BITCLIQUE: um algoritmo de Branch-and-Bound para o problema da clique máxima ponderada

Wladimir Araújo Tavares
Curso de Ciência da Computação, Universidade Federal do Ceará
Campus de Quixadá, Quixadá, Ceará, CEP 63900-000
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, Agroparc, 84911 Avignon Cedex 9, France
philippe.michelon@lia.univ-avignon.fr

RESUMO

Nós apresentamos um algoritmo de branch-and-bound, chamado BITCLIQUE, para o problema da clique máxima ponderada. Ele depende de uma ordem inicial dos vértices. Duas ordens são consideradas, em função dos pesos dos vértices ou dos pesos de seus vizinhos, levando a duas versões do algoritmo. Ele efetivamente combina adaptações de várias ideias já empregadas com sucesso para resolver o problema, como a utilização de uma heurística de coloração ponderada para definir regras de poda e ramificação, e o uso do paralelismo de bits para simplificar as operações realizadas no grafo. O algoritmo proposto supera os principais algoritmos do estado da arte na maioria das instâncias consideradas em termos de números de subproblemas enumerados quanto no tempo de computação demandado.

PALAVRAS CHAVE. Clique Máxima Ponderada, Heurística de Coloração Ponderada, Paralelismo de Bits
Área Principal: Otimização Combinatória, Teoria e Algoritmos em Grafos

ABSTRACT

We present a new branch-and-bound algorithm, called BITCLIQUE, for the maximum weight clique problem. It depends on an given ordering of vertices. Two orderings are considered, as functions of the weight of each vertex or the sum of the weights of its adjacent vertices. This leads to two versions of the algorithm. It effectively combines adaptations of several ideas already successfully employed to solve the problem, such as the use of a weighted integer coloring heuristic for pruning and branching purposes, and the use of bit vectors for simplifying the operations on the graph. The proposed algorithm outperforms state-of-art branch-and-bound algorithms in most of the considered instances in terms of the number of enumerated subproblems as well in terms of computational time.

KEYWORDS. Maximum Weighted Clique. Weighted Coloring Heuristic. Bit-Parallelism
Main Area: Combinatorial Optimization, Theory and Algorithms in Graphs

1. Introdução

O problema da clique máxima ponderada (PCMP) é uma generalização do problema da clique de máxima(PCM), onde pesos positivos inteiros são associados aos vértices. Formalmente, dado um grafo ponderado $G = (V, E, w)$, onde $w : V \leftarrow \mathbb{Z}^+$ é uma função de ponderação, PCMP consiste em encontrar uma clique com o maior peso, onde o peso de uma clique C é definido como $w(C) = \sum_{v \in C} w(v)$.

Além de ser teoricamente desafiador, PCMP possui um grande número de aplicações práticas em várias áreas: teoria da codificação, bioinformática, visão computacional, robótica, etc. Uma lista abrangente de aplicações do PCMP pode ser encontrada em [Pardalos and Xue(1994), Bomze et al.(1999), Wu and Hao(2014)]. Outra importante aplicação surge no processo de geração de colunas para problemas de coloração de grafos [Mehrotra and Trick(1996), Gualandi and Malucelli(2012), Held et al.(2012)] e em geração de plano de cortes em programação inteira [Corrêa et al.(2015)]. Estas duas últimas aplicações demandam a necessidade da resolução de sucessivas instâncias de PCMP.

Alguns procedimentos foram desenvolvidos para resolver exatamente PCMP usando técnicas de programação inteira [Rebennack et al.(2011), Warrier et al.(2005)] ou métodos combinatórios [Balas and Xue(1991), Babel(1994), Balas and Xue(1996), Östergård(2002), Yamaguchi and Masuda(2008),Kumlander(2008),Held et al.(2012)]. Em [Rebennack et al.(2011)], encontramos uma lista de desigualdades válidas e algoritmos especializados para separá-las. Essas rotinas de separações levam ao desenvolvimento de vários algoritmos de branch-and-cut. Porém, o tempo exigido para executar as rotinas de separação e a resolução das relaxações lineares nem sempre justificam a melhoria dos limites superiores provido pela inclusão dos cortes encontrados. Os testes computacionais realizados em [Held et al.(2012)] mostraram que os algoritmos de branch-and-cut dos resolvedores CPLEX e Gurobi não são mais eficientes que os algoritmos combinatórios, exceto para grafos muito densos (densidade superior a 97%).

Neste artigo, apresentamos um algoritmo para PCMP chamado BITCLIQUE. Ele combina adaptações de várias idéias aplicadas com sucesso para resolver PCMP, como o uso de uma heurística de coloração ponderada inteira para a poda e a ramificação, e o uso de vetores de bits para simplificar operações realizadas sobre o grafo. O algoritmo proposto supera os algoritmos do estado da arte na maioria das instâncias de grafos aleatórios, DIMACS-W e EXACTCOLOR em relação ao número de subproblemas enumerados como em termos de tempo de execução.

O restante do texto está organizado da seguinte maneira. Na seção 2, apresentamos algumas notações importantes que serão utilizadas ao longo do texto. Na seção 3, faremos uma breve apresentação dos algoritmos do estado da arte. Na seção 4, apresentamos a estrutura geral dos algoritmos de branch-and-bound apresentados na literatura. Na seção 5, a estratégia de limite de superior baseada na coloração ponderada é exibida. Na seção 6, a estratégia de ramificação baseada na coloração ponderada é exposta. Na seção 7, o algoritmo BITCLIQUE é finalmente apresentado. Na seção 8, os resultados dos testes computacionais realizados são mostrados.

2. Preliminares

Um **grafo** G é par ordenado (V, E) composto por um conjunto finito V, cujos elementos são denominados **vértices**, e por um conjunto $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$, cujos elementos são denominados **arestas**. Para todo grafo G, denotamos $V(G)$ e $E(G)$, respectivamente, os conjuntos de vértices e arestas de G.

Um grafo **ponderado**, denotado por (V, E, w) ou (G, w), é composto por um grafo G e uma função de ponderação dos vértices $w : V \rightarrow \mathbb{R}^+$. O peso de um vértice v, denotado por $w(v)$, é o peso associado ao vértice v pela função w. O **peso** de um conjunto $S \subseteq V$, denotado $w(S)$, representa a soma dos pesos de todos os vértices de S.

As seguintes notações serão utilizadas neste artigo:

- $N(v) = \{u \in V : (v,u) \in E\}$ é a vizinhança de um vértice v em G formada pelos vizinhos de v.
- Se $U \subseteq V$, então $G[U] = (U, E[U])$ denota o subgrafo de G induzido por U, onde $E[U] = \{\{u,v\} : u,v \in U\}$
- S é um conjunto independente de G se somente se $\forall u,v \in S, \{u,v\} \notin E(G)$.
- O par $(\mathbb{S}, y)$ representa uma coloração ponderada inteira de $G = (V, E, w)$, composto por uma coleção de conjuntos independentes $\mathbb{S} = \{S_1, S_2, \ldots, S_k\}$ e uma função de ponderação dos conjuntos independentes $y : S \to \mathbb{Z}$ satisfazendo a seguinte propriedade $\sum_{S \in \mathbb{S}: v \in S} y(S) \geq w(v) \quad \forall v \in V$
- $y(\mathbb{S})$ denota o peso de uma coloração ponderada $(\mathbb{S}, y)$, é calculado pelo somatório dos pesos dos conjuntos independentes, ou seja, $y(\mathbb{S}) = \sum_{S \in \mathbb{S}} y(S)$
- $\omega(G, w)$ denota o peso da clique mais pesada de $G = (V, E, w)$.
- $\overrightarrow{G} = (V, A))$ representa um **grafo direcionado** composto por um conjunto finito V e por um conjunto $A \subseteq \{(u,v) : u,v \in V, u \neq v\}$, cujos elementos são denominados **arcos**.
- $\overrightarrow{G}_\rho = (V, A)$ representa o **grafo direcionado acíclico induzido por** ρ, onde $\rho = [\rho_1, \ldots, \rho_{|V|}]$ é uma ordem inicial dos vértices. O conjunto de arcos A é obtido trocando cada aresta $\{\rho_i, \rho_j\} \in E$ com $i < j$ por um arco de ρ_i para ρ_j. Logo, $A = \{(\rho_i, \rho_j) : \{\rho_i, \rho_j\} \in E, i < j\}$
- $\ell(\overrightarrow{G}, w, v)$ denota o peso do caminho mais pesado no grafo $\overrightarrow{G}$ terminado no vértice v, onde o peso de caminho é dado pela soma dos pesos dos seus vértices.
- $\ell(\overrightarrow{G}, w)$ é o peso do caminho mais peso de $\overrightarrow{G}$, obtido por $\max_{v \in V} \ell(\overrightarrow{G}, w, v)$.

Podemos estabelecer facilmente que o peso de uma coloração ponderada $(\mathbb{S}, y)$ de um grafo ponderado (G, w) é um limite superior para $\omega(G, w)$.

Proposição 1 *[Balas and Xue(1991)] Seja $(\mathbb{S}, y)$ uma coloração ponderada para (G, w) então $y(\mathbb{S}) \geq \omega(G, w)$.*

Prova Seja K uma clique ponderada máxima de (G, w). Por definição, temos que $\forall S_i \in \mathbb{S}, |K \cap S_i| \leq 1$. Logo,

$$y(\mathbb{S}) \geq \sum_{\{S \in \mathbb{S}: |K \cap S| = 1\}} y(S) = \sum_{v \in K} \sum_{\{S \in \mathbb{S}: v \in S\}} y(S) \geq \sum_{v \in K} w(v) = \omega(G, w)$$

De modo semelhante, podemos mostrar que $\ell(\overrightarrow{G}, w)$ é um limite superior para $\omega(G, w)$.

Proposição 2 *[Yamaguchi and Masuda(2008)] Para um grafo ponderado $G = (V, A, w)$ e uma seqüencia ρ, $\omega(G, w) \leq \ell(\overrightarrow{G}_\rho)$*

Prova Seja K uma clique máxima em G. Para qualquer sequência ρ, $\overrightarrow{G}_\rho$ tem um caminho que contém todos os vértices em K. Logo, $\omega(G) = w(K) \leq \ell(\overrightarrow{G}_\rho)$.

3. Algoritmo do estado da arte

Uns dos algoritmos mais populares para PCMP é CLIQUER proposto por Östergård [Östergård(1999)]. Ele usa o método de busca que ficou mais tarde conhecido como método de busca das bonecas russas. Dado um grafo $G = (V, E, w)$ e uma ordem inicial dos vértices $\rho = (\rho_1, \ldots, \rho_n)$, os subproblemas resolvidos estão associados com os subgrafos $G_i = (V_i, E_i, w)$, $i \in \{1, \ldots, n\}$, onde $V_i = \{\rho_1, \ldots, \rho_i\}$, $E_i = \{\{u, v\} \in E | u, v \in V_i\}$. A solução ótima de G_i é armazenado em $c[\rho_i]$. Durante o processo de busca, para cada $S \subseteq V$, CLIQUER utiliza duas estratégias para estimar um limite superior para $\omega(G[S], w)$: a primeira usa a informação obtida pelas computações anteriores armazenadas em c; e a segunda toma o somatório dos pesos dos vértices de S, $\sum_{v \in S} w(v)$. Apesar do seu bom resultado em grafos esparsos, sua utilização é comprometida devido a sua performance pobre em grafos densos (com densidade superior a 50%).

Com o objetivo de melhorar a performance em grafos densos, Kumlander propôs um outro algoritmo de Bonecas Russas [Kumlander(2008)], denotado por DK, usando uma heurística de coloração ponderada como procedimento de limite superior. Inicialmente, V é particionado em conjuntos independentes $C_1, C_2, \ldots, C_k$. Durante o processo de busca, para cada $S \subseteq V$, DK utiliza também duas estratégias para obter um limite superior para S: a primeira usa novamente o vetor de soluções ótimas armazenado c; e a segunda utiliza o somatório dos pesos dos vértices mais pesados em cada classe de cor, $\sum_{j=1}^{k} \max\{w(u) : u \in S \cap C_j\}$. Testes computacionais conduzidos em [Shimizu et al.(2012)], mostram que DK supera CLIQUER apenas em grafos aleatórios com densidade superior a 90%.

Em 2008, Yamaguchi e Masuda propuseram um outro algoritmo de branch-and-bound em [Yamaguchi and Masuda(2008)], identificado por YM. Ele usa o peso do caminho mais pesado no grafo $\overrightarrow{G}_\rho$ como procedimento de limite superior. Testes computacionais confirmaram que YM é mais eficiente que DK para grafos aleatórios com densidade superior 50%. Além disso, eles mostraram que o procedimento de limite superior adotado em YM é potencialmente melhor que o limite superior utilizado DK.

Em 2012, Held, Cook e Sewell [Held et al.(2012)] desenvolveram um algoritmo de branch-and-bound, denotaremos por HCS, adotando idéias presentes em [Balas and Xue(1991), Babel(1994), Sewell(1998)]. Este algoritmo foi utilizado durante o processo de geração de colunas para problemas de coloração de grafos. Ele foi comparado com CLIQUER, os algoritmos de branch-and-cut dos resolvedores GUROBI 3.0.0 e CPLEX 12.2. A performance de HCS foi observada em 25 instâncias difíceis de geração de colunas. CLIQUER foi o mais eficiente para instâncias com densidade inferior a 50%. Para as densidades maiores ou iguais 97%, os algoritmos de branch-and-cuts foram os mais rápidos. Por outro lado, CLIQUER falha para instâncias com densidade muito alta enquanto os algoritmos de branch-and-cut apresentam um desempenho pobre em grafos esparsos. Diferente deles, HCS mostra-se competitivo em todas as instâncias por todas as densidades.

4. Algoritmo de Branch-and-Bound

A maioria dos algoritmos combinatórios para PCMP utiliza alguma forma sistemática de enumeração das cliques maximais de G com o auxílio de algum procedimento heurístico para computar um limite superior para cada subproblema. Um subproblema de PCMP é definido pela tripla (C, P, LB), onde LB é o peso da clique mais pesada encontrada até o momento, $C = \{v_1, \ldots, v_d\}$ (onde d é a profundidade do subproblema) é a clique atual construída e P é o conjunto de vértices que podem entrar na clique atual construída, ou seja, $P \subseteq \cap_{i=1}^{d} N(v_i)$, chamado de conjunto candidato.

Se (C, P, LB) não é podado, um vértice v de P é selecionado, para ser adicionado à clique atual. Quando o vértice v é adicionado, dizemos que o vértice v foi ramificado. Todas as cliques maximais contendo o vértice v são enumerados implicitamente no subproblema $(C \cup \{v\}, P \cap N(v), LB)$, depois o vértice v é removido de P.

Assim, em cada subproblema, precisamos particionar o conjunto $P = U \cup R$ tal que $\omega(G[U], w) \leq LB - w(C)$, de modo que apenas os vértices que estão em R são utilizados para o processo de ramificação. Os vértices de R 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. Os seguintes subproblemas serão gerados:

$$
\begin{array}{lcl}
x_{v_n} = 1 \quad \vee & \Leftrightarrow & (C \cup \{v_n\}, P \cap N(v_n), LB) \\
(x_{v_n} = 0 \wedge x_{v_{n-1}} = 1) \quad \vee & \Leftrightarrow & (C \cup \{v_{n-1}\}, (P \setminus \{v_n\}) \cap N(v_{n-1}), LB) \\
(x_{v_n} = 0 \wedge x_{v_{n-1}} = 0 \wedge x_{v_{n-2}} = 1) \quad \vee & \Leftrightarrow & (C \cup \{v_{n-2}\}, (P \setminus \{v_n, v_{n-1}\}) \cap N(v_{n-2}), LB) \\
\vdots & & \vdots \\
(x_{v_n} = 0 \wedge x_{v_{n-1}} = 0 \wedge \ldots \wedge x_{v_1} = 1) & \Leftrightarrow & (C \cup \{v_1\}, (P \setminus \{v_n, \ldots, v_2\}) \cap N(v_1), LB)
\end{array}
$$

Esse esquema de ramificação foi estabelecido primeiramente em [Balas and Yu(1986)].

A estrutura básica para um algoritmo de branch-and-bound para PCMP incorporando o esquema de ramificação de Balas e Yu será apresentada no Algoritmo 1.

Algoritmo 1 Estrutura básica dos algoritmos de branch-and-bound para PCMP

```
1:  Função MAIN
2:      LB ← 0
3:      CLIQUE(∅, V(G), LB)
4:  Função CLIQUE(C, P, LB)
5:      se w(C) > LB então
6:          LB ← w(C)
7:      Particione P = U ∪ R tal que ω(G[U], w) ≤ LB − w(C)
8:      Seja (v_1, ..., v_n) uma ordem dos vértices de R.
9:      para i ← n até 1 faça
10:         se w(C) + UB(P) > LB então
11:             CLIQUE(C ∪ {v_i}, ∩N(v_i), LB)
12:             P ← P \ {v_i}
```

5. Estratégia de procedimento de limite superior baseado na coloração ponderada

A ideia de usar a coloração ponderada para calcular um limite superior para a clique máxima ponderada foi primeiro explorada em [Balas and Xue(1991)]. Uma coloração ponderada pode ser obtida iterativamente gerando conjuntos independentes maximais enquanto uma peso apropriado são atribuídos a eles a fim de cobrir o peso de cada vértice. Para cada vértice, $res(v)$ mantém o peso residual de cada vértice. Inicialmente, $res(v)$ é igual a $w(v)$. Quando $res(v) = 0$, dizemos o vértice v está totalmente colorido.

Na iteração i, um conjunto independente maximal S_i é construído no subgrafo induzido pelos vértices ainda não coloridos, $\{v \in V | res(v) > 0\}$. O peso $y(S_i)$ é definido como o menor peso residual dos vértices em S_i. Depois disso, o peso residual de cada vértice em S_i é decrementado por $y(S_i)$.

No final da iteração i, o par $(\{S_1, \ldots, S_i\}, y)$ satisfaz a seguinte propriedade $res(v) + \sum_{\{i|v \in S_i\}} y(S) = w(v) \forall v \in V$

No Algoritmo 2, nós propomos uma implementação da heurística de coloração ponderada usando vetores de bits. O procedimento tem os seguintes parâmetros: um vetor de bits U, representando os vértices que devem ser coloridos; um vetor ρ, representando uma ordem inicial dos vértices que coincide com a ordem em os bits são organizados no vetor de bits; um vetor π, representando a ordem em que os vértices são efetivamente coloridos

Algoritmo 2 Heurística de coloração ponderada

Função BITCOLOR$(U, \rho, \pi, color)$
  $\forall v \in V, res(v) \leftarrow w(v)$
  $i \leftarrow 1$
  $cont \leftarrow 1$
  $UB \leftarrow 0$
  enquanto $\boxed{U \neq \emptyset}$ **faça**
    $S_i \leftarrow \emptyset$
    $\boxed{Q \leftarrow U}$
    enquanto $\boxed{Q \neq \emptyset}$ **faça**
      $\boxed{v \leftarrow \text{Selecione o primeiro vértice de } Q \text{ seguindo a ordem } \rho.}$
      $\boxed{Q \leftarrow Q \setminus N(v)}$
      $\boxed{Q \leftarrow Q \setminus \{v\}}$
      $S_i \leftarrow S_i \cup \{v\}$
    $min\_res \leftarrow min\{res(v) | v \in S_i\}$
    $y(S_i) \leftarrow min\_res$
    $UB \leftarrow UB + y(S_i)$
    para $v \in S_i$ **faça**
      $res(v) \leftarrow res(v) - y(S_i)$
      se $res(v) = 0$ **então**
        $\boxed{U \leftarrow U \setminus \{v\}}$
        $\pi[cont] \leftarrow v$
        $color[v] \leftarrow UB$
        $cont \leftarrow cont + 1$
    $i \leftarrow i + 1$

pela heurística de coloração e um vetor *color* tal que $color[\pi_i]$ armazena o peso da coloração ponderada para colorir o subgrafo $G[\pi_1, \ldots, \pi_i]$.

Em cada iteração, um conjunto independente S_i é construído. O vetor de bits Q armazena os vértices que podem ser adicionado no conjunto independente atual S_i. Os vértices que são adicionados em S_i são selecionados de Q seguindo a ordem inicial ρ. Quando um vértice v é selecionado, o conjunto independente S_i é atualizado ($S_i \leftarrow S_i \cup \{v\}$). O vértice v e a sua vizinhança vizinhança são removidos do vetor de bits Q ($Q \leftarrow Q \backslash N(v); Q \leftarrow Q \setminus \{v\}$). O processo continua enquanto $Q \neq \emptyset$. Quando Q torna-se vazio, um conjunto independente maximal S_i é obtido. O peso de $y(S_i)$ é dado pelo menor peso residual dos vértices em S_i, e o peso residual dos vértices de S_i é decrementado por $y(S_i)$. Os vértices com peso residual igual a zero são removidos do vetor de bits U. A heurística de coloração ponderada termina quando U torna-se vazio.

Como todos os bits dos vetores de bits são organizados seguindo a ordem ρ. A instrução "selecione o primeiro vertice de Q seguindo ρ" pode ser substituída por "selecione o primeiro bit setado em 1 em Q". As operações destacadas representam operações que podem ser substituídas por operações manipulando vetores de bits.

Nós vamos considerar duas possíveis esquemas de ordenação inicial dos vértices ρ:

- ordem menor peso primeiro é uma uma ordem $(\rho_1, \ldots, \rho_n)$ tal que ρ_1 é o vértice com o menor peso em G. Em caso de empate, o vértice escolhido é aquele com a maior soma dos pesos do seus vértices adjacentes em G. O vértice ρ_i é escolhido seguindo a

mesma regra no grafo $G[V \setminus \{\rho_1, \ldots, \rho_{i-1}\}]$. Essa ordem foi proposta primeiramente em [Östergård(1999)].

- ordem menor grau último é uma ordem $(\rho_1, \ldots, \rho_n)$ tal que ρ_n é o vértice com o menor a soma dos pesos dos seus vértices em G. Em caso de empate, o vértice escolhido é aquele com o maior peso em G. O vértice ρ_i é escolhido seguindo a mesma regra no grafo $G[V \setminus \{v_{i+1,\ldots,v_n}\}]$.

Nós podemos mostrar que a coloração ponderada prover um limite superior potencialmente melhor que o limite obtido pela caminho mais pesado.

Proposição 3 *Seja (G, w) um grafo ponderado. Para qualquer ordem inicial ρ, existe uma coloração ponderada $(\mathbb{S}, y)$ de (G, w) tal que $y(\mathbb{S}) \leq \ell(\overrightarrow{G}_\rho, w)$*

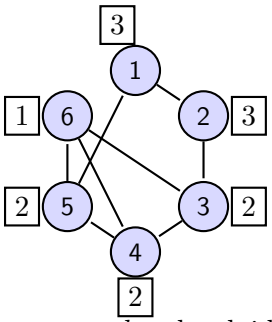
Prova Seja $\mathbb{S} = \{S_1, \ldots, S_k\}$ uma coloração ponderada obtida pelo Algoritmo 2. Podemos mostrar que existe um caminho H em $\overrightarrow{G}_\rho$ contendo um vértice de cada S_i. H é definido percorrendo os conjuntos na seguinte ordem $S_k, \ldots, S_1$. Em S_k, escolha qualquer vértice. Então, para $i = k-1, \ldots, 1$, seja $H = vH'$ um caminho definido depois de percorrer $S_k, \ldots, S_{i+1}$. Se $H \cap S_i \neq \emptyset$, nenhum vértice é escolhido de S_i. Caso contrário, estendemos H para uvH', para $u \in S_i$ tal que $(u, v) \in E(\overrightarrow{G}_\rho)$. Tal vértice existe porque S_i é maximal e foi formado seguindo a ordem ρ. Então,

$$y(\mathbb{S}) = \sum_{S \in \mathbb{S}} y(S) \leq \sum_{v \in H} \sum_{S \in \mathbb{S}: v \in S} y(S) = \sum_{v \in H} w(v) \leq \ell(\overrightarrow{G}_\rho, w) \tag{1}$$

6. Estratégia de Ramificação

Dado um subproblema (C, P, LB), a heurística de coloração ponderada devolve uma ordem total $\pi = (\pi_1, \ldots, \pi_{|P|})$ de P tal que $color[\pi] \leq color[\pi_2] \leq \ldots \leq color[\pi_{|P|}]$, onde $color[\pi_i]$ é o peso da coloração ponderada do subgrafo $G[\pi_1, \ldots, \pi_i]$. Os vértices são escolhidos para a ramificação enquanto $w(C) + color[\pi_i] > LB$, para $i = |P|, \ldots, 1$.

Considere o seguinte grafo ponderado abaixo. Usando a ordem do menor peso primeiro, temos a seguinte ordem inicial $\rho = (6, 5, 4, 3, 2, 1)$. A coloração ponderada devolvida pelo Algoritmo 2 é:



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

Os vetores π e *color* devolvido pela heurística de coloração ponderada são:

π_i	6	5	3	4	2	1
$color[\pi_i]$	1	3	3	5	5	8

Começando com $(C = \emptyset, P = V, LB = 0)$, o primeiro vértice ramificado é o vértice 1. Quando este vértice é ramificado, a clique {1,2} com peso 6 é encontrada. O próximo subproblema seria $(C = \emptyset, P = V \setminus \{1\}, LB = 6)$, seguindo a ordem de ramificação o vértice ramificado é o vértice 2, mas $w(C) + color[2] = 5 < LB = 6$, então o processo de ramificação é abortado.

Algoritmo 3 Algoritmo BITCLIQUE

Função MAIN
$\rho \leftarrow (\rho_1, \ldots, \rho_n)$.
$LB \leftarrow$weight of best clique returned by multi-neighborhood tabu search.
BITCLIQUE$(\emptyset, V(G), LB)$

Função BITCLIQUE(C, P, LB)
se $w(C) > LB$ **então**
$LB \leftarrow w(C)$
`BITCOLOR`(P, ρ, π, $color$)
para $i \leftarrow |P|$ to 1 **faça**
$v \leftarrow \pi[i]$
se $w(C) + color[v] \leq LB$ **então**
retorne
$newP \leftarrow P \cap N(v)$
$P \leftarrow P \setminus \{v\}$;
BITCLIQUE($C \cup \{v\}$, $newP$, LB)

7. Algoritmo BITCLIQUE

O Algoritmo BITCLIQUE efetivamente combina a adaptação de várias ideias já usadas em outros algoritmo para PCMP (Ver Algoritmo 3). Mais especificamente, nós usamos:

- Uma busca tabu com multivizinhança para obter um limite inferior inicial com alta qualidade.
- Uma ordem inicial ρ que desempenha dois papéis importantes:
  1. Definir um critério para escolha dos vértices durante a coloração ponderada.
  2. Estabelecer uma organização dos bits nos vetores de bits.
- Uma versão da heurística de coloração ponderada com paralelismo à nível de bits.
- Uma estratégia de ramificação baseada na coloração ponderada inspirada em [Babel(1994)].

A estrutura de BITCLIQUE está descrita no Algoritmo 3. Em cada subproblema, a heurística de coloração ponderada é executada, devolvendo uma ordem de coloração π e um vetor de limite superior armazenado em *color* que são usados para poda e a ramificação.

8. Testes computacionais

Nós comparamos a performance computacional de BITCLIQUE com o algoritmo CLIQUER[1], YM(gentilmente cedido pelos autores) e HCS[2]. Nós consideramos duas versões do nosso algoritmo, de acordo com a ordem inicial ρ: BITCLIQUE1 usa a ordem de menor peso primeiro enquanto BITCLIQUE2 usa a ordem de menor grau ponderado último.

Nós avaliamos o desempenho do nosso algoritmo utilizando as principais instâncias disponíveis na literatura: grafos aleatórios, DIMACS-W e EXACTCOLOR. Todos os testes foram realizados num computador com processador *Intel Core i7-2600K 3.40Ghz*, 8 Mb de cache, com 6Gb de memória, utilizando o sistema operacional Linux.

[1]disponível em `http://users.aalto.fi/~pat/cliquer.html`
[2]disponível em `https://code.google.com/p/exactcolors/`

8.1. Grafos Aleatórios

Em nosso experimento, nós geramos 10 grafos aleatórios para cada combinação n (número de vértices) e p (probabilidade de existência de cada aresta) considerada. Nós estamos usando o modelo de grafos aleatórios $G(n, p)$. Neste modelo, o grafo é construído adicionando arestas aleatoriamente. Cada aresta é incluída no grafo com probabilidade p, independentemente de todas as outras arestas. Logo, todos os grafos com n vértices e m arestas têm a mesma probabilidade de serem gerados definida por

$$p^m(1-p)^{\frac{n(n-1)}{2}-m} \tag{2}$$

Os pesos atribuídos aos vértices estão uniformemente distribuído no intervalo entre 1 e 10. Nós estamos usando dois geradores de números pseudo-aleatório: *drand*, que devolve um número em ponto flutuante de precisão dupla uniformemente distribuído no intervalo [0,1) e *urand*, que retorna um inteiro sem sinal uniformemente distribuído no intervalo $0, .., 2^{24} - 1$.

Como é usual na literatura, o número de vértices das instâncias diminui à medida que a probabilidade de existência de aresta aumenta. Consideramos 20 combinações, levando a 200 instâncias no total. Para cada par (n, p), calculamos a média do número de subproblemas na árvore de *B & B* e a média do tempo consumido pelas 10 instâncias para cada algoritmo. O tempo de limite de resolução para cada instância é 1h (3600s). Quando o tempo limite é ultrapassado, tal ocorrência é assinalada na tabela como $*^x$, onde $x \in [1, 10]$ significa o número de instâncias não resolvidas no tempo limite. Identificamos em negrito o melhor desempenho médio em cada caso.

A Tabela 1 mostra que BITCLIQUE1 supera todos os outros algoritmos para instâncias de grafos aleatórios com densidade superior a 50%. Esta diferença torna-se mais significativa a medida que a densidade do grafo aumenta. BITCLIQUE1 é competitivo para as densidade entre 40% e 50% e mais eficiente para no intervalo entre 60% e 80%. A partir de 90%, o algoritmo BITCLIQUE2 torna-se mais eficiente que os demais, indicando que a ordem inicial do menor grau ponderado último é mais apropriada para as instâncias com a densidade mais alta.

Para as instâncias DIMACS-W e EXACTCOLOR, o tempo limite de resolução utilizado para cada instância é 2h (7200s). Quando o tempo limite é ultrapassado, tal ocorrência é assinalado como * na tabela. Identificamos em negrito o melhor desempenho em cada caso.

8.2. DIMACS-W

As instâncias DIMACS-W são obtidas a partir das instâncias DIMACS. Muitas heurísticas para PCMP usam as instâncias DIMACS-W como instâncias de referência. Teoricamente, existem várias maneiras de definir uma função de ponderação dos vértices. Vamos adotar a função de ponderação descrita em [Pullan(2008)]: para cada vértice i, w_i é igual a $(i \bmod 200) + 1$.

A Tabela 2 apresenta os resultados computacionais para essas instâncias. Os novos algoritmo BITCLIQUE1 e BITCLIQUE2 são frequentemente mais rápidos que os anteriores. Existem reduções significativas, como no caso das instâncias *phat_1000-2* e *phat_1500-2*. Notamos também que os novos algoritmos conseguiram resolver mais instâncias dentro do tempo limite estabelecido.

8.3. EXACTCOLOR

Estas instâncias são geradas durante o processo de geração de colunas para problemas de coloração de grafos. Em [Held et al.(2012)], um conjunto de instâncias de PCMP é apresentado[3].

[3]O conjunto completo pode ser obtido em `http://code.google.com/p/exactcolors/wiki/MWISInstances`.

Instância	Tempo de execução(segundos)				
(n,p)	**CLIQUER**	**YM**	**HCS**	**BITCLIQUE1**	**BITCLIQUE2**
(2500,0.10)	**0.09**	0.27	5.43	0.77	0.81
(5000,0.10)	**0.54**	3.06	94.33	3.25	4.80
(2500,0.20)	**0.53**	3.14	32.44	2.00	3.07
(5000,0.20)	**8.34**	80.12	900.53	42.11	77.35
(2500,0.30)	**8.21**	54.46	370.41	23.46	50.48
(5000,0.30)	**303.90**	2988.57	$*^{10}$	1260.86	3289.24
(1200,0.40)	**3.37**	17.01	73.74	4.94	9.22
(2500,0.40)	**239.24**	1463.14	$*^{10}$	464.34	1094.99
(600,0.50)	1.09	3.73	10.44	**0.91**	1.39
(1200,0.50)	**75.27**	357.71	1221.97	79.12	171.66
(600,0.60)	17.63	55.33	127.39	**9.54**	16.46
(1200,0.60)	$*^{8}$	$*^{10}$	$*^{10}$	**2730.49**	$*^{10}$
(400,0.70)	22.51	52.32	78.04	**6.75**	9.52
(500,0.70)	189.15	411.20	685.53	**53.97**	89.38
(600,0.70)	858.35	1988.08	$*^{7}$	**254.26**	484.17
(300,0.80)	159.94	159.30	167.96	15.42	**14.93**
(400,0.80)	$*^{2}$	$*^{5}$	$*^{9}$	**313.46**	380.15
(200,0.90)	459.22	100.66	36.16	5.46	**1.12**
(300,0.90)	$*^{10}$	$*^{10}$	$*^{10}$	1924.74	**480.46**
(200,0.95)	$*^{10}$	1582.71	34.36	23.75	**0.61**

Tabela 1: Tempo de execução para instâncias de grafos aleatórios

Tabela 3 apresenta o tempo de execução para as instâncias mais difíceis EXACT-COLOR. BITCLIQUE1 e BITCLIQUE2 apresentam um performance melhor ou equivalente dentre todas as densidades consideradas. Mais ainda, BITCLIQUE1 resolve mais instâncias que BITCLIQUE2. Existem reduções significativas em *DSJC250.1*.

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.

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

Balas, E. and Yu, C. S. (1986). Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068.

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.

Corrêa, R. C., Marenco, J., Delle Donne, D., and Koch, I. (2015). A strengthened general cut-generating procedure for the stable set polytope. *Electronic Notes in Discrete Mathematics*, 50:261–266.

Gualandi, S. and Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100.

Instância		Tempo de execução(segundo)				
Name	**(n,p)**	**CLIQUER**	**YM**	**HCS**	**BITCLIQUE1**	**BITCLIQUE2**
san1000	(1000,0.50)	$> 2h$	4.85	3.34	**2.21**	2.57
brock400_1	(400,0.75)	167.10	254.19	369.78	**29.72**	38.73
brock400_2	(400,0.75)	235.03	282.84	433.53	**39.61**	58.49
brock400_3	(400,0.75)	187.99	246.18	394.57	**30.53**	43.56
brock400_4	(400,0.75)	185.32	127.91	201.43	**16.29**	23.02
brock800_1	(800,0.65)	1060.59	2192.97	5542.95	**400.76**	946.47
brock800_2	(800,0.65)	1077.23	3101.38	$> 2h$	**640.83**	1458.54
brock800_3	(800,0.65)	1138.74	2485.99	6456.42	**470.91**	1064.17
brock800_4	(800,0.65)	1332.39	3156.93	$> 2h$	**644.40**	1539.57
p_hat500-2	(500,0.50)	3.50	9.10	7.69	0.59	**0.10**
p_hat700-2	(700,0.50)	101.23	444.35	186.28	19.67	**0.29**
p_hat1000-2	(1000,0.49)	3117.21	$> 2h$	4780.87	750.75	**10.09**
p_hat1500-2	(1500,0.51)	$> 2h$	$> 2h$	$> 2h$	$> 2h$	**624.42**
sanr400_0.7	(400,0.70)	19.75	40.07	62.56	**6.26**	8.90
san400_0.7_1	(400,0.70)	$> 2h$	147.24	2.79	**2.36**	5.19
san400_0.7_2	(400,0.70)	$> 2h$	22.94	6.50	**3.35**	4.73
san400_0.7_3	(400,0.70)	$> 2h$	12.39	11.97	2.34	**0.86**
p_hat300-3	(300,0.74)	10.36	8.96	6.49	0.66	**0.11**
p_hat500-3	(500,0.75)	$> 2h$	$> 2h$	2369.84	328.54	**5.17**
gen200_p0.9_44	(200,0.90)	373.35	68.68	20.03	3.24	**0.52**
gen200_p0.9_55	(200,0.90)	958.72	36.25	11.69	1.26	**0.37**
gen400_p0.9_55	(400,0.90)	$> 2h$	$> 2h$	$> 2h$	$> 2h$	**3595.54**
gen400_p0.9_75	(400,0.90)	$> 2h$	$> 2h$	$> 2h$	$> 2h$	**363.60**
C250.9	(250,0.90)	2707.85	434.03	242.60	21.52	**4.20**
san200_0.9_1	(200,0.90)	$> 2h$	5.15	**0.38**	0.58	0.50
san200_0.9_2	(200,0.90)	118.11	28.53	5.08	0.55	**0.49**
san200_0.9_3	(200,0.90)	1278.10	119.80	36.23	8.25	**1.64**
san400_0.9_1	(400,0.90)	$> 2h$	$> 2h$	1205.10	**78.08**	$> 2h$
sanr200_0.9	(200,0.90)	621.94	60.86	23.50	4.69	**1.08**
hamming10-2	(1024,0.99)	708.93	$> 2h$	**0.10**	0.66	$> 2h$
MANN_a27	(378,0.99)	$> 2h$	$> 2h$	$> 2h$	$> 2h$	**1.28**
		Número de instâncias resolvidas				
		20	23	25	27	29

Tabela 2: Tempo de execução para as instâncias DIMACS-W

Held, S., Cook, W., and Sewell, E. C. (2012). Maximum-weight stable sets and safe lower bounds for graph coloring. *Math. Program. Comput.*, 4(4):363–381.

Kumlander, D. (2008). On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. 14:165–174.

Mehrotra, A. and Trick, M. A. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354.

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.

Rebennack, S., Oswald, M., Theis, D., Seitz, H., Reinelt, G., and Pardalos, P. (2011). A branch and cut solver for the maximum stable set problem. *Journal of Combinatorial Optimization*, 21(4):434–457.

Instâncias	(n,p)	Tempo de Execução(segundos)				
		CLIQUER	YM	HCS	BITCLIQUE1	BITCLIQUE2
DSJC250.5	(250,0.50)	**0.08562**	0.09572	0.53295	0.09947	0.09844
DSJC500.5	(500,0.50)	5.51194	5.89928	28.16698	**3.25946**	3.29579
DSJC1000.5	(1000,0.50)	733.11313	693.66427	3091.07580	**449.52101**	458.15096
C2000.5.1029	(2000,0.50)	*	*	*	*	*
flat300_28_0	(300,0.52)	0.36160	0.44294	1.80078	**0.29889**	0.29897
flat1000_50_0	(967,0.51)	**36.58473**	159.55484	673.98914	58.26586	103.86839
flat1000_60_0	(999,0.51)	**187.10821**	409.66139	1674.89864	194.59528	252.22985
flat1000_76_0	(1000,0.51)	935.53172	863.26174	3940.79916	**586.53005**	**572.81380**
school1	(336,0.71)	48.64885	39.23673	12.60694	**2.45248**	**0.66964**
DSJC250.1	(241,0.89)	*	5831.52009	5261.10567	**717.33390**	**592.85220**
DSJC500.1.117	(494,0.90)	*	*	*	*	*
DSJC1000.1.3915	(998,0.90)	*	*	*	*	*
2-Insertions_4	(149,0.95)	29.37893	0.38999	**0.06504**	0.19995	0.29629
1-Insertions_6	(527,0.96)	*	2242.33236	44.24440	**1.78066**	24.36488
2-Insertions_5	(369,0.97)	*	6089.65302	85.43386	**7.03292**	**6.52866**
3-Insertions_4	(208,0.97)	*	94.30376	**0.60412**	1.09367	**0.26069**
4-Insertions_4	(295,0.98)	*	4083.37884	11.38059	**4.16241**	67.18725
3-Insertions_5	(460,0.98)	*	*	*	**1887.78702**	*
		Número de instâncias resolvidas				
		9	14	14	15	14

Tabela 3: Tempo de execução para instâncias Exactcolor

Sewell, E. C. (1998). A branch and bound algorithm for the stability number of a sparse graph. *INFORMS J. on Computing*, 10(4):438–447.

Shimizu, S., Yamaguchi, K., Saitoh, T., and Masuda, S. (2012). Some improvements on kumlander-s maximum weight clique extraction algorithm. 6(12):1770 – 1774.

Östergård, P. R. (1999). A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3(0):153 – 156. 6th Twente Workshop on Graphs and Combinatorial Optimization.

Östergård, 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.

Warrier, D., Wilhelm, W. E., Warren, J. S., and Hicks, I. V. (2005). A branch-and-price approach for the maximum weight independent set problem. *Networks*, 46(4):198–209.

Wu, Q. and Hao, J.-K. (2014). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, (0):–.

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.