–74.

BOURJOLLY P. GILL, G. L. J.-M.; MERCURE, H. An exact quadratic 0-1 algorithm for the stable set problem. In: **Cliques, coloring, and satisfiability**. [S.l.]: American Mathematical Soc., 1996. p. 53–73.

BROTCORNE, L.; LAPORTE, G.; SEMET, F. Fast heuristics for large scale covering-location problems. **Computers & Operations Research**, Elsevier, v. 29, n. 6, p. 651–665, 2002.

CARTER, M. W.; LAPORTE, G.; LEE, S. Y. Examination timetabling: Algorithmic strategies and applications. **Journal of the Operational Research Society**, JSTOR, p. 373–383, 1996.

CHEN, F.; ZHAI, H.; FANG, Y. Available bandwidth in multirate and multihop wireless ad hoc networks. **Selected Areas in Communications, IEEE Journal on**, IEEE, v. 28, n. 3, p. 299–307, 2010.

DORNDORF, U.; JAEHN, F.; PESCH, E. Modelling robust flight-gate scheduling as a clique partitioning problem. **Transportation Science**, INFORMS, v. 42, n. 3, p. 292–301, 2008.

ETZION, T.; OSTERGARD, P. R. Greedy and heuristic algorithms for codes and colorings. **Information Theory, IEEE Transactions on**, IEEE, v. 44, n. 1, p. 382–388, 1998.

GARY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-completeness**. [S.l.]: WH Freeman and Company, New York, 1979.

HÅSTAD, J. Clique is hard to approximate within n 1-&epsiv. In: IEEE. **Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on**. [S.l.], 1996. p. 627–636.

JAIN, K.; PADHYE, J.; PADMANABHAN, V. N.; QIU, L. Impact of interference on multi-hop wireless network performance. **Wireless networks**, Springer, v. 11, n. 4, p. 471–487, 2005.

KARP, R. M. Reducibility among combinatorial problems. In: **Complexity of computer computations**. [S.l.]: Springer, 1972. p. 85–103.

MALOD-DOGNIN, N.; ANDONOV, R.; YANEV, N. Maximum cliques in protein structure comparison. In: **Experimental Algorithms**. [S.l.]: Springer, 2010. p. 106–117.

MITTEN, L. Branch-and-bound methods: General formulation and properties. **Operations Research**, INFORMS, v. 18, n. 1, p. 24–34, 1970.

PATTILLO, J.; YOUSSEF, N.; BUTENKO, S. Clique relaxation models in social network analysis. In: **Handbook of Optimization in Complex Networks**. [S.l.]: Springer, 2012. p. 143–162.

RAVETTI, M. G.; MOSCATO, P. Identification of a 5-protein biomarker molecular signature for predicting alzheimer's disease. **PLoS One**, Public Library of Science, v. 3, n. 9, p. e3111, 2008.

SEGUNDO, P. S.; MATIA, F.; RODRIGUEZ-LOSADA, D.; HERNANDO, M. An improved bit parallel exact maximum clique algorithm. **Optimization Letters**, Springer, v. 7, n. 3, p. 467–479, 2013.

SEGUNDO, P. S.; RODRIGUEZ-LOSADA, D.; GALAN, R.; MATIA, F.; JIMENEZ, A. Exploiting cpu bit parallel operations to improve efficiency in search. In: IEEE. **Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on**. [S.l.], 2007. v. 1, p. 53–59.

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

SEGUNDO, P. S.; RODRÍGUEZ-LOSADA, D.; MATÍA, F.; GALÁN, R. Fast exact feature based data correspondence search with an efficient bit-parallel mcp solver. **Applied Intelligence**, Springer, v. 32, n. 3, p. 311–329, 2010.

TOMITA, E.; SEKI, T. An efficient branch-and-bound algorithm for finding a maximum clique. In: **Discrete mathematics and theoretical computer science**. [S.l.]: Springer, 2003. p. 278–289.

TOMITA, E.; SUTANI, Y.; HIGASHI, T.; TAKAHASHI, S.; WAKATSUKI, M. A simple and faster branch-and-bound algorithm for finding a maximum clique. In: **WALCOM: Algorithms and computation**. [S.l.]: Springer, 2010. p. 191–203.

WEIDE, O.; RYAN, D.; EHRGOTT, M. An iterative approach to robust and integrated aircraft routing and crew scheduling. **Computers & Operations Research**, Elsevier, v. 37, n. 5, p. 833–844, 2010.

WU, Q.; HAO, J.-K. A review on algorithms for maximum clique problems. **European Journal of Operational Research**, Elsevier, v. 242, n. 3, p. 693–709, 2015.