Um Algoritmo de Branch and Bound para o Problema da Clique Máxima Ponderada

Wladimir Araújo Tavares
Mestrado e Doutorado em Ciência da Computação, Universidade Federal do Ceará
Campus do Pici, Bloco 910, Pici, Fortaleza, Ceará, CEP 60440-900
Université d'Avignon et des Pays de Vaucluse
Laboratoire d'Informatique d'Avignon, 339, Chemin des Meinajariés, Agroparc, BP 1228, 84911
Avignon Cedex 9, France
wladimir@lia.ufc.br

Manoel Bezerra Campêlo Neto, Carlos Diego Rodrigues
Departamento de Estatística e Matemática Aplicada, Universidade Federal do Ceará
Campus do Pici, Bloco 910, Pici, Fortaleza, Ceará, CEP 60440-900
{mcampelo,diego}@lia.ufc.br

Philippe Michelon
Université d'Avignon et des Pays de Vaucluse
Laboratoire d'Informatique d'Avignon, 339, Chemin des Meinajariés, Agroparc, BP 1228, 84911
Avignon Cedex 9, France
philippe.michelon@lia.univ-avignon.fr

RESUMO

Neste artigo, apresentamos um algoritmo de branch and bound que encontra uma clique com peso máximo em um grafo ponderado arbitrário. O algoritmo combina e/ou aperfeiçoa técnicas já empregadas com sucesso como o uso da heurística de coloração ponderada como procedimento de limite superior, estratégia de ramificação baseada na coloração ponderada e o uso de operações com paralelismo bits. Resultados dos experimentos computacionais mostram que o nosso algoritmo domina o algoritmo branch and bound anterior em todas as instâncias consideradas e supera os dois algoritmos do estado da arte para o problema quando a densidade do grafo é maior ou igual a 0.6.

PALAVRAS CHAVE. Clique Máxima Ponderada, Heurística de Coloração Ponderada, Paralelismo de Bits

Área Principal: Otimização Combinatória, Teoria e Algoritmos em Grafos

ABSTRACT

In this paper, we present a branch and bound algorithm to find a maximum weight clique in an arbitrary weighted graph. The algorithm combines and/or improves techniques already employed successfully such as the weighted coloring heuristic as an upper bound procedure, branching rule based on weighted coloring and the use of bit-parallel operations. Results of computational experiments show that our algorithm dominates the previous Branch and Bound algorithm in all considered instances and outperforms the two state of the art algorithms to the problem when the graph density is greater than or equal to 0.6.

KEYWORDS. Maximum Weighted Clique. Weighted Coloring Heuristic. Bit-Parallelism

Main Area: Combinatorial Optimization, Theory and Algorithms in Graphs

1. Introdução

O problema da clique ponderada máxima (CLIQUE PONDERADA) pode ser definido da seguinte maneira: dado um grafo G com peso w(v) associado a cada vértice $v \in V$, encontre uma clique C (ou seja, um subconjunto de vértices adjacentes entre si) tal que o somatório dos pesos de C seja o maior possível. Este problema é considerado difícil, mesmo quando todos os pesos são iguais segundo Karp (1972). Este caso é conhecido como o problema da clique máxima (CLIQUE).

Além de ser teoricamente desafiador, o problema CLIQUE PONDERADA tem um grande número de aplicações práticas em diversas áreas:

- Teoria de Códigos em Brouwer et al. (1990) e Sloane (1989).
- Bioinformática e Computação Biológica em Tomita e Seki (2003) e Butenko e Wilhelm (2005).
- Visão Computacional em Hotta et al. (2003).
- Robótica em Segundo et al. (2010).
- Determinação de vencedores em leilões combinatóriais Wu e Hao (2015).

Devido a sua grande importância, existem três levantamentos bibliográficos sobre o problema (Pardalos e Xue (1994); Bomze et al. (1999); Wu e Hao (2015)). Vários heurísticas estão disponíveis para encontrar soluções subótimas para a CLIQUE PONDERADA: algoritmo aumentante de Mannino e Stefanutti (1999), algoritmo de rede neural distribuído de Bomze et al. (2000), heurística de pivoteamento complementar de Massaro et al. (2002), Phased Local Search de Pullan (2008).

Ao longo dos anos, vários algoritmos exatos foram desenvolvidos para o problema CLIQUE PONDERADA. Em Carraghan e Pardalos (1990), um algoritmo enumeração parcial simples foi apresentado. Em Balas e Xue (1991), um branch and bound eficiente foi desenvolvido usando a coloração ponderada mínima de um subgrafo cordal, fornecendo um limite inferior. Em Babel (1994), um branch and bound foi apresentado utilizando dois ingredientes principais: uma heurística de coloração ponderada baseada no DSATUR e um regra de ramificação baseada na coloração ponderada. Depois, em Balas e Xue (1996), um outro algoritmo de branch and bound foi publicado usando como procedimento de limite superior, uma heurística de coloração ponderada fracionária.

Em Ostergard (2002), um algoritmo de bonecas russas foi desenvolvido utilizando um limite superior trivial e um outro limite superior dado pelas soluções de subproblemas memorizados. Em Warren e Hicks (2007), um algoritmo híbrido combinando técnicas do algoritmo de Babel e do algoritmo Balas e Xue é apresentado. Em Kumlander (2008), um algoritmo de bonecas russas é proposto; Nele o limite superior trivial é substituído por um limite dado por uma coloração ponderada realizada uma única vez. Em Yamaguchi e Masuda (2008), apresenta-se um algoritmo de branch-and-bound que adota como limite superior - o comprimento do maior caminho em uma orientação acíclica do grafo. Em Shimizu et al. (2012), algumas modificações no algoritmo de Kumlander são apresentadas, como por exemplo, melhoria na ordem inicial dos vértices e utilização de tabelas de memorização do limite superior baseado na coloração ponderada fixa. Em Shimizu et al. (2013), o limite superior de subproblemas pequenos é calculado por programação dinâmica e os resultados são armazenados em tabelas de pré-processamento. Um algoritmo de bonecas russas é apresentado utilizando um limite superior baseado nessas tabelas.

Para grafos com densidade baixa (≤ 0.4), os melhores resultados computacionais são alcançados pelo algoritmo em Ostergaard. Para grafos com densidade média e alta (> 0.4), os melhores resultados computacionais são obtidos pelos algoritmos apresentados em Yamaguchi e Masuda (2008) e Shimizu et al. (2013). Para muitos pesquisadores, o algoritmo de Ostergaard e o algoritmo de Yamaguchi e Masuda representam o estado da arte para o problema CLIQUE PONDERADA.

Neste artigo, apresentaremos um algoritmo de branch-and-bound que combina e/ou aperfeiçoa alguns dos ingredientes utilizados nos algoritmos existentes, como segue:

- Uma ordenação inicial dos vértices conforme proposta em Ostergard (2002).
- Uso da heurística de coloração ponderada inteira de Balas como limite superior proposta por Balas e Xue (1996).
- Estratégia de ramificação inspirada em Babel (1994) com base na coloração ponderada.
- A utilização de estrutura de dados que possibilita a redução dos requisitos de memória e o aproveitamento do paralelismo de bits (Segundo et al. (2010, 2011, 2013); Corrêa et al. (2014)).

Apresentaremos também resultados de experimentos computacionais que mostram que o nosso algoritmo superior aos demais quando a densidade do grafo é maior ou igual 0.6.

Na seção 2, apresentaremos a estrutura geral do algoritmo de branch and bound para o problema CLIQUE PONDERADA. Na seção 3, apresentaremos uma versão bit-paralela da heurística de coloração ponderada, procedimento utilizado como limite superior. Na seção 4, a estratégia de ramificação com base na coloração ponderada é apresentado. Na seção 5, o algoritmo de branch-and-bound completo é exibido. Na seção 6, os resultados dos testes computacionais são analisados. Na última seção, a conclusão e as perspectivas de trabalhos futuros são destacadas.

2. Estrutura Geral

Carraghan e Pardalos apresentaram um método de enumeração parcial simples que será denominado CP. Este método pode ser utilizando como estrutura geral para apresentar os principais algoritmos existentes para CLIQUE PONDERADA. A apresentação da estrutura geral será focada nos pontos chaves que impactam no desempenho do método.

Algoritmo 1 Algoritmo proposto por Carraghan and Pardalos

Função MAIN
  $C^* \leftarrow \emptyset$
  $CP(\emptyset, V)$
Função CP(set C, set P)
  se $w(C) > w(C^*)$ **então**
    $C^* \leftarrow C$
  enquanto $P \neq \emptyset$ **faça**
    se $\boxed{w(C) + w(P) \leq w(C^*)}$ **então**
      return
    $\boxed{i \leftarrow \max\{j | v_j \in P\}}$
    CP($C \cup \{v_i\}$, $\boxed{P \cap N(v_i)}$)
    $P \leftarrow P \setminus \{v_i\}$

Durante a execução do algoritmo 1, dois conjuntos de vértices são utilizados: C (chamado clique atual) e P (chamado de conjunto candidato). O conjunto C representa a clique que está em construção e o conjunto P representa os vértices que podem entrar na clique em construção. O conjunto P é formado por todos os vértices que são conectados a todos os vértices da clique em construção. Cada vértice v de P pode ser utilizado para expandir a clique atual $C \cup \{v\}$. C^* representa a clique com o maior peso encontrada até o momento. Para $U \subseteq V, w(U) = \sum_{v \in U} w(v)$ denota o peso do conjunto U.

O algoritmo inicia com uma clique vazia ($C = \emptyset$) e um conjunto candidato P igual a V ($P = V$). Dada uma clique atual C e um conjunto de candidato P, $w(C) + w(P)$ define um limite superior trivial para a clique ponderada máxima. Logo, se $w(C) + w(P) \leq w(C^*)$ então a clique C pode ser descartada (uma poda na árvore de enumeração). Caso contrário, a sub-árvore da clique C deve continuar a ser explorada. Neste caso, precisamos definir uma estratégia de ramificação para selecionar um vértice $v \in P$ para ser ramificado. O algoritmo CP seleciona o último vértice de P para ser ramificado. A ordenação dos vértices de P segue uma ordem inicial fixa dos vértices $v_1, \ldots, v_n$ de G, onde v_1 é o vértice com menor peso em G, v_2 é o vértice com o menor peso em $G - \{v_1\}$ e assim por diante.

Os demais algoritmos de B&B propostos na literatura seguem uma estrutura similar ao Algoritmo 1. Essencialmente, eles se diferenciam em três pontos chaves, destacados em caixas, que impactam crucialmente no desempenho de cada algoritmo. São eles:

1. Procedimento de limite superior.
2. Estratégia de ramificação.
3. Estrutura de dados para a manipulação do conjunto candidato.

O primeiro ponto chave $\boxed{w(C) + w(P) \leq w(C^*)}$ está relacionado com o procedimento de limite superior. No algoritmo CP, o limite superior utilizado é o limite trivial dado pela soma dos pesos dos vértices do conjunto de candidatos. Heurísticas de coloração ponderada são freqüentemente utilizadas para a obtenção de limite superiores Balas e Xue (1991); Babel (1994); Balas e Xue (1996); Kumlander (2008); Shimizu et al. (2012).

O segundo ponto chave $\boxed{i \leftarrow \max\{j | v_j \in P\}}$ está relacionado com a estratégia de ramificação. Por exemplo, no algoritmo de CP, a estratégia de ramificação é fixa baseada em uma ordem inicial dos vértices. Na literatura, há pelos menos dois tipos de estratégias de ramificação:

1. Ramificação fixa baseada em uma ordem inicial. Carraghan e Pardalos (1990); Ostergard (2002); Kumlander (2008); Shimizu et al. (2012, 2013).
2. Ramificação dinâmica usando uma ordenação que pode mudar a cada novo conjunto candidato. Balas e Xue (1991, 1996); Babel (1994); Yamaguchi e Masuda (2008). Em Balas e Xue (1991, 1996), a maneira em que a ordenação do conjunto candidato deve ser feita não é especificada.

Apesar da estratégia de ramificação dinâmica ser mais custosa computacionalmente, bons resultados computacionais foram obtidos para o problema da CLIQUE PONDERADA, principalmente em grafos com densidade média e alta, em Babel (1994); Yamaguchi e Masuda (2008). Para o problema CLIQUE, bons resultados computacionais foram obtidos nos algoritmos MCQ (Tomita e Seki (2003)), MCR (Tomita e Kameda (2007)), MCS (Tomita et al. (2010)), BBMAXCLIQUE (Segundo et al. (2010)) e BBMCI/BBMCR (Segundo et al. (2013)) utilizando uma estratégia de ramificação dinâmica.

O terceiro ponto chave $\boxed{P \cap N(v)}$ está relacionado com a estrutura de dados que armazena o grafo e a eficiência das operações realizadas sobre G durante o algoritmo. Mais recentemente, a utilização de estrutura de dados que armazena bits individualmente (*bitmaps*) começou a ser explorada (Segundo et al. (2010, 2011, 2013); Corrêa et al. (2014)) para o problema CLIQUE. A utilização dos *bitmaps* possibilita a redução dos requisitos de memória e uma melhor aproveitamento do paralelismo das operações bit-a-bit realizada pelo hardware. O paralelismo de bits ainda não foi aproveitado para o problema CLIQUE PONDERADA.

O algoritmo que apresentamos nesse trabalho também segue a estrutura do Algoritmo 1. Nas próximas seções, mostramos nossas escolhas e implementações para cada um dos três pontos chaves apontados acima.

3. Limite Superior

Os principais algoritmos de branch-and-bound tentam obter o melhor compromisso entre a qualidade do limite superior e o tempo gasto para calculá-lo. Por exemplo, o limite superior proposto por Kumlander tem uma qualidade inferior ao de Balas, porém demanda esforço computacional bem menor. Confiando na qualidade do limite de Balas, propomos o uso do paralelismo de bits como forma de reduzir seu tempo de cálculo. Nesta seção, apresentaremos a versão bit-paralela da heurística de Balas.

Seja S a coleção de todos os conjuntos independentes de G. O problema CLIQUE PONDERADA pode ser formulado da seguinte maneira:

$$\max \sum_{v \in V} w(v)x(v) \tag{1}$$

$$\text{subject to} \sum_{v \in S} x(v) \leq 1, \quad \forall S \in \mathcal{S} \tag{2}$$

$$x(v) \in \{0,1\} \quad \forall v \in V \tag{3}$$

Uma solução viável define uma clique C em G tal que se $x(v) = 1$ então o vértice v está na clique, $x(v) = 0$, caso contrário.

Para obter um limite superior, podemos recorrer ao seu dual, o problema da coloração ponderada mínima:

$$\min \sum_{S \in \mathcal{S}} y(S) \tag{4}$$

$$\text{subject to} \sum_{\{i|v \in S_i\}} y(S_i) \geq w(v), \quad \forall v \in V \tag{5}$$

$$y(S) \geq 0 \qquad \forall S \in \mathcal{S} \tag{6}$$

Cada conjunto independente S representa uma classe de cor. Uma solução viável define uma coloração ponderada $(y, \mathcal{S})$, cada classe de cor $S \in \mathcal{S}$ tem um peso inteiro não-negativo $y(S)$, tal que para cada vértice v, a soma dos pesos das classes de cores contendo v seja pelo menos igual ao peso de v. O peso da coloração é dado pela soma dos pesos das classes de cores.

Quando exigimos $y(S) \in \mathbb{Z}$ em (6), definimos o problema da coloração ponderada mínima inteira.

Observe que o problema da coloração ponderada mínima tem um número exponencial de variáveis. Na verdade, resolver este problema é NP-difícil. Entretanto, qualquer solução viável é um limite superior para a clique ponderada máxima.

Para encontrar uma solução viável, a heurística de coloração ponderada de Balas encontra uma coleção de conjunto independentes $S = \{S_1, \ldots, S_k\}$ tal que

$$\sum_{\{i|v \in S_i\}} y(S_i) \geq w(v) \tag{7}$$

As classes de cores são obtidas sequencialmente de maneira gulosa. Para cada vértice v, a variável $res(v)$ mantém o peso residual para colorir o vértice v. Inicialmente, o valor de $res(v)$ será igual a $w(v)$. A cada iteração da heurística, a seguinte invariante é satisfeita:

$$res(v) + \sum_{\{i|v \in S_i\}} y(S) = w(v) \tag{8}$$

A classe de cor definida na iteração i é um conjunto independente maximal S_i no subgrafo induzido por $\{v \in V | res(v) > 0\}$, a quem atribui-se o peso $y(S_i)$ dado pelo menor peso residual de um vértice em S_i.

No Algoritmo 2, propomos uma implementação, utilizando paralelismo de bits, para a heurística de Balas. O procedimento recebe um bitmap U representando um conjunto de vértices U. Durante o algoritmo, os vértices que são coloridos pela heurística são removidos do bitmap U. O bitmap Q representa os vértices que podem ser adicionados a uma classe de cor S_i em construção.

Inicialmente, o bitmap Q será inicializado com os vértices do bitmap U. Quando um vértice v de Q é escolhido para ser inserido em S_i, a classe de cor S_i é atualizada $S_i \leftarrow S_i \cup \{v\}$. Os vértices adjacentes a v e o próprio v são removidos do bitmap Q, ou seja, $Q \leftarrow Q \setminus N(v); Q \leftarrow Q \setminus \{v\}$. O processo continua enquanto ainda existir um vértice $v \in Q$. Quando Q torna-se vazio, uma classe de cor maximal S_i é obtida. O peso da classe de cor S_i é dado pelo menor peso residual dos vértices de S_i. O peso residual de cada vértices de S_i é atualizado, decrementando pelo peso da classe de cor S_i.Os vértices com peso residual iguais a zero podem ser removidos de U. A heurística de coloração ponderada termina quando U torna-se vazio. As instruções em caixas são instruções que são substituídas por instruções específicas que manipulam diretamente o bitmap. As operações sobre bitmaps que serão utilizadas no nosso algoritmos são:

Operação bitmap	Descrição
SET(B,v)	adiciona o elemento $v < n$ no bitmap B
RESET(B, v)	remove o elemento $v < n$ do bitmap B
TEST(B, v)	retorna 1 se $v < n$ está no bitmap B, 0 caso contrário
LSB(B, i, n)	retorna o menor elemento em $\{w*i, \ldots, n-1\} \cap B$, onde w é o maior número que os compiladores atuais podem tratar (tipicamente 64). O valor de retorno corresponde índice do bit menos significativo a partir do bit $w * i$ no bitmap B
INTER(A,B,B',n)	a intersecção dos dois bitmaps B e B' é computado utilizando o operador bit-a-bit AND e armazenado no bitmap A.
DIFF(A,B,B',n)	a diferença $B \setminus B'$ de dois bitmaps utilizando o operador bit-a-bit AND e o operador bit-a-bit NOT é computado e armazenado no bitmap A.

Para maiores detalhes sobre a representação e a implementação das operações sobre bitmap, nós recomendamos os artigos Segundo et al. (2010, 2011, 2013); Corrêa et al. (2014). Os três artigos são especializados para o problema CLIQUE. Os dois primeiros artigos desenvolvem um algoritmo de branch-and-bound. O último apresenta um algoritmo de boneca russas. Todos os algoritmos recomendados usam o paralelismo de bits.

Algoritmo 2 Heurística de Coloração Ponderada

Função COLORAÇÃO(U, S, y)
Entrada: bitmap U, Classes de cor $S_1, \ldots, S_k$ e Pesos das classes de cor $y(S_1), \ldots, y(S_k)$
$\forall v \in V, res(v) \leftarrow w(v)$
$i \leftarrow 1$
enquanto $U \neq \emptyset$ **faça**
$\boxed{S_i \leftarrow \emptyset}$; $\boxed{Q \leftarrow U}$
enquanto $Q \neq \emptyset$ **faça**
$\boxed{\text{v} \leftarrow \text{Selecione o primeiro elemento de Q.}}$
$\boxed{Q \leftarrow Q \setminus N(v)}$; $\boxed{Q \leftarrow Q \setminus \{v\}}$; $S_i \leftarrow S_i \cup \{v\}$
$y(S_i) \leftarrow min\{res(v) | v \in S_i\}$
para $v \in S_i$ **faça**
$res(v) \leftarrow res(v) - y(S_i)$
$\boxed{U \leftarrow U \setminus \{v \in S_i \quad | \quad res(v) = 0\}}$
$i \leftarrow i + 1$

Um passo importante no Algoritmo 2 é a escolha do vértice $v \in Q$. Tal escolha segue a ordem inicial dos vértices. Usaremos a ordem inicial proposta por Ostergaard em seu algoritmo de bonecas russas para o problema CLIQUE PONDERADA. Os vértices $v_1, \ldots, v_n$ de G são ordenados no bitmap de tal maneira que v_1 é o vértice com menor peso em G. Em caso de empate, escolhe-se o vértice com a maior vizinhança ponderada em G. O vértice v_2 é escolhido de forma análoga em $G - \{v_1\}$ e assim por diante. Intuitivamente, o vértice com o menor índice em um bitmap será o vértice com o menor peso e mais restrições de coloração (maior vizinhança ponderada). Note que para este vértice existe potencialmente um número menor de classes de cores possível. Dessa maneira, a heurística de coloração tem maior chance de obter uma coloração ponderada mais apertada.

4. Estratégia de Ramificação

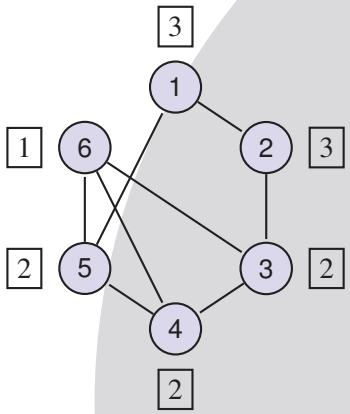
A partir das classes de cores e pesos gerados pela heurística de coloração podemos definir a ordem de ramificação dos vértices e determinar um limite superior para o conjunto de vértices candidato corrente. Sejam $S_1, \ldots, S_k$ a lista de classes de cores e $y(S_1), \ldots, y(S_k)$ os pesos obtidos pela heurística de coloração ponderada. Seja $s(v)$ o índice da última classe na ordem dada pela coloração ponderada a que v pertence, ou seja, $s(v)$ é o maior $j \in \{1, \ldots, k\}$ tal que $v \in S_j$. Seja $\sigma = (v_1, \ldots, v_n)$ uma ordenação dos vértices tal que $s(v_1) \leq s(v_2) \leq \ldots \leq s(v_n)$.

Seja $color(v_i)$ o somatório dos pesos das classes de cor até aquela com índice $s(v_i)$. Precisamente,

$$color(v_i) = \sum_{j=1}^{s(v_i)} y(S_j) \tag{9}$$

Observe que $color(v_1) \leq color(v_2) \leq \ldots \leq color(v_n)$. Note que $color(v_i)$ é o peso da coloração ponderada restrita ao subgrafo $G(\{v_1, \ldots, v_i\})$. Por sua vez, $color(v_i)$ é um limite superior para a clique máxima ponderada do grafo $G(\{v_1, \ldots, v_i\})$.

Considere o seguinte exemplo:



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

1. $S_1 = \{2, 6\}, y(S_1) = 1$
2. $S_2 = \{3, 5\}, y(S_2) = 2$
3. $S_3 = \{2, 4\}, y(S_3) = 2$
4. $S_4 = \{1\}, y(S_4) = 3$

Na tabela seguinte, o valor de $s(v)$ para cada vértice v é mostrado:

v	6	5	4	3	1	2
$s(v)$	1	2	3	2	4	3

Observe que a ordem em que os vértices são efetivamente coloridos não é exatamente igual à ordem inicial dos vértices. Na próxima tabela, o valor de $color(v)$ é apresentado seguindo a ordem σ:

σ	6	3	5	4	2	1
color(v)	1	3	3	5	5	8

Tanto a ordem em que os vértices são coloridos σ como o vetor color podem ser obtidos de maneira iterativa. Sabemos que um vértice v foi totalmente colorido quando $sum(v) = \sum_{\{i \leq j | v \in S_i\}} y(S_i) = w(v)$. Em outras palavras, seguindo a ordem das classes de cores, assim que $sum(v) = w(v)$, sabemos que o vértice v não vai aparecer em nenhum outra classe de cor seguinte. No Algoritmo 3 obtemos σ e $color(v)$. Na verdade, esse procedimento pode ser incorporado ao Algoritmo 2, onde podemos identificar o vértice v como colorido assim que $res(v) = 0$.

Algoritmo 3 Estratégia de Ramificação

Função RAMIFICAÇÃO(S, y, σ, color, cont)
Entrada: Classes de cor $S_1, \ldots, S_k$,
Pesos inteiros não negativos $y(S_1), \ldots, y(S_k)$,
Ordem dos vértices σ,
Limite Superior color,
cont número de vértices.

```
para v ∈ V faça
    sum(v) ← 0
soma ← 0
cont ← 0
para i ∈ {1, .., |S|} faça
    soma ← soma + y(S_i)
    para v ∈ S_i faça
        sum(v) ← sum(v) + y(S_i)
        se sum(v) = w(v) então
            σ(cont) ← v
            color(v) ← soma
            cont ← cont + 1
```

5. Branch And Bound

Nesta seção, apresentamos um algoritmo B&B para o problema CLIQUE PONDERADA denominado BPBBCP(Bit-Paralelo Branch And Bound para CLIQUE PONDERADA). Ele combina de forma efetiva vários elementos que já foram utilizados em outros algoritmos de Branch and Bound e Bonecas Russas. Mais precisamente, usamos:

- Ordenação inicial dos vértices usada no algoritmo de Bonecas Russa de Ostergaard Ostergard (2002).
- Limite Superior dado por uma implementação bitmap da heurística de coloração ponderada inteira proposta por Balas e Xue (1996)
- Estratégia de ramificação baseada na coloração, porém inspirada em Babel (1994).

- Estrutura de dados que possibilita a redução dos requisitos de memória e o aproveitamento do paralelismo de bits, aprimorando a implementação de Segundo et al. (2010, 2013) para o problema CLIQUE.

Observou-se empiricamente que, no nó raiz da árvore de busca, a estratégia de ramificação seguindo a ordenação inicial dos vértices obtém um resultado melhor que a ordenação dada pela heurística de coloração. Dessa maneira, a heurística de coloração ponderada não é utilizada no nó raiz e o limite superior trivial será utilizado.

Algoritmo 4 Algoritmo BBMWCP

Função MAIN
Ordenação inicial dos vértices proposta por Ostergaard Ostergard (2002).
para i $\leftarrow$ 1 até $|V|$ **faça**
$\sigma(i) \leftarrow v_i$
$color(v_i) \leftarrow \sum_{j=1}^{i} w(v_i)$
$C^* \leftarrow \emptyset$
BPBBCP($\emptyset$, V, σ, $color$, $|V|$)

Função BPBBCP(C, U, σ, $color$, $cont$)
se $w(C) > w(C^*)$ **então**
$C^* \leftarrow C$
$peso \leftarrow w(C)$
para i $\leftarrow$ $cont - 1$ até 0 **faça** ▷ ordem reversa
$v \leftarrow \sigma(i)$
se $peso + color[v] \leq w(C^*)$ **então**
return
$U' \leftarrow U \cap N(v)$
$U \leftarrow U \setminus \{v\}$;
COLORAÇÃO(U', S, y)
RAMIFICAÇÃO(S, y, σ', $color'$, $cont'$)
BPBBCP($C \cup \{v\}$, U', σ', $color'$, $cont'$)

6. Resultados computacionais

Todos os testes foram conduzidos num computador com processador Intel Core i7-2600K 3.40Ghz, 8 Mb de cache, com 6Gb de memória, utilizando sistema operacional Linux. Nós comparamos o desempenho computacional do algoritmo BPBBCP com o algoritmo de Ostergaard obtido eletronicamente em (http://tcs.legacy.ics.tkk.fi/ pat/wclique.html) e o algoritmo de Yamaguchi e Masuda (YM) disponibilizado gentilmente pelos autores.

Nós vamos avaliar o desempenho do algoritmo BCBBCP utilizando as instâncias de grafos aleatórios.

Para cada combinação n (número de vértices) e p (probabilidade de existência de cada aresta), 10 grafos aleatórios foram gerados. Os pesos atribuídos aos vértices variam entre 1 e 10. Como é usual na literatura Yamaguchi e Masuda (2008); Ostergard (2002), escolhemos valores de p menores para grandes valores de n. Consideramos 34 combinações (n,p), levando a 340 instâncias no total. Para cada par (n,p), calculamos a média do tempo consumido pelas 10 instâncias por cada um dos três algoritmos. Os resultados estão apresentados na Tabela 1.

Quando o tempo de resolução de uma instância ultrapassa o tempo limite de 2h (7200s), a execução dos outros problemas relativos ao mesmo par é suspensa. Tal ocorrência é assinalada como * na tabela. Identificamos em negrito o melhor desempenho médio em cada caso.

Instancia	Numero de Subproblemas $\times 10^{-3}$			Tempo(segundos)			Problemas Resolvidos	Gap Médio
(n,p)	Ostergaard	Yamagucchi	BPBBCP	Ostergaard	Yamagucchi	BPBBCP	BPBBCP	BPBBCP
(5000,0.10)	3.68×10^{2}	8.80×10^{2}	$\mathbf{2.82}\times10^{0}$	**0.52**	2.69	1.80	10	0.00
(10000,0.10)	4.88×10^{3}	4.64×10^{3}	$\mathbf{3.78}\times10^{1}$	**4.66**	35.96	26.82	10	0.00
(15000,0.10)	2.32×10^{4}	1.17×10^{4}	$\mathbf{1.64}\times10^{2}$	**18.99**	213.05	130.53	10	0.00
(5000,0.20)	1.25×10^{4}	7.17×10^{3}	$\mathbf{2.16}\times10^{2}$	**7.86**	77.54	36.02	10	0.00
(10000,0.20)	3.17×10^{5}	4.98×10^{5}	$\mathbf{2.58}\times10^{3}$	**174.55**	1965.02	1563.41	10	0.00
(15000,0.20)	$\mathbf{2.28}\times10^{6}$	*	*	**1340.81**	*	*	0	0.00
(4000,0.30)	1.61×10^{5}	1.15×10^{5}	$\mathbf{2.49}\times10^{3}$	**90.29**	670.90	296.65	10	0.00
(5000,0.30)	4.78×10^{5}	3.09×10^{5}	$\mathbf{7.54}\times10^{3}$	**291.64**	2590.65	1096.90	10	0.00
(7500,0.30)	$\mathbf{4.36}\times10^{6}$	*	*	**2611.79**	*	*	0	0.00
(2000,0.40)	1.23×10^{5}	4.51×10^{4}	$\mathbf{1.66}\times10^{3}$	**61.84**	291.78	99.62	10	0.00
(3000,0.40)	1.29×10^{6}	5.91×10^{5}	$\mathbf{1.60}\times10^{4}$	**743.33**	3897.02	1507.43	10	0.00
(4000,0.40)	$\mathbf{7.10}\times10^{6}$	*	*	**4585.31**	*	*	0	0.002
(1000,0.50)	5.11×10^{4}	1.63×10^{4}	$\mathbf{6.71}\times10^{2}$	24.05	63.56	**21.17**	10	0.00
(1500,0.50)	6.43×10^{5}	1.73×10^{5}	$\mathbf{7.19}\times10^{3}$	**335.78**	1187.45	336.40	10	0.00
(2000,0.50)	5.00×10^{6}	*	$\mathbf{5.29}\times10^{4}$	**2827.01**	*	3101.54	10	0.00
(2250,0.50)	1.05×10^{7}	*	*	**6247.56**	*	*	1	0.00
(900,0.60)	8.17×10^{5}	1.80×10^{5}	$\mathbf{8.05}\times10^{3}$	360.74	854.22	**221.86**	10	0.00
(1000,0.60)	2.05×10^{6}	4.56×10^{5}	$\mathbf{1.94}\times10^{4}$	944.70	2233.14	**574.88**	10	0.00
(1200,0.60)	7.73×10^{6}	*	$\mathbf{7.36}\times10^{4}$	3747.19	*	**2638.68**	10	0.00
(500,0.70)	5.77×10^{5}	7.97×10^{4}	$\mathbf{3.87}\times10^{3}$	188.29	230.21	**54.38**	10	0.00
(600,0.70)	2.44×10^{6}	2.91×10^{5}	$\mathbf{1.46}\times10^{4}$	855.19	1334.16	**260.83**	10	0.00
(700,0.70)	1.11×10^{7}	*	$\mathbf{5.99}\times10^{4}$	4162.69	*	**1233.16**	10	0.00
(750,0.70)	*	*	$\mathbf{1.07}\times10^{5}$	*	*	**2395.50**	10	0.00
(800,0.70)	*	*	$\mathbf{1.81}\times10^{5}$	*	*	**4368.12**	10	0.00
(250,0.80)	1.06×10^{5}	7.42×10^{3}	$\mathbf{3.83}\times10^{2}$	22.66	11.55	**2.49**	10	0.00
(300,0.80)	6.63×10^{5}	3.73×10^{4}	$\mathbf{1.97}\times10^{3}$	160.88	86.75	**16.01**	10	0.00
(350,0.80)	2.98×10^{6}	1.47×10^{5}	$\mathbf{7.47}\times10^{3}$	761.43	564.75	**71.51**	10	0.00
(400,0.80)	1.16×10^{7}	5.56×10^{5}	$\mathbf{2.98}\times10^{4}$	3238.24	2895.00	**326.93**	10	0.00
(450,0.80)	*	*	$\mathbf{1.09}\times10^{5}$	*	*	**1320.44**	10	0.00
(500,0.80)	*	*	$\mathbf{3.46}\times10^{5}$	*	*	**4813.08**	10	0.00
(150,0.90)	1.01×10^{5}	2.16×10^{3}	$\mathbf{1.16}\times10^{2}$	14.13	2.21	**0.36**	10	0.00
(200,0.90)	2.79×10^{6}	1.80×10^{4}	$\mathbf{1.30}\times10^{3}$	464.92	73.44	**6.33**	10	0.00
(250,0.90)	*	4.74×10^{5}	$\mathbf{3.12}\times10^{4}$	*	3406.32	**187.87**	10	0.00
(300,0.90)	*	*	$\mathbf{2.77}\times10^{5}$	*	*	**2115.77**	10	0.00

Tabela 1: Tabela dos resultados dos experimentos computacionais em grafos aleatórios

A Tabela 1 mostra que o algoritmo BPBBCP domina completamente o algoritmo Yamaguchi em todos os casos analisados e domina os dois algoritmos somente em grafos com densidade ≥ 0.6. Para a combinação (250,0.9), BPBBCP chega a ser 18 vezes mais rápido que o algoritmo Yamaguchi. Além disso, o algoritmo BPBBCP resolve instâncias dentro do tempo limite estipulado que os outros dois algoritmos não resolvem.

Para instâncias com baixa densidade (inferior ou igual 0.5), a redução no número de subproblemas não foi suficiente para compensar a utilização de um limite superior com uma qualidade melhor, mas com um esforço computacional maior, como o limite de Balas e Yamaguchi.

7. Conclusão e Trabalhos Futuros

Neste artigo, apresentamos um novo algoritmo de Branch And Bound combinando e/ou melhorando várias técnicas já empregadas com sucesso como uso de uma heurística de coloração ponderada, uma estratégia de ramificação baseada na coloração e o paralelismo de bits. Os resultados computacionais mostram que essa combinação de técnicas resulta em uma redução no número de subproblemas. O algoritmo BPBBCP domina o algoritmo de Branch-And-Bound de Yamaguchi & Masuda e supera os dois algoritmos comparados quando a densidade do grafo aleatório é superior a 0.6.

Em Corrêa et al. (2014), um algoritmo de Bonecas Russas é apresentado para o problema CLIQUE. Tal algoritmo supera os demais algoritmos do estado da arte. O autor argumenta que o método das Bonecas Russas pode constituir uma alternativa interessante ao método de Branch and Bound para alguns problemas de otimização combinatória. Como trabalho futuro, desenvolveremos

um algoritmo das Bonecas Russas para o CLIQUE PONDERADA utilizando técnicas proposta neste trabalho.

Referências

Babel, L. (1994). A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38.

Balas, E. and Xue, J. (1991). Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs. *SIAM J. Comput.*, 20(2):209–221. Addendum: SIAM J. Comput. 21(5): 1000 (1992).

Balas, E. and Xue, J. (1996). Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica*, 15(5):397–412.

Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). The maximum clique problem. *Handbook of Combinatorial Optimization (Supplement Volume A)*, 4:1–74.

Bomze, I. M., Pelillo, M., and Stix, V. (2000). Approximating the maximum weight clique using replicator dynamics. *IEEE Trans. Neural Networks*, 11:1228–1241.

Brouwer, A. E., Shearer, L. B., Sloane, N. I. A., Warren, and Smith, D. (1990). A new table of constant weight codes. *IEEE Trans Inform Theory*.

Butenko, S. and Wilhelm, W. E. (2005). Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173:1–17.

Carraghan, R. and Pardalos, P. M. (1990). A parallel algorithm for the maximum weight clique problem. Technical report, Dept. of Computer Science, Pennsylvania State University.

Corrêa, R. C., Michelon, P., Cun, B. L., Mautor, T., and Donne, D. D. (2014). A bit-parallel russian dolls search for a maximum cardinality clique in a graph. *CoRR*, abs/1407.1209.

Hotta, K., Tomita, E., and Takahashi, H. (2003). A view-invariant human face detection method based on maximum cliques. *Trans. IPSJ*, 44(SIG14 (TOM9):57–70.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103.

Kumlander, D. (2008a). On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. In An, L. T. H., Bouvry, P., and Dinh, T. P., editors, *MCO*, volume 14 of *Communications in Computer and Information Science*, pages 165–174. Springer.

Kumlander, D. (2008b). On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. In Le Thi, H., Bouvry, P., and Pham Dinh, T., editors, *Modelling, Computation and Optimization in Information Systems and Management Sciences*, volume 14 of *Communications in Computer and Information Science*, pages 165–174. Springer Berlin Heidelberg.

Mannino, C. and Stefanutti, E. (1999). An augmentation algorithm for the maximum weighted stable set problem. *Computational Optimization and Applications*, 14(3):367–381.

Massaro, A., Pelillo, M., Immanuel, and Bomze, M. (2002). A complementary pivoting approach to the maximum weight clique problem. *SIAM J. Optim*, 12:928–948.

Ostergard, P. R. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120:197–207. Special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

Pardalos, P. M. and Xue, J. (1994). The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328.

Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14(2):117–134.

Segundo, P. S., Matia, F., Rodriguez-Losada, D., and Hernando, M. (2013). An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, 7(3):467–479.

Segundo, P. S., Rodriguez-Losada, D., and Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers OR*, 38(2):571–581.

Segundo, P. S., Rodriguez-Losada, D., Matia, F., and Galan, R. (2010). Fast exact feature based data correspondence search with an efficient bit-parallel mcp solver. *Appl. Intell.*, 32(3):311–329.

Shimizu, S., Yamaguchi, K., Saitoh, T., and Masuda, S. (2012). Some improvements on kumlander-s maximum weight clique extraction algorithm. volume 6, pages 1770 – 1774. International Conference on Electrical, Computer, Electronics and Communication Engineering (ICECECE 2012).

Shimizu, S., Yamaguchi, K., Saitoh, T., and Masuda, S. (2013). *Optimal Table Method for Finding the Maximum Weight Clique*. Proceedings of the 13th International Conference on Applied Computer Science (ACS'13).

Sloane, N. J. A. (1989). Unsolved problems in graph theory arising from the study of codes. In *in Graph Theory Notes of New York 18*, pages 11–20.

Tomita, E. and Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optimization*, 37(1):95–111.

Tomita, E. and Seki, T. (2003). An efficient branch-and-bound algorithm for finding a maximum clique. In *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science*, DMTCS'03, pages 278–289, Berlin, Heidelberg. Springer-Verlag.

Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., and Wakatsuki, M. (2010). A simple and faster branch-and-bound algorithm for finding a maximum clique. In *WALCOM*, pages 191–203.

Warren, J. S. and Hicks, I. V. (2007). Combinatorial branch-and-bound for the maximum weight independent set problem. Technical report, Texas A&M University.

Wu, Q. and Hao, J. (2015a). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709.

Wu, Q. and Hao, J. (2015b). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Syst. Appl.*, 42(1):355–365.

Wu, Q., Hao, J., and Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals OR*, 196(1):611–634.

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